ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ЭВОЛЮЦИОННЫЕ МЕТОДЫ МОДЕЛИРОВАНИЯ И ОПТИМИЗАЦИИ СЛОЖНЫХ СИСТЕМ

Конспект лекций

Авторы-составители:
Е.С. Семенкин, М.Н. Жукова, В.Г. Жуков, И.А. Панфилов, В.В. Тынченко

КРАСНОЯРСК 2007

СТАНДАРТНЫЕ ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ

ВВЕДЕНИЕ

Вопрос: Что является самым мощным средством при решении проблем (задач) во Вселенной?

Ответ 1: Человеческий мозг, который создал «колесо, Нью Йорк, войны и т.д.» (по Дугласу Адамсу)

Ответ 2: Эволюционный механизм, который создал человеческий мозг (по Дарвину).

В основе большинства концепций искусственного интеллекта лежит богатое разнообразие природных явлений. Один из примеров – искусственные нейронные сети, основная идея которых основывается на функционировании нейронов мозга. ГА являются направлением более общей теории ЭА, основанной на следующем принципе: «каждый биологический вид целенаправленно развивается и изменяется для того, чтобы наилучшим образом приспособиться к окружающей среде».

В 1975 г. вышла основополагающая книга Дж. Холланда “Адаптация в естественных и искусственных системах”, в которой был предложен первый генетический алгоритм. Термин "генетические алгоритмы" ввел в 1975 г. Д. Голдберг. Его книга содержит детальное обсуждение, как теоретических аспектов ГА, так и возможных областей их применения:

- искусственная жизнь,
- автоматическое обучение,
- извлечение данных (знаний),
- инженерное проектирование,
- планирование и управление,
- моделирование, идентификация,
- численная и комбинаторная оптимизация.

ЭА базируются на коллективном обучающем процессе внутри популяции индивидуумов, каждый из которых представляет собой поисковую точку в пространстве допустимых решений данной задачи (рис.1). Популяция случайно инициализируется, и затем охватывает лучшие регионы поискового пространства посредством случайных процессов селекции, мутации и рекомбинации. Окружающая среда представляет качественную

информацию (степень пригодности) о поисковых точках (индивидуумах), а процесс селекции отбирает тех индивидуумов, у которых значение пригодности выше. Отобранные потомки являются, в свою очередь, родителями в следующем поколении. Механизм рекомбинации перемешивает генетическую информацию родителей (тем самым рождается один или несколько потомков), и наконец, механизм мутации способствует в некоторой степени обновлению генетической информации потомков.

Методологическая основа ГА зиждется на гипотезе селекции, которая в самом общем виде может быть сформулирована так: чем выше приспособленность особи, тем выше вероятность того, что в потомстве, полученном с ее участием, признаки, определяющие приспособленность, будут выражены еще сильнее.

Т.о., ГА заимствуют из биологии:

- понятийный аппарат;
- идею коллективного поиска экстремума при помощи популяции особей;
- способы представления генетической информации;
- способы передачи генетической информации в череде поколений (генетические операторы);
- идею о преимущественном размножении наиболее приспособленных особей.

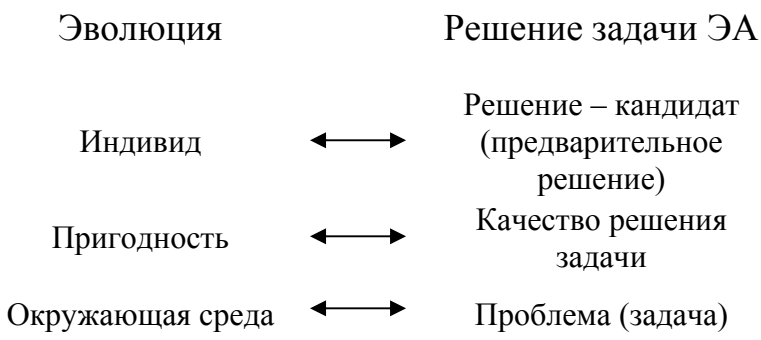


Рис. 1. Метафоры

В общем виде работу генетического алгоритма можно представить следующим образом (рис. 2):

1. Инициализировать случайным образом популяцию решений.
2. С помощью оператора селекции выбрать часть популяции (родителей) для порождения потомков.
3. Применить оператор скрещивания и получить потомков.
4. Новые решения (потомки) подвергаются мутации.
5. Формируется новая популяция: выбрать решения из родителей и потомков.
6. Повторять 2 – 5 пока не выполнится условие остановки.

Производительность ЭА:

- Приемлемая производительность по приемлемой цене для широкого диапазона задач
- Внутренний параллелизм (ошибкоустойчивость, терпимость ошибки)
- Наибольший эффект достигается на сложных задачах, ЭА превосходят другие методы

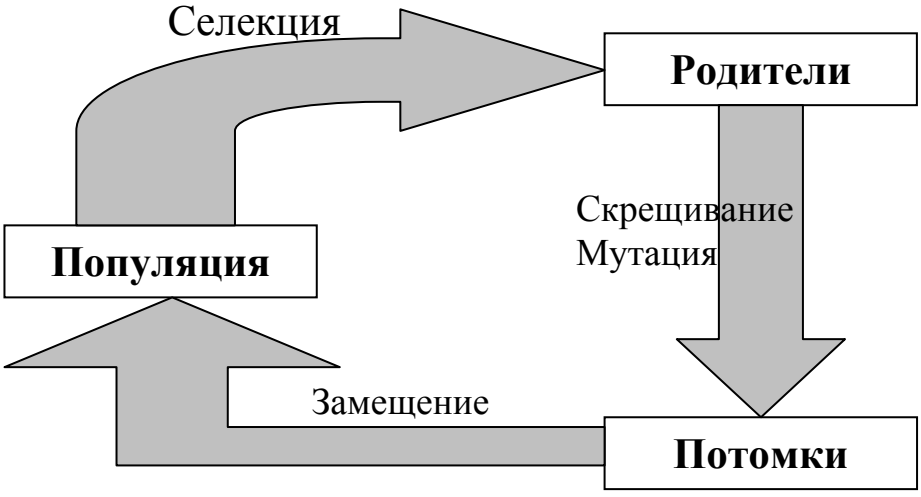


Рис.2. Эволюционный цикл

Преимущества ЭА:

- Не предполагает использования пространства задачи (т.к. происходит кодирование решений в хромосому)
- Широко применимы

- Низкая цена разработки и применения
- Легко объединить с другими методами
- Решения могут быть интерпретированы (в отличии от нейронных сетей)
- Может выполняться интерактивно – можно включить в популяцию предложенные пользователем решения
- Обрабатывается большое количество альтернативных решений

Недостатки ЭА:

- Нет гарантии нахождения оптимального решения за конечное время
- Слабая теоретическая база
- Может потребоваться настройка параметров
- Часто в вычислительном отношении дорогой метод, то есть медленный

ОСНОВЫ ПРОЕКТИРОВАНИЯ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ

Для построения эволюционного алгоритма необходимо выполнить следующие шаги:

- Выбрать представление решений (способ кодирования фенотипа в генотип)
- Решить, как инициализировать популяцию
- Определить основные генетические операторы:
    - Выбрать подходящий оператор мутации
    - Выбрать подходящий оператор скрещивания
- Выбрать способ оценки пригодности индивида
- Решить, как управлять нашей популяцией:
    - Решить, как выбрать индивидов-родителей
    - Решить, как выбрать индивидов для замены
- Решить, когда остановить алгоритм

Проектирование представления

Подобно тому, как природный хромосомный материал представляет собой линейную последовательность различных комбинаций четырех нуклеотидов, решения в ГА представляются в виде хромосом (генотипов). Генотип – строка конечной длины, состоящая из генов, представленных символами некоторого алфавита.

В ГА существует строгое различие между фенотипом (решением, выраженным в терминах поставленной задачи) и генотипом (хромосомой, представлением решения). ГА работает с генотипом, фенотип служит для определения пригодности индивида (оценки качества решения поставленной задачи), поэтому для работы алгоритма необходимо определить некоторую функцию кодирования

$e : D \to S$,

где D - пространство поиска, S - пространство представлений решений)

и функцию декодирования

$e^{-1} : S \to D$.

Таким образом, на самом деле ГА решают не задачу

$f(d) \to \underset{d \in D}{opt}$, где $f : D \to R^1$,

а задачу

$\mu(s) \to \underset{s \in S}{opt}$,

где $\mu : S \to R^1$ и $s = e(x),\ \mu(s) = f(e^{-1}(s)) = f(x)$.

Очевидно, что существует много способов кодирования решений, поэтому при проектировании решений необходимо помнить следующее:

- способ, который мы выбираем, должен соответствовать проблеме, которую мы решаем
- следует иметь в виду, как генотипы будут оцениваться
- следует иметь в виду, какие генетические операторы должны применяться

Дискретное представление. Представление индивида может использовать дискретные значения (бинарные, целые числа, или любые другие системы с дискретным множеством значений). Ниже приведен пример двоичного представления (рис. 3-4).

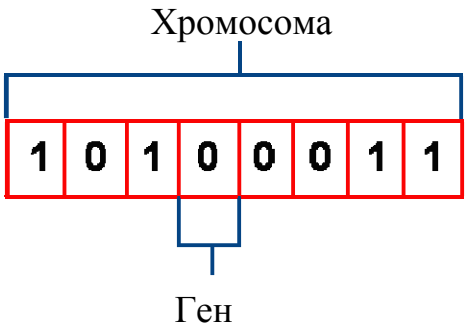


Рис.3. Дискретное представление (двоичный алфавит)

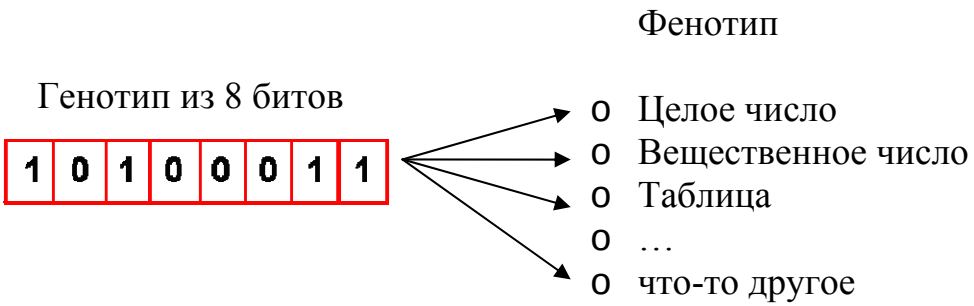


Рис. 4. Дискретное представление (двоичный алфавит)

Вещественное представление - самое подходящее кодирование в случаях, когда решение мы ищем в списке вещественных чисел. Индивиды представлены как кортеж из n вещественных чисел

$$X = [x_1 \quad x_2 \quad \ldots \quad x_n], x_i \in R$$

Порядковое представление. Индивиды представлены как перестановки. Обычно такое кодирование используется в задачах упорядочения/планирования (пример: Задача Коммивояжера, где каждому городу соответствует уникальное число от 1 до n. Решение может быть следующим (5, 4, 2, 1, 3)). При порядковом представлении требуются специальные операторы для удостоверения правильности перестановки.

Представление на основе дерева. Индивиды в популяции – деревья (рис.5). Любое S-выражение может быть представлено как дерево функций и термов. Обычно термы – входные данные задачи, функции – математические функции, процедуры, алгоритмы обработки входных данных (рис.6). Такие функции и термы могут быть:

- Функции: sine, cosine, add, sub, and, If-Then-Else, Turn...

- Термы: X, Y, 0.456, true, false, p, Sensor1…

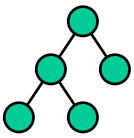


Рис.5. Дерево решения

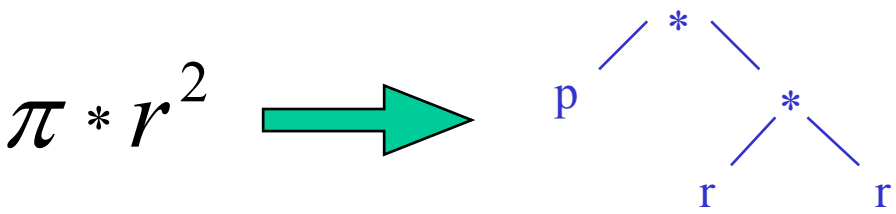


Рис.6. Пример: вычисление площади круга

Существуют и другие способы представления решений. Однако, на практике наибольшее распространение получили ГА с бинарным представлением решений. Рассмотрим метод бинаризации для задачи безусловной параметрической оптимизации.

Метод бинаризации. В общем виде задачу безусловной оптимизации можно представить в виде (1.1), где переменные могут быть выражены в различных шкалах измерений.

$$\chi(X) \to \underset{X \in R \times F \times N \times P}{extr},$$
$$\chi : R \times F \times N \times P \to R^1, \qquad (1.1)$$

где R - область изменения метрических переменных, F - область изменения порядковых переменных, N - область изменения переменных наименований, P - область изменения переменных перестановок.

Предложенный Л.А. Растригиным метод бинаризации позволяет сводить задачи типа (1.1) к задачам псевдобулевой оптимизации (1.2). Следует отметить, что к задаче (1.2) сводятся любые задачи дискретного программирования, а также непрерывного, путем дискретизации пространства поиска.

$$\chi(X) \to \underset{X \in B_{2^n}}{extr},$$
$$\chi : B_{2^n} \to R^1, B_{2^n} = (X : x_j \in B_2, j = \overline{1,n}), B_2 = \{0,1\}. \qquad (1.2)$$

Основная идея бинарного кодирования заключается в следующем. Булев вектор n компонент, может принимать 2^n - значений. Необходимо каким-либо образом установить соответствие между значениями переменных исходной задачи и значениями булевого вектора (задать алгоритм кодирования). В случае, если число значений переменных исходной задачи не равно 2^n, то либо переопределяют область изменения исходных переменных, либо одному значению исходной переменной ставят в соответствие несколько значений булевого вектора.

Перевод порядковых переменных в булевы

Пусть дана порядковая переменная $f \in \{a_1,...,a_q\}$. Бинарным представлением переменной f будет вектор $X = (x_1,...,x_n),\ x_i = 0 \wedge 1,\ i = \overline{1,n}$, где $n = \log_2 q$. Находим значение булевого вектора, переведя его в целое число $k = \sum_{j=1}^{n} x_j \cdot 2^{j-1},\ k \in [1;q]$. Тогда полагаем $f = a_k$.

Перевод метрических переменных в булевы

Кодирование метрической шкалы начинается с перевода ее в порядковую. Для этого интервал $[a,b]$ разбивается на $h = (b-a)/\varepsilon$ равных частей (ε - точность решения задачи оптимизации). Теперь переменная r измеряется в порядковой шкале, $r \in \{y_1,...,y_{h+1}\}$, где $y_i = a + \varepsilon \cdot (i-1), i = \overline{1,h+1}$. Далее порядковая переменная преобразуется в булеву как показано выше.

Перевод переменных наименований и перестановок в булевы

Эффективных алгоритмов перевода порядковых переменных и переменных наименований до сих пор не предложено. Непосредственный перевод их в булевы переменные дает результат аналогичный полному перебору.

Предложено по мере накопления информации о функционале в процессе оптимизации, определять отношения порядка на множестве перестановок и наименований. Далее работать так же как и с порядковыми шкалами.

При кодировании вещественных параметров возникает проблема противоречия точность-размерность. Динамическое кодирование позволяет решить данную проблему следующим образом: в начале осуществляется грубый поиск с малой точностью (малой размерностью булевого вектора),

затем поиск ограничивается некоторой перспективной областью (оператор увеличения), параметры перекодируются с более высокой точностью. Оператор увеличения может применяться многократно.

Код Грея это бинарный код целого числа, такой, что для смежных целых чисел расстояние между их кодами в метрике Хемминга равно *1*.

Стандартное рефлексивное Грей кодирование целого числа получается применением оператора исключающего ИЛИ к стандартному бинарному кодированию целого числа и к тому же бинарному коду, смещенному на одну позицию вправо, последний бит отсекается.

Кодирование и декодирование Грея можно выразить в виде операций с матрицами. Пусть *x* и *y* *n*-битовое бинарное и Грей кодирование целого числа соответственно, и G–матрица трансформации, состоящая из *1* на главной диагонали и диагонали верхнего минора и *0* в других позициях. Грей кодирование и декодирование тогда можно представить как $x^T G$ и $y^T G^{-1}$ соответственно. Можно показать, что любая перестановка столбца в матрице G приводит к другой матрице трансформации.

На практике можно использовать следующий алгоритм перевода бинарного кода в код Грея и наоборот: начать со старшего (правого) бита для получения Грей кода и с левого для получения бинарного кода, последовательно заменять биты по правилу:

$$(...,1,1,...) \rightarrow (...,1,0,...)$$
$$(...,1,0,...) \rightarrow (...,1,1,...)$$
$$(...,0,1,...) \rightarrow (...,0,1,...)$$
$$(...,0,0,...) \rightarrow (...,0,0,...)$$

На каждом шаге заменяется два бита и производится смещение на один бит таким образом, что на последующем шаге заменяется один новый бит и один старый.

Пример:

$X = (10011)$ - бинарное представление целого числа, Y - его Грей код.

а) Применение исключающего ИЛИ (XOR).

$$XOR = \begin{cases} 1, if\ x_1 = 1, x_2 = 0 \\ 0, if\ x_1 = 0, x_2 = 1 \\ 0, if\ x_1 = 1, x_2 = 1 \\ 1, if\ x_1 = 0, x_2 = 0 \end{cases}$$

$$\begin{array}{r} 10011 \\ xor\ \ 01001 \\ \hline Y=(11010) \end{array}$$ и обратно $$\begin{array}{r} 11010 \\ xor\ \ 01001 \\ \hline x=(10011) \end{array}$$.

b) Матричное кодирование.

$$G=\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Y=X^T\cdot G=(10011)\cdot\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}=(11010)^T$$

$$X=Y^T\cdot G^{-1}=(11010)\cdot\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}=(10011)^T$$

c) Алгоритм последовательной замены битов.

$$\begin{array}{rr} X=(100\underline{11}) & Y=(\underline{11}010) \\ 10\underline{01}0 & 1\underline{00}10 \\ 1\underline{00}10 & 10\underline{01}0 \\ \underline{10}010 & 100\underline{10} \\ \hline Y=(11010) & \hline X=(10011) \end{array}$$ и

Для многих практических задач код Грея оказывается более эффективным, чем бинарное кодирование. Код Грея частично сохраняет структуру окрестностей пространства поиска в бинаризованном пространстве. В результате, Грей код не может образовать оптимумов больше, чем у исходной целевой функции. Более того, т.к. у текущей точки в коде Грея соседних точек больше, чем в дискретной целевой функции, то в пространстве поиска Грей кода число оптимумов обычно меньше. В противоположность, бинарный код часто создает новые оптимумы, которых у исходной целевой функции не было. Пример различных систем окрестностей показан на рис. 7.

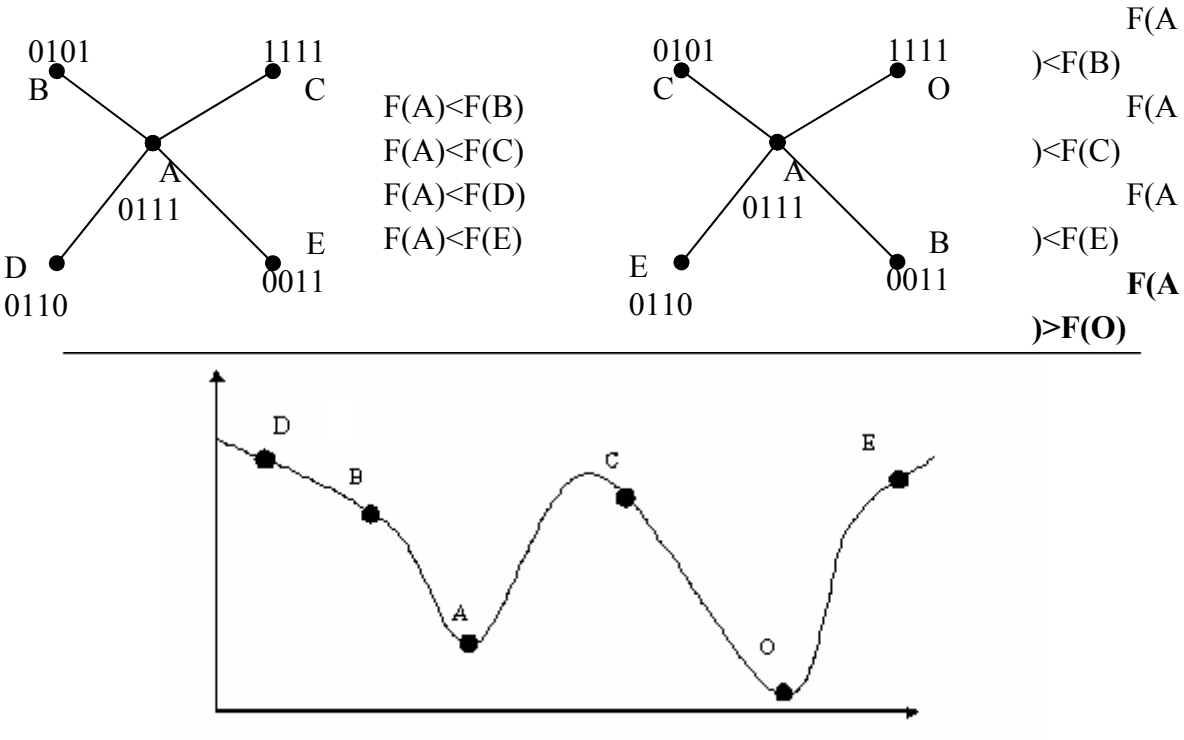


Рис.7. Различные системы окрестностей при бинарном кодировании

Получение фенотипа из генотипа

Иногда получение фенотипа из генотипа это простой и очевидный процесс (например, вычисление значения функции декодирования). В то же время генотип может быть рядом параметров для алгоритма, который, обрабатывая исходные данные, производит фенотип (рис.8).

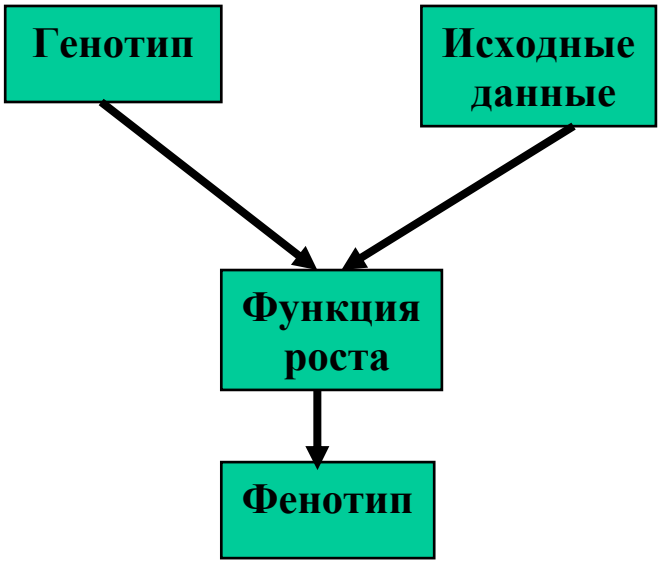


Рис.8. Получение фенотипа из генотипа

Инициализация

Если априорная информация о пространстве поиска отсутствует, начальная популяция должна быть инициализирована равномерно в пространстве поиска (если это возможно). Причем, индивиды могут быть распределены равномерно как в пространстве генотипов, так и в пространстве фенотипов. Примеры:

- Двоичное представление: генерировать 0 или 1 с вероятностью ½;
- Вещественное представление: генерировать решение равномерно на заданном интервале.

Если существуют некоторые предположения о пространстве поиска или известны некоторые варианты решения задачи, можно распределить популяцию в соответствие с имеющимися эвристиками или результатами предыдущих опытов. Однако, возможны следующие негативные последствия:

- Потеря генетического разнообразия;
- Непоправимое смещение хода эволюции.

Основные генетические операторы. Мутация

Оператор мутации – одноместный оператор поиска, случайное изменение в одном или нескольких генах индивида. В ГА мутация рассматривается как метод восстановления потерянного генетического материала, а не как поиск лучшего решения. Обычно мутация применяется к генам с очень низкой вероятностью $p_m \in [0.001; 0.01]$.

Возможно использование одного или нескольких операторов мутации в алгоритме для выбранного типа представления. Необходимо учитывать следующие важные моменты:

- По крайней мере один оператор мутации должен быть в алгоритме;
- Оператор мутации должен позволять достигнуть любой части пространства поиска;
- Величина мутации важна и должна быть управляемой;
- Мутация должна производить допустимые решения.

Примеры:

Мутация для бинарного (дискретного) представления

Мутация обычно происходит с вероятностью p_m для каждого гена. Хорошим эмпирическим правилом считается выбор вероятности мутации равным $p_m = 1/n$, где n - число генов в хромосоме (в среднем хотя бы один ген будет подвержен мутации). В случае бинарного алфавита мутация состоит в инвертировании случайно выбранных битов (рис. 9).

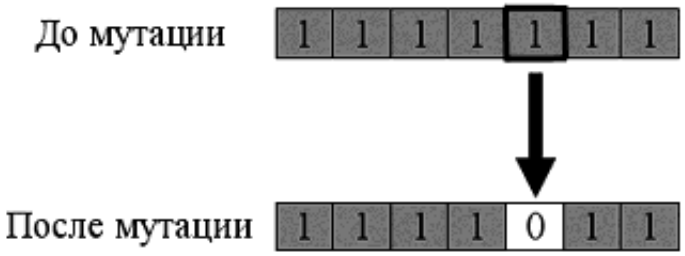


Рис. 9. Пример мутации в ГА для бинарного представления

Мутация для представления с вещественными значениями

Изменение значений происходит путем добавления какого-либо случайного шума. Часто используется нормальное (Гауссовское) распределение N(0,σ), где 0- математическое ожидание, σ - среднеквадратичное отклонение. Тогда оператор мутации имеет вид:

$\hat{x}_i = x_i + N(0, \sigma_i)$.

Мутация для представления, основанного на порядке

Обычно осуществляется случайная перестановка генов (рис. 10).

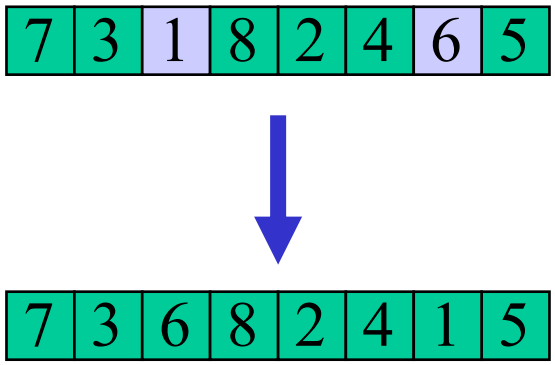


Рис. 10. Пример мутации в ГА для представления, основанного на порядке

Мутация для представлений, основанных на деревьях

Одноточечная мутация – случайно выбирается один узел и заменяется узлом аналогичного типа (рис. 11).

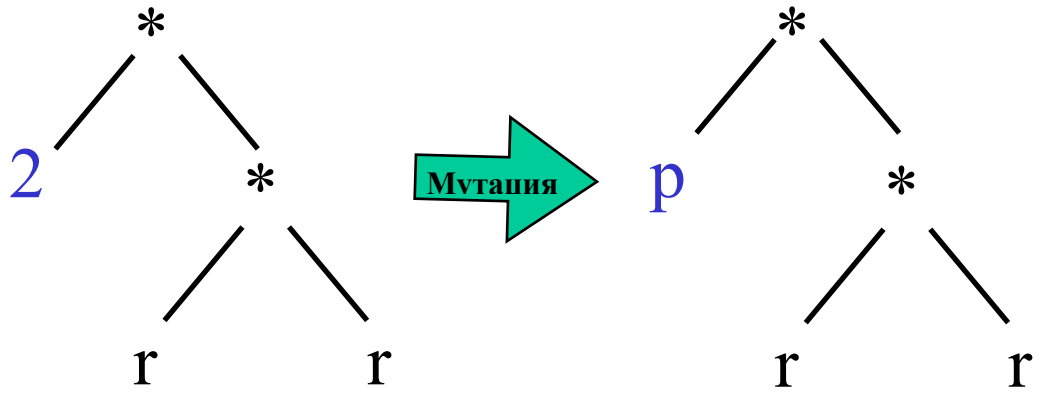


Рис. 11. Пример мутации в ГА для представления, основанного на деревьях

Основные генетические операторы. Скрещивание

Оператор скрещивания (рекомбинация) – генетический оператор поиска. При скрещивании отобранные индивиды (родители) по заданному правилу передают части своих хромосом. Потомок может унаследовать только те гены, которые есть у его родителей. Существуют различные схемы скрещивания: точечное, равномерное, скрещивание более чем двух родителей и другие. При точечном скрещивании для выбранных родителей выбираются точки разрыва хромосомы, родители определенным образом обмениваются соответствующими участками хромосом. При равномерном – соответствующий ген потомка может быть унаследован от любого родителя с равной вероятностью. Скрещивание осуществляется с вероятностью p_{xover}, иначе с вероятностью $(1 - p_{xover})$ родители клонируются в следующее поколение.

Возможно использование одного или нескольких операторов скрещивания для выбранного представления. Необходимо учитывать следующие важные моменты:

- Потомок должен унаследовать гены от каждого родителя. В противном случае оператор скрещивания становится оператором мутации;
- Оператор скрещивания может быть разработан с учетом представления, чтобы скрещивание не было всегда катастрофичным (чтобы потомки не были всегда хуже родителей);
- Скрещивание должно производить допустимые решения.

Типы скрещивания для дискретного представления

Наиболее популярным типом скрещивания является *одноточечное скрещивание* – случайно выбирается точка разрыва, родительские хромосомы разрываются в этой точке и обмениваются правыми частями (рис. 12).

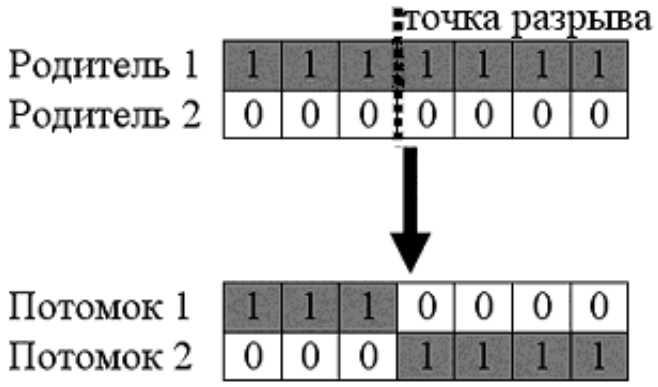


Рис. 12. Пример одноточечного скрещивания

При двухточечном скрещивании хромосома как бы замыкается в кольцо, выбираются 2 точки разрыва, родители обмениваются частями (рис. 13).

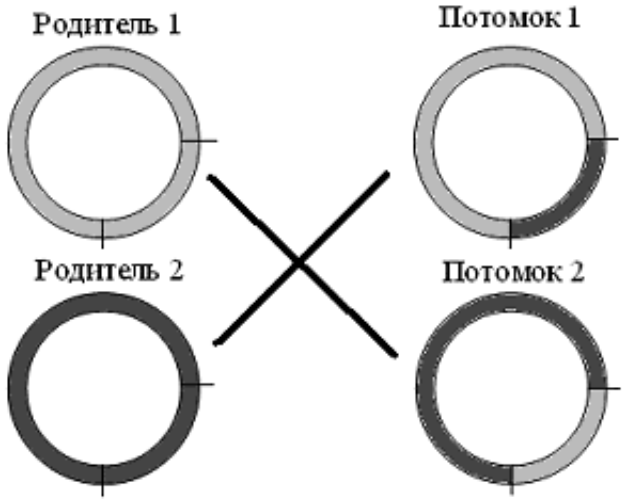


Рис. 13. Пример двухточечного скрещивания

При равномерном скрещивании потомок может унаследовать с равной вероятностью гены любого из родителей (рис. 14).

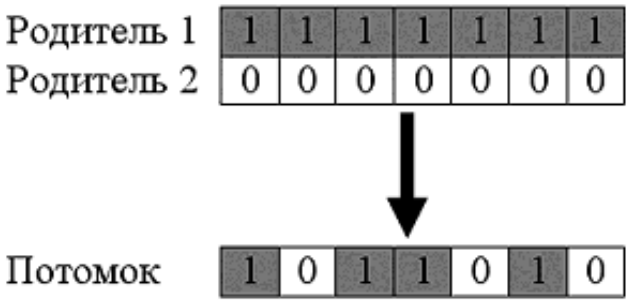


Рис. 14. Пример равномерного скрещивания

Равномерное скрещивание по всей популяции (uniform gene pool recombination) получается применением равномерного скрещивания ко всем членам популяции, т.е. потомок может унаследовать любой ген, имеющийся в популяции в заданной позиции хромосомы.

Скрещивание для представления с действительными значениями

Ген потомка получается как среднее арифметическое из генов родителей следующим образом (рис.15):

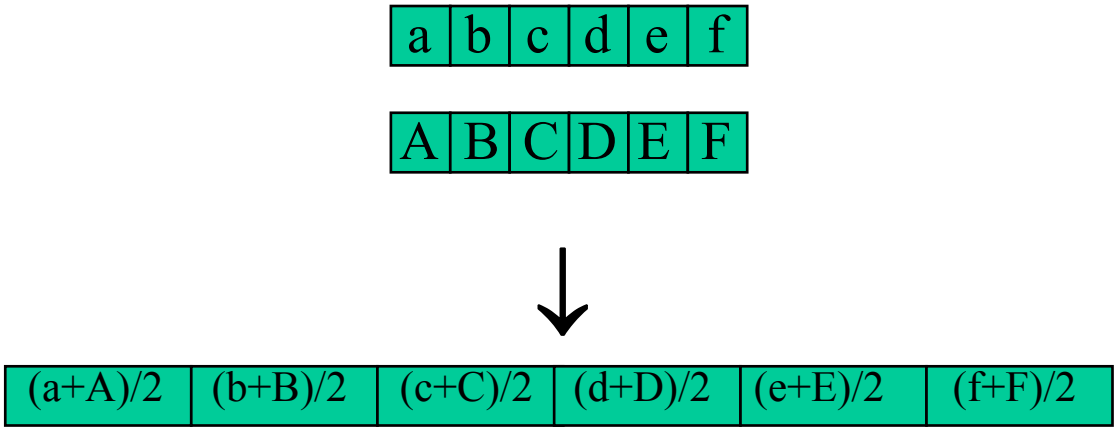


Рис. 15. Пример скрещивания для вещественного представления

Скрещивание для представления, основанного на порядке

Скрещивание происходит следующим образом (пример на рис. 16):

- Выбрать произвольную часть первого родителя и скопировать ее первому потомку;
- Скопировать оставшиеся гены, которые не перешли первому потомку;
- Начиная справа от точки разрыва скопированной части, использовать порядок генов от второго родителя;
- Если достигнут конец хромосомы, перейти к началу;
- Повторить этот процесс, поменяв родителей местами.

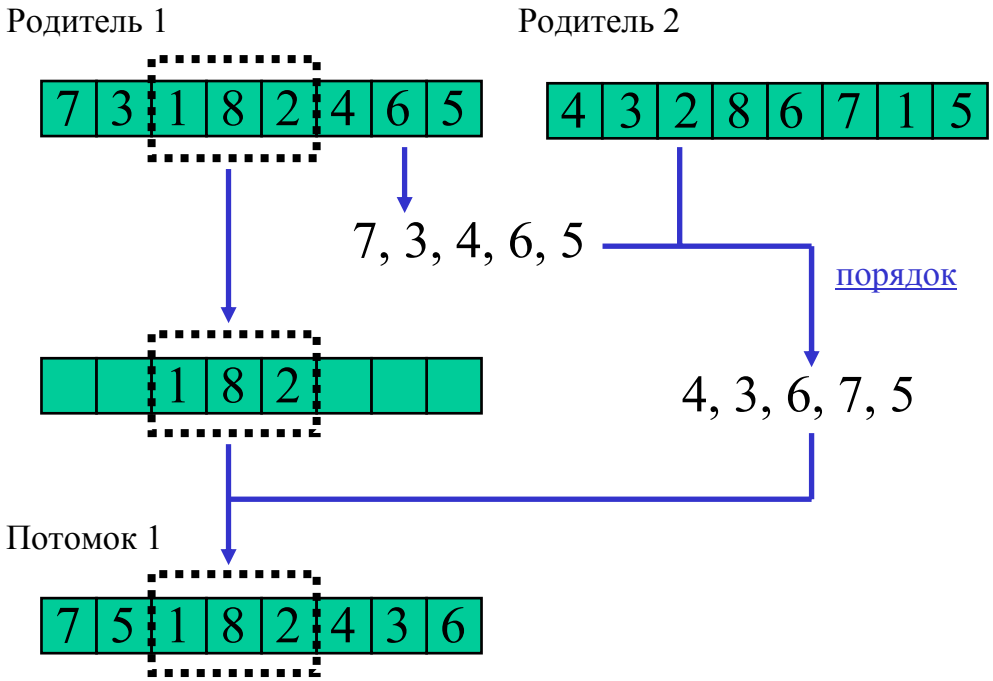


Рис.16. Пример скрещивания для представления, основанного на порядке

Скрещивание для представления, основанного на деревьях

Скрещивание осуществляется следующим образом. Выбираются родительская пара. У каждого из родителей выбирается точка скрещивания (дуга в графе). Родители обмениваются генами (поддеревьями), находящимися ниже точки скрещивания. Полученная пара является потомками. Пример скрещивания (рис. 17):

Оценка пригодности индивида

Пригодность индивида – это некоторая количественная оценка качества решения поставленной задачи. Функция пригодности может быть спроектирована с учетом особенностей решаемой задачи (что делает ГА довольно гибким, универсальным). Обычно функция пригодности принимает положительные значения, и значение пригодности максимизируют. В общем случае при решении задач безусловной оптимизации в качестве значения функции пригодности может выступать значение оптимизируемого функционала $fitness(X^i) = \chi(X^i)$, где X^i - *i*-й индивид в популяции, $fitness(X^i)$ - пригодность индивида X^i, $\chi(X^i)$ - значение целевой функции в точке X^i.

Необходимо иметь ввиду:

- Это наиболее затратный шаг для реальных приложений;
- Никогда не следует заново оценивать неизмененных индивидов;
- Можно использовать приближенную пригодность (аппроксимацию функции пригодности), но не слишком долго;
- Возможен учет нарушений ограничений задачи (например, введение штрафов);

Возможен учет многокритериальности задачи.

Методы отбора индивидов (селекция)

Оператор селекции - оператор, посредством которого индивиды в соответствии с их пригодностью выбираются для порождения потомков или для перехода в следующее поколение. Наиболее приспособленные особи должны выбираться с большей вероятностью для сохранения своих генов в следующем поколении. Таким образом,, оператор селекции позволяет сконцентрировать поиск на наиболее многообещающих регионах пространства.

Предложено множество различных схем отбора в ГА. Конкретный тип оператора селекции (а также и др. генетических операторов) проектируется исходя из решаемой задачи (например, когда требуется контроль допустимости решений, учет ограничений, многокритериальности и т.д.), однако наиболее распространены следующие базовые типы селекции.

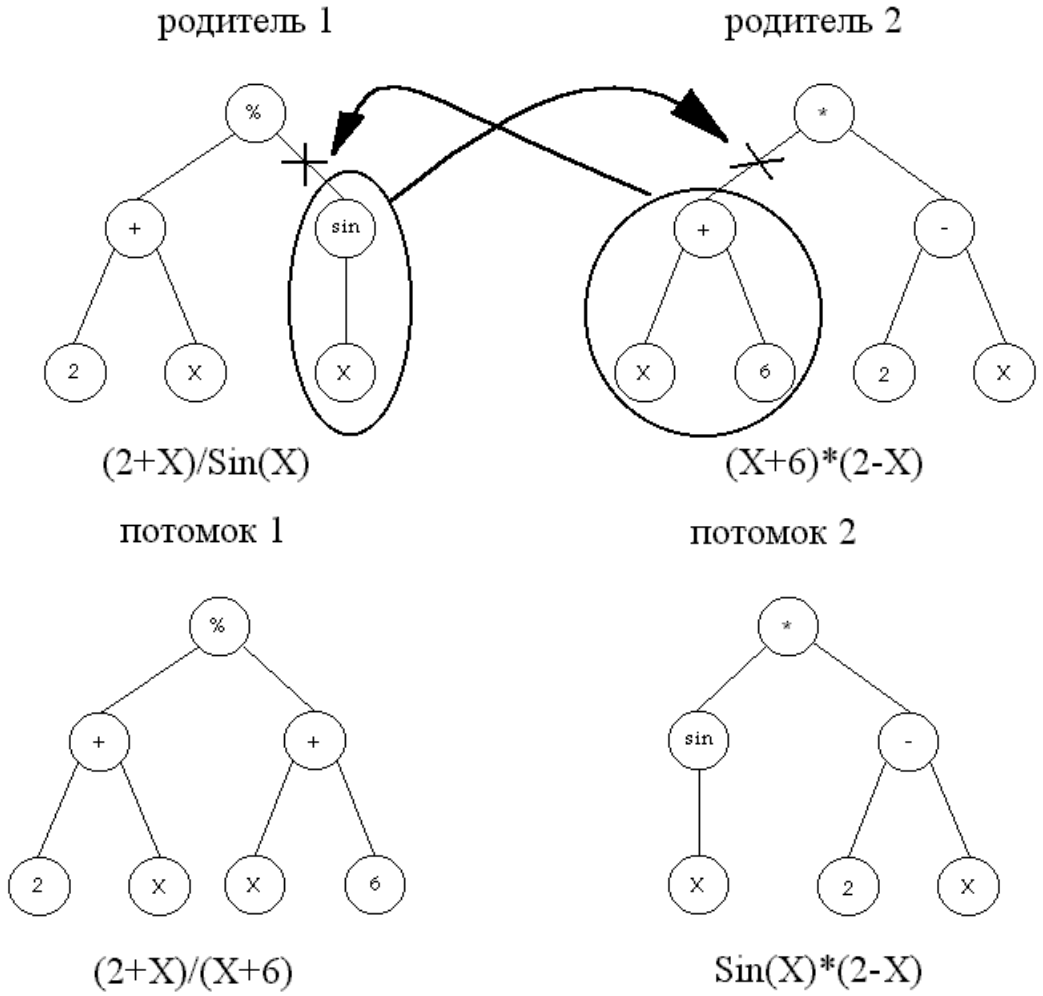


Рис. 17. Пример скрещивания для представления, основанного на деревьях

Пропорциональная селекция

В *пропорциональной селекции* вероятность индивида быть отобранным пропорциональна его пригодности. Вероятность вычисляется следующим образом (для задачи минимизации):

$$P(X^i) = \frac{-\mathit{fitness}(X^i) + C}{r * C - \sum_{j=1}^{r} \mathit{fitness}(X^j)},$$

где r - размер популяции, $\mathit{fitness}(X^i)$ - пригодность индивида, $C : P(X^i) \geq 0, \forall i$, $\sum_{j=1}^{r} P(X^j) = 1$.

Пропорциональная селекция обладает следующими недостатками: преждевременная сходимость и стагнация.

Стагнация возникает, когда на определенном этапе поиска все индивиды получают относительно высокую и примерно равную пригодность,

что приводит к очень низкому селективному давлению (наилучшее решение лишь немного предпочитается худшему).

Преждевременная сходимость (проблема супериндивида) возникает, когда на ранних этапах появляется индивид с пригодностью намного большей, чем у других индивидов в популяции, но очень плохой с точки зрения решаемой задачи. Вероятность супериндивида быть отобранным стремится к единице, в то время как вероятности других членов популяции – к нулю. В итоге он копирует себя в следующее поколение и вскоре «широкий» поиск прекращается.

Ранговая селекция

При применении *ранговой селекции* в задаче минимизации индивиды популяции ранжируются в соответствии с их пригодностью: $R_i < R_j$ если $f(X^i) \le f(X^j)$. Тогда

$$P(X^i) = \frac{R_i}{\sum_{k=1}^{r} R_k} = \frac{i}{\sum_{k=1}^{r} i} = \frac{2i}{r(r+1)}, \text{ где } \sum_{j=1}^{r} P(X^j) = 1.$$

Ранговая селекция устраняет недостатки пропорциональной: нет стагнации, т.к. даже к концу работы алгоритма $P(X^1) \neq P(X^2) \neq \ldots$, нет преждевременной сходимости, т.к. нет индивидов с вероятностью отбора близкой к единице.

Турнирная селекция

В *турнирной селекции* для отбора индивида создается группа из t ($t \ge 2$) индивидов, выбранных случайным образом. Индивид с наибольшей пригодностью в группе отбирается, остальные – отбрасываются. Параметр t называется размером турнира. Наиболее популярным является бинарный турнир. Этот тип селекции не требует сортировки популяции и вычисления пригодности для всех индивидов. Недостатки: худший индивид никогда не выбирается.

Селекция с усечением

В процессе селекции с усечением с порогом τ, только доля τ из всех лучших индивидов может быть отобрана, причем в этой доле каждый имеет одинаковую вероятность отбора.

Элитарная селекция

Как минимум одна копия лучшего индивида всегда переходит в следующее поколение. Преимущества: гарантия сходимости. Недостатки: большой риск захвата локальным оптимумом.

Стратегия замещения

На данном этапе необходимо определить, кто из текущих индивидов погибнет (будет удален из популяции), а кто выживет (перейдет в следующее поколение). Способ замещения индивидов при формировании новой популяции также влияет на качество работы алгоритма.

Можно использовать стохастическую стратегию замещения или же некоторый детерминированный метод.

В большинстве известных генетических алгоритмов для формирования новой популяции используется оператор селекции. В таком случае, выбор группы индивидов к которым применяется селекция и определяет стратегию замещения. Например:

- селекция применяется только к потомкам;
- селекция применяется к потомкам и родителям одновременно;
- определенная часть новой популяции может быть заполнена родителями, оставшаяся потомками;
- и т.д.

Элитизм

Должна - ли пригодность постоянно возрастать?

- Если должна, можно использовать *элитарную стратегию* (элитизм) - введение в популяцию лучшего индивида (группы лучших индивидов) предыдущей итерации.
- Если пригодность не должна постоянно возрастать, то элитарная стратегия не используется. Тем не менее, необходимо сохранять наилучшего индивида в отдельном месте (*консервация*). При отсутствии элитизма в генетическом алгоритме, в процессе своей работы алгоритм будет бесконечное число раз находить и забывать оптимальное решение. Поэтому в ГА консервация обязательна.

Рекомбинация или Мутация?

В начале развития теории эволюционных алгоритмов эти операторы использовались следующим образом:

- скрещивание – основной оператор в генетических алгоритмах (ГА);

- мутация - основной оператор в эволюционных стратегиях (ЭС) и эволюционном программировании (ЭП).

В последствии в алгоритмы были включены оба этих оператора.

Какой же из операторов предпочтительнее? Считается, что ЭА в процессе работы используют две стратегии: исследование и использование (exploration ‘n’ exploitation):

- Исследование – поиск и изучение неизвестных регионов пространств поиска. Недостаток: интенсивные исследования аналогичны случайному поиску, т.е. не обеспечивают сходимость.
- Использование – это попытка развития (улучшения) существующих решений. Недостаток: слишком интенсивное использование приводит к локальному поиску, следовательно имеем быструю сходимость к ближайшему локальному экстремуму.

В ГА оператор исследования – мутация, использования – скрещивание. Мутация – оператор широкого поиска, позволяет избегать захвата локальными экстремумами. Скрещивание – оператор, использующий свойства хороших решений, обладает нарастающим эффектом по мере сходимости алгоритма.

Очевидно, что разработчик должен обеспечить разумную комбинацию использования этих операторов. Тем не менее, существуют ситуации, когда использование конкретного оператора предпочтительнее.

Рекомбинация предпочтительнее:

- На заключительных этапах поиска;
- Область притяжения глобального оптимума велика или задача одноэкстремальная;
- Пригодность можно разделить, разделив решение на составные части (идея о строительных блоках);
- Пригодность сепарабельна, т.е. $F(x, y) = f_1(x) + f_2(y)$;
- Скрещивание семантически понятое и обоснованное, т.е. использует известные свойства задачи.

Мутация предпочтительнее:

- На ранних этапах поиска;
- Задача многоэкстремальная, пространство поиска велико;
- Мутация использует известные свойства задачи.

Критерии остановки алгоритма

Если значение в точке оптимума известно и ограничений на количество вычислений нет, то алгоритм следует останавливать при достижении оптимума.

Если значение в точке оптимума неизвестно, то для остановки алгоритма можно использовать следующие критерии:

- Ограничить вычислительные ресурсы (задать максимальное число вычислений функции пригодности);
- Определить максимальное число поколений без улучшения решения;
- Определить максимальное число поколений без улучшения средней пригодности по популяции.

Общие рекомендации

- Необходимо обеспечить генетическое разнообразие популяции. В некорректно спроектированном алгоритме потеря генетического разнообразия подобна «снежному кому» - информация теряется очень быстро, все решения становятся похожими друг на друга.
- Необходимо обеспечить разумный баланс между исследованием и использованием.
- Необходимо помнить принцип WYTIWYG: «What you test is what you get» - результат зависит от данных. Не следует использовать алгоритм, настроенный на простых тестовых (игрушечных) данных, для обработки реальных данных.
- Необходимо помнить принцип «Avoid user's frustration» - избегайте собственных предположений. Использование детерминированных механизмов может необратимо смещать поиск в неверном направлении.
- Никогда не следует делать никаких заключений из отдельного запуска алгоритма. Необходимо использовать статистические методы (математическое ожидание, медиана) для достаточного количества независимых запусков. Пользователь должен наблюдать статистическую устойчивость получаемых результатов. Например, если $\overline{X}^k$ - мат. ожидание полученного решения по k независимым запускам алгоритма, то

распределение значений мат. ожидания по различному числу запусков должно иметь вид (рис. 18):

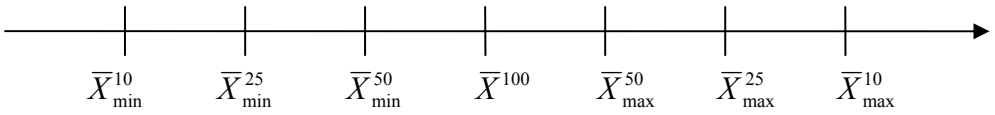


Рис. 1.8. Пример статистической устойчивости результатов

Как оценивать эффективность генетических алгоритмов? С точки зрения приложений:

- перспективы проекта – эффективным признается алгоритм, который находит достаточно хорошее решение хотя бы один раз;
- перспективы продукта - эффективным признается алгоритм, который находит достаточно хорошее решение почти при любом запуске.

Обычно для оценки эффективности стохастических поисковых алгоритмов используются следующие оценки: надежность и трудоемкость. *Надежность* – процент успешных запусков (решение найдено) алгоритма от общего числа запусков. *Трудоемкость* - среднее число вычислений оптимизируемого функционала, требуемое для нахождения решения задачи. Алгоритм с наибольшей надежностью считается лучшим. Среди алгоритмов с одинаковой надежностью – лучший тот, у которого наименьшая трудоемкость.

Заключение

Генетический алгоритм определяется следующими параметрами:

$GA(\Sigma, G, \rho, f, \mu, I, S, Gap, \Omega, R, \tau)$,

Σ - пространство поиска (фенотип),

G - пространство представления (генотип);

ρ - функция представления: $\rho : \Sigma \to G$,

f - функция пригодности: $f : \Sigma \to R^+$,

μ -размер популяции,

I - функция инициализации: $I : \mu \to P(G)$,

S - механизм селекции,

Gap - процент популяции, обрабатываемый генетическими операторами и возвращаемый назад,

Ω - множество пар: генетических операторов и их параметров, $\Omega(\omega, k_\omega)$,

R - политика замещения,

τ - критерий останова.

Проверка эффективности работы генетических алгоритмов на тестовых функциях

Вопрос об определении адекватного множества тестовых задач для конкретного типа алгоритмов совсем не прост, хотя некоторые необходимые принципы кажутся очевидными.

– Задачи в тестовом наборе должны быть непохожи друг на друга, чтобы у создателя алгоритма было меньше возможностей настроить его на специальных примерах.

– В тестовых задачах должны содержаться такие ситуации, которые обычно вызывают затруднения; процесс проверки алгоритма должен показать, что эти трудности преодолеваются.

– Тестовые задачи должны вызывать затруднения у локальных методов.

– Тестовые задачи должны быть нелинейными и несепарабельными.

– Тестовые задачи должны быть масштабируемыми.

Наиболее часто для проверки эффективности адаптивных алгоритмов глобального поиска используются функции Розенброка, Шекеля, Растригина, Катникова, Гриванка, Де Йонга. Предлагается тестовый набор, в который включены следующие функции:

– Ф1, Ф2 – многоэкстремальные функции одной переменной. Глобальные экстремумы расположены около границы допустимой области. Значения функций в точках глобального оптимума слабо отличаются от значений в точках локальных оптимумов. Характер изменения Ф1 симметричен относительно начала координат (рис. 1). Ф2 сильно осциллирует на всей области определения (рис. 2).

– Ф5, Ф9, Ф10 – классические тестовые задачи. Ф5 – функция Розенброка, имеет протяженный овраг в области оптимума («банан» Розенброка) (рис. 5). Ф9 и Ф10 – функции Катковника, многоэкстремальные, имеют сложную систему локальных оптимумов, локальные оптимумы очень «глубокие» (рис. 9, 10).

– Ф3, Ф4 – функции Растригина. Ф3 – классический вариант, численные коэффициенты подобранны таким образом, что значение функций в точке глобального оптимума слабо отличается от значений в точках соседних локальных оптимумов (рис. 3). Ф4 модифицированная Ф3. Нарушена регулярность расположения локальных оптимумов – функция развернута относительно оси *Z*. Функция имеет растяжение по переменной *x*

и сжатие по y – области притяжения локальных оптимумов имеют овражный вид (рис. 4).

– Ф6, Ф12, Ф13, Ф14 – многоэкстремальные функции с очень большим количеством локальных оптимумов (рис. 6, 12, 13, 14).

– Ф7, Ф8, Ф11, Ф15 – многоэкстремальные функции, имеющие в области определения множества постоянства. Ф7 – функция Де Йонга 2, специально разработана для тестирования эволюционных алгоритмов, почти на всей области определения имеет равное значение, области притяжения локальных и глобального оптимумов очень малы (рис. 7). Ф8 – функция «Сомбреро», имеет глобальный оптимум в начале координат, значения функции в точках равно удаленных от глобального оптимума равны (рис. 8). Ф15 – функция Шекеля «лисьи норы». Функция имеет в области определения 25 оптимумов с очень маленькими зонами притяжения, значения в локальных и глобальном оптимумах слабо различаются, в оставшейся части области определения функция имеет равное значение (рис. 15).

Как видно из описания, используемые тестовые функции реализуют многие свойства, затрудняющие оптимизацию: многоэкстремальность, несепарабельность, овражность, множества постоянства, значения функции в глобальном оптимуме слабо отличается от значений в локальных и др.

Набор тестовых задач

Функция 1.

$$I(x) = 0.05(x-1)^2 + (3 - 2.9 \cdot e^{-2.77257 \cdot x^2})(1 - \cos(x(4 - 50 \cdot e^{-2.77257 \cdot x^2})))$$

$$x \in [-1;1],\ \min = I(0.9544) = 0.00017$$

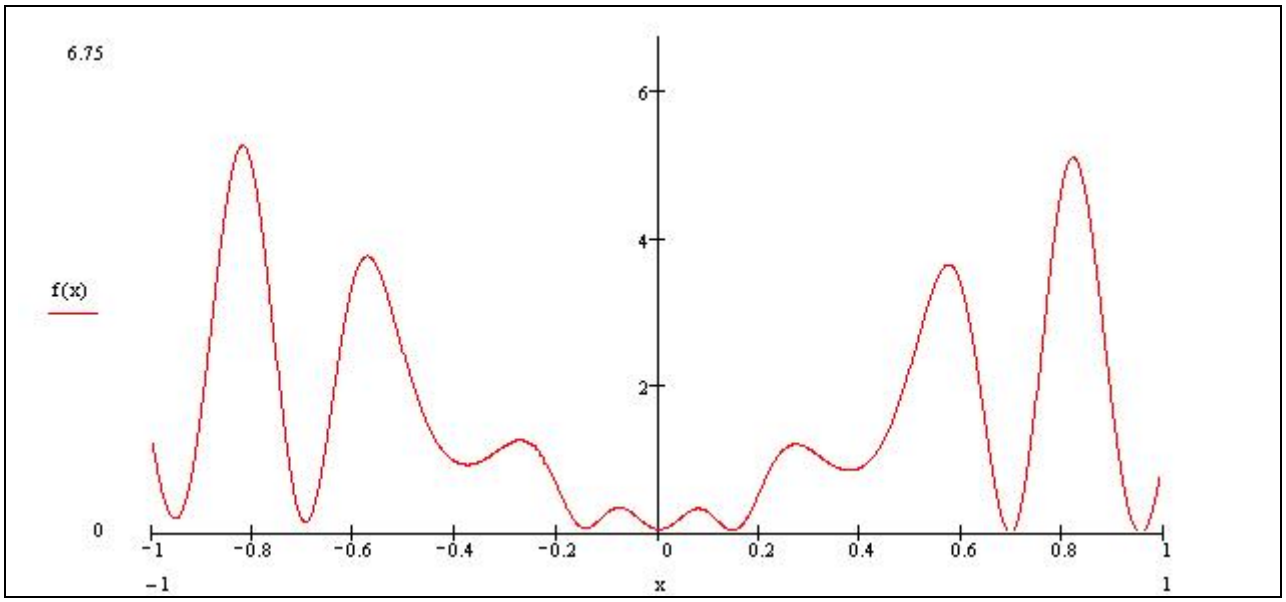


рис 1. График функции 1.

Функция 2.

$I(x) = 1 - 0.5\cos(1.5(10x - 0.3))\cos(31.4x) + 0.5\cos(\sqrt{5} \cdot 10x)\cos(35x)$

$x \in [-1;1],\ \min = I(-0.8113) = 0.1527$

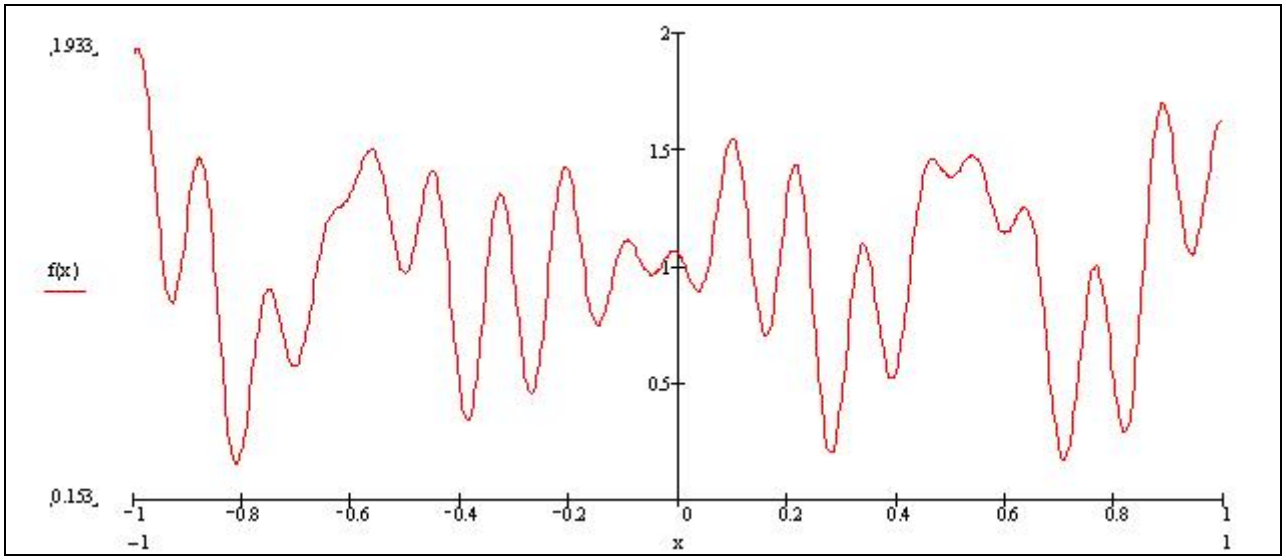


рис 2. График функции 2.

Функция 3. Функция Растригина.

$I(x, y) = 0.1x^2 + 0.1y^2 - 4 \cdot \cos(0.8 \cdot x) - 4 \cdot \cos(0.8 \cdot y) + 8$

$x, y \in [-16;16],\ \min = I(0,0) = 0$

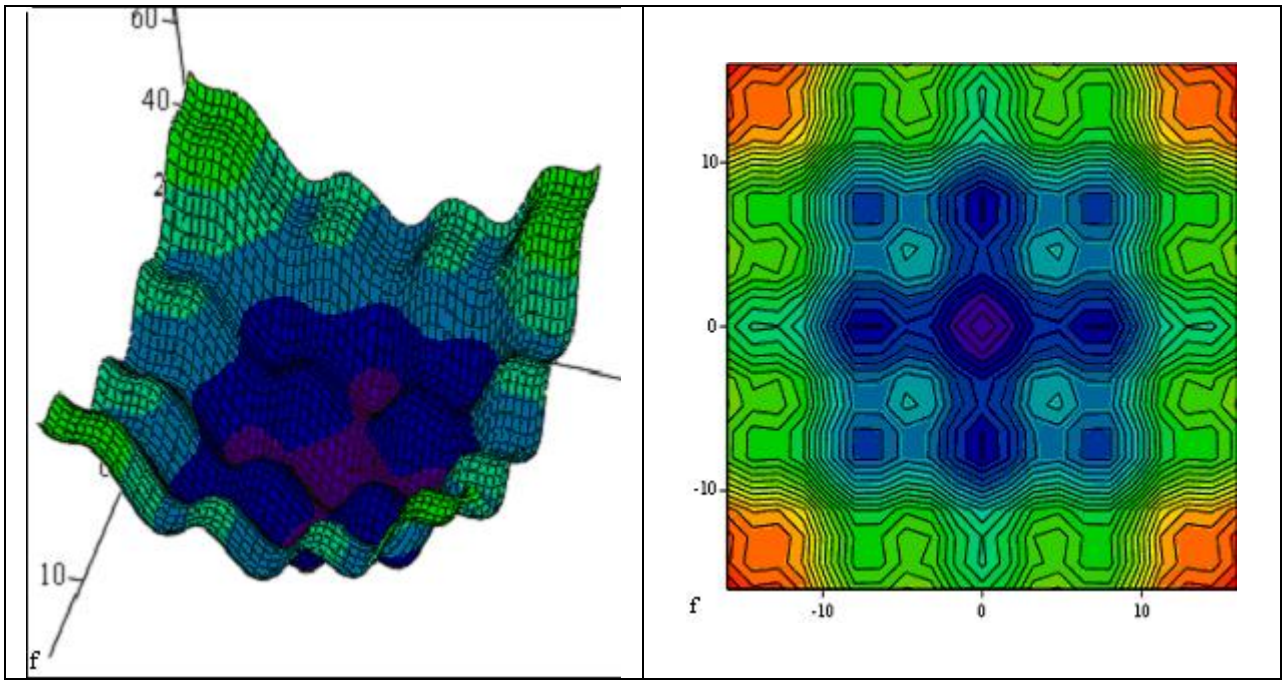


рис 3. График функции 3.

Функция 4. Функция Растригина овражная с поворотом осей.

$I(x,y) = (0.1 \cdot K_x \cdot A)^2 + (0.1 \cdot K_y \cdot B)^2 - 4 \cdot \cos(0.8 \cdot K_x \cdot A) - 4 \cdot \cos(0.8 \cdot K_y \cdot B) + 8$,

где $A = x \cdot \cos(\alpha) - y \cdot \sin(\alpha)$, $B = x \cdot \sin(\alpha) + y \cdot \cos(\alpha)$, K_x, K_y - растяжение/сжатие по x, y, α - угол поворота, $\alpha = \pi/2$, $kx = 1.5, ky = 0.8$, $x, y \in [-16;16]$, $\min = I(0,0) = 0$

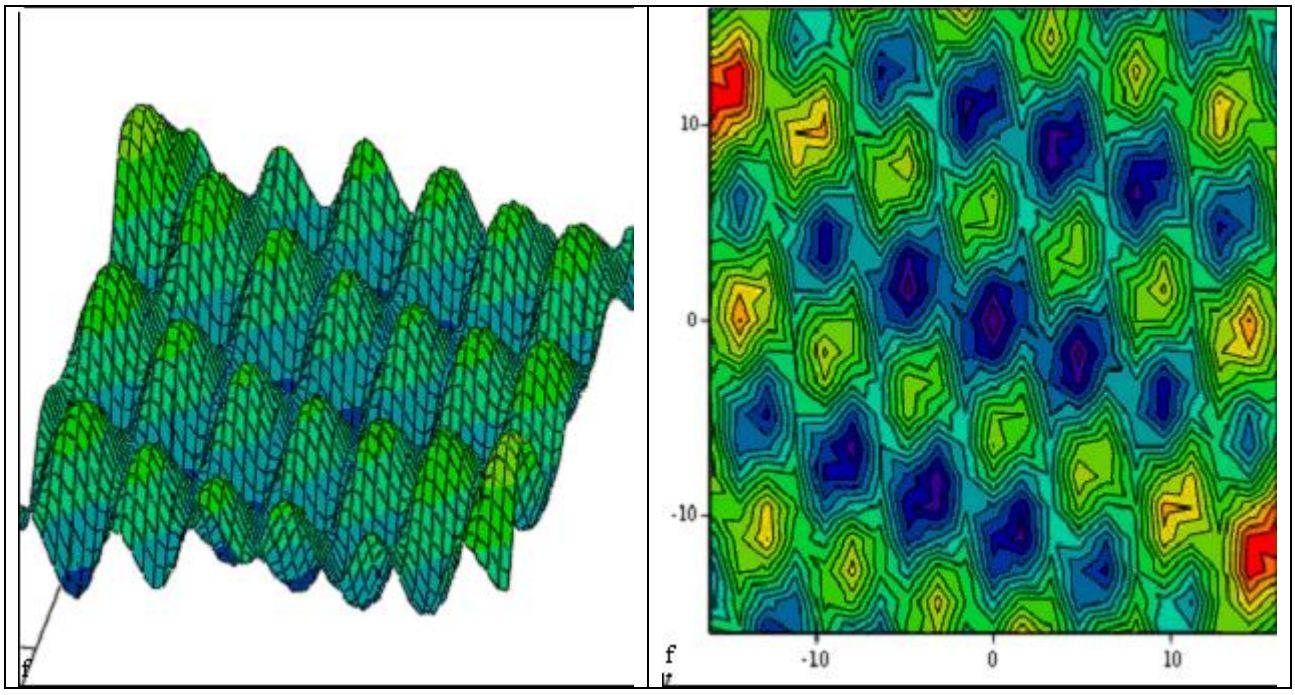


рис 4. График функции 4.

Функция 5. Функция Розенброка.

$$I(x,y) = 100 \cdot \left(y - x^2\right)^2 + (1-x)^2$$

$$x_1, x_2 \in [-2;2],\ \min = I(1,1) = 0$$

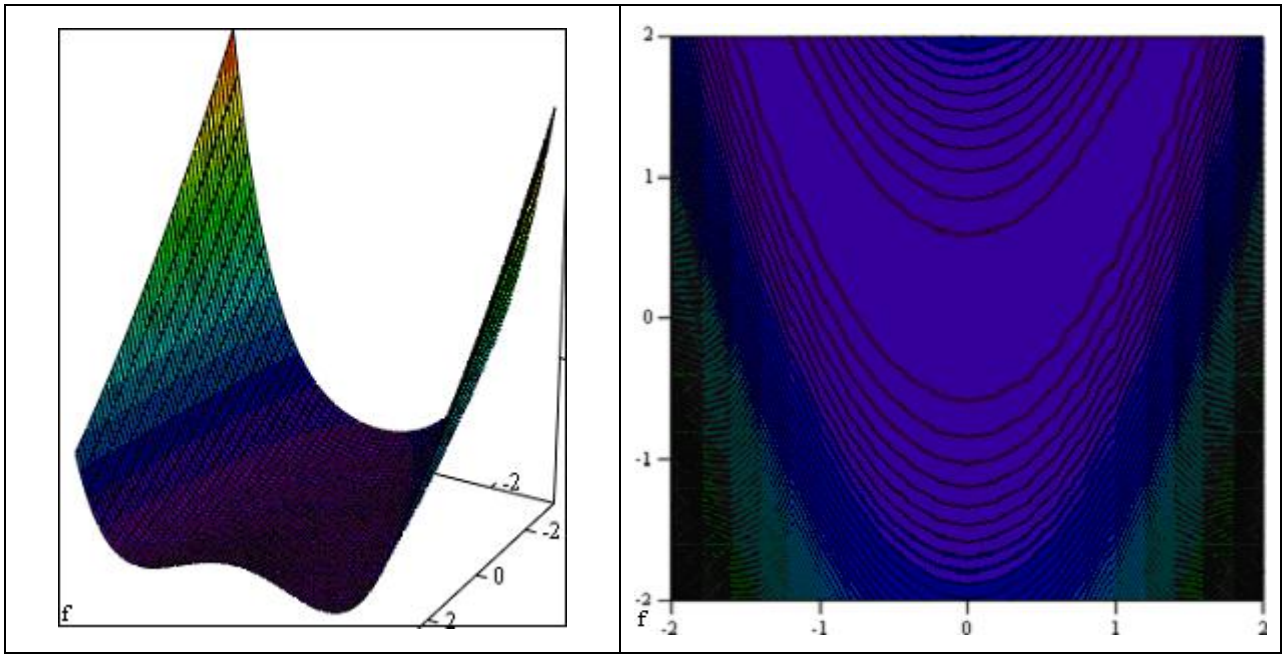


рис 5. График функции 5.

Функция 6. Функция Griewank.

$$I(x,y) = \frac{-10}{0.005 \cdot (x^2 + y^2) - \cos(x) \cdot \cos(y/\sqrt{2}) + 2} + 10$$

$$x_1, x_2 \in [-16;16],\ \min = I(0,0) = 0$$

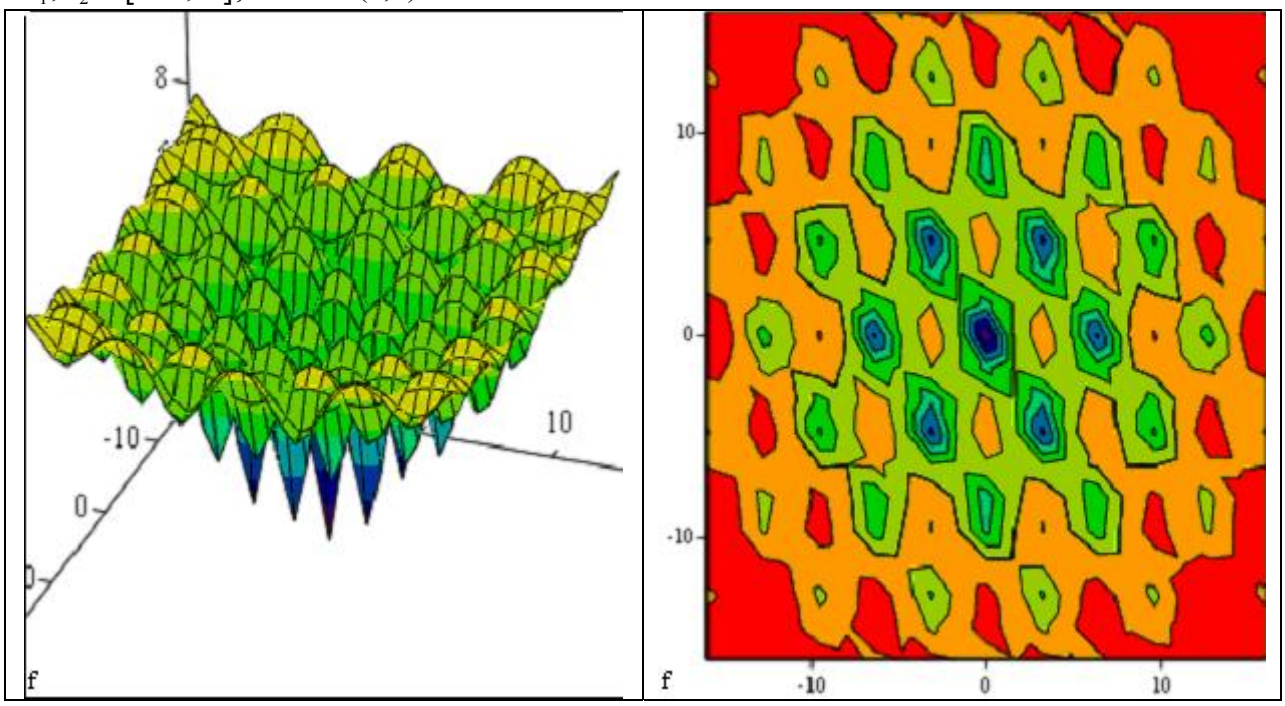


рис 6. График функции 6.

Функция 7. Функция De Jong 2.

$$F(x,y)=\frac{-100}{100(x^2-y)+(1-x)^2+1}+100,\ x_1,x_2\in[-5;5],\ \min=I(1,1)=0$$

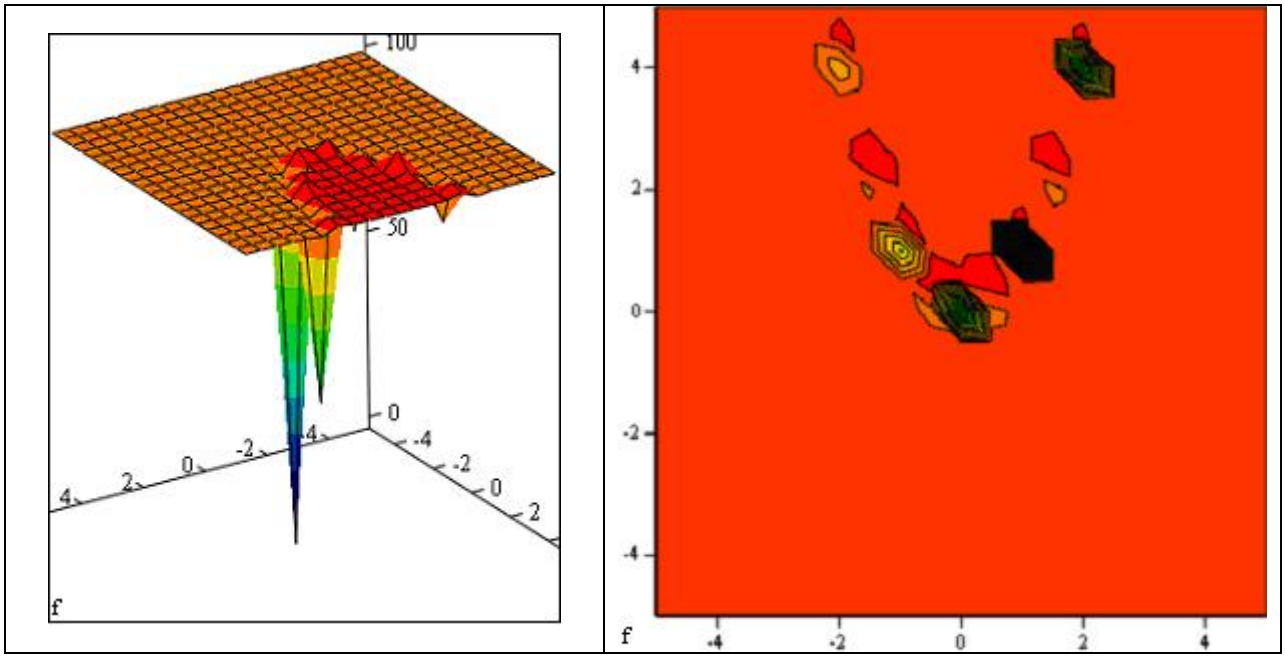


рис 7. График функции 7.

Функция 8. Функция «Сомбреро».

$$F(x,y)=\frac{1-\sin^2(\sqrt{x^2+y^2})}{1+0.001\cdot(x^2+y^2)}$$

$x_1,x_2\in[-10;10],\ \min=I(0,0)=0$

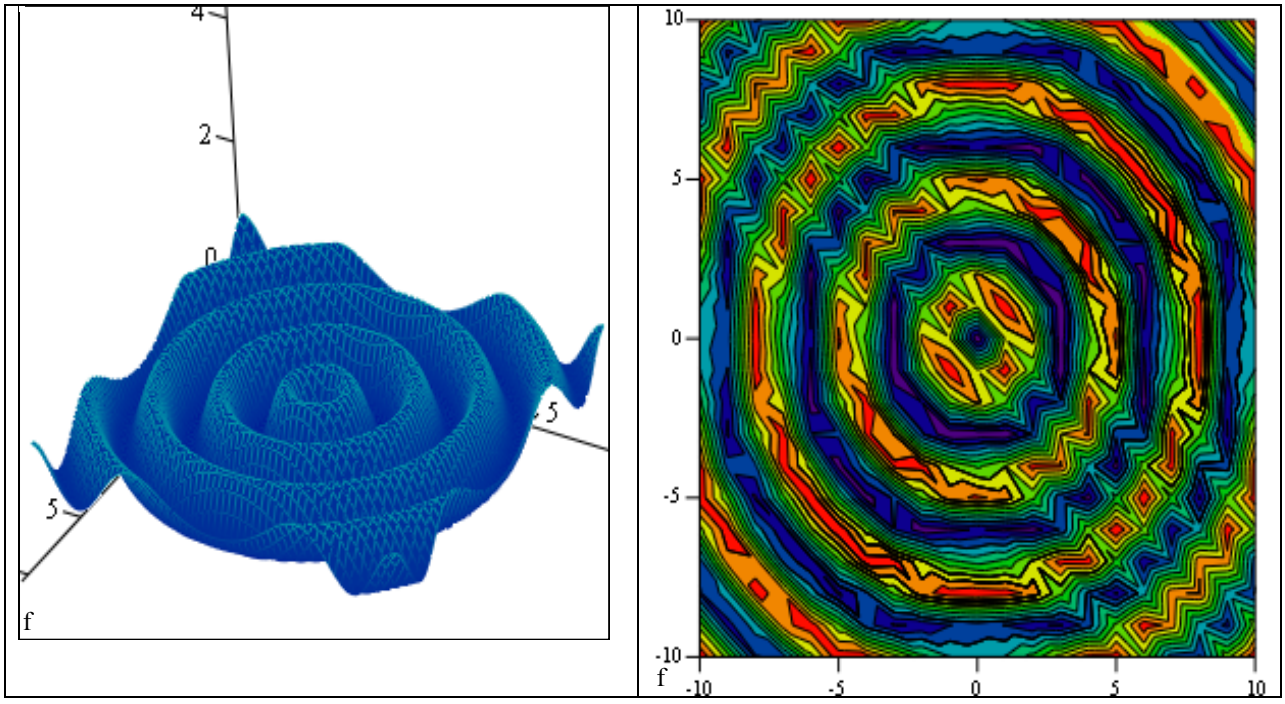


рис 8. График функции 8.

Функция 9. Функция Катковника.

$$I(x_1,x_2)=0.5(x_1^2+x_2^2)\left[2*A+A\cos(1.5x_1)\cos(3.14x_2)+A\cos(\sqrt{5}x_1)\cos(3.5x_2)\right]$$
$$A=0.8,\quad x_1,x_2\in[-2.5;2.5],\ \min=I(0,0)=0$$

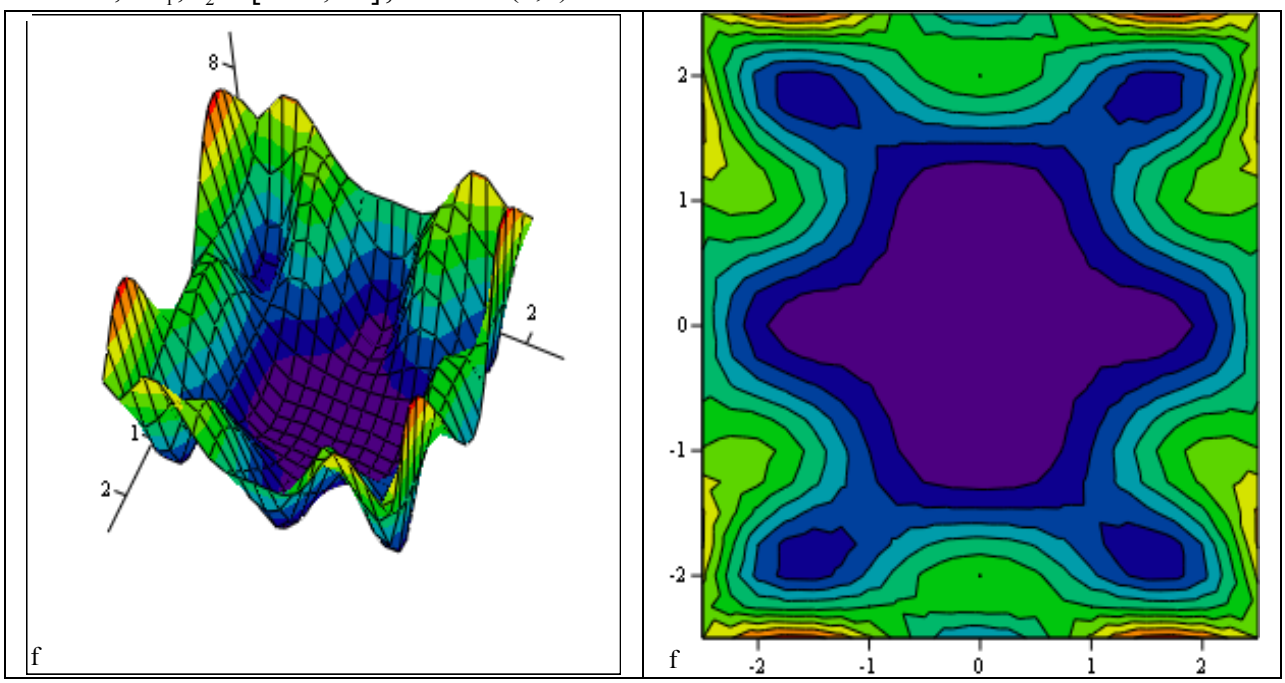


рис 9. График функции 9.

Функция 10. Функция Катникова

$$I(x_1,x_2)=0.5(x_1^2+x_2^2)\left[2*A+A\cos(1.5x_1)\cos(3.14x_2)+A\cos(\sqrt{5}x_1)\cos(3.5x_2)\right]$$
$$A=0.8,\quad x_1,x_2\in[-5;5],\ \min=I(0,0)=0$$

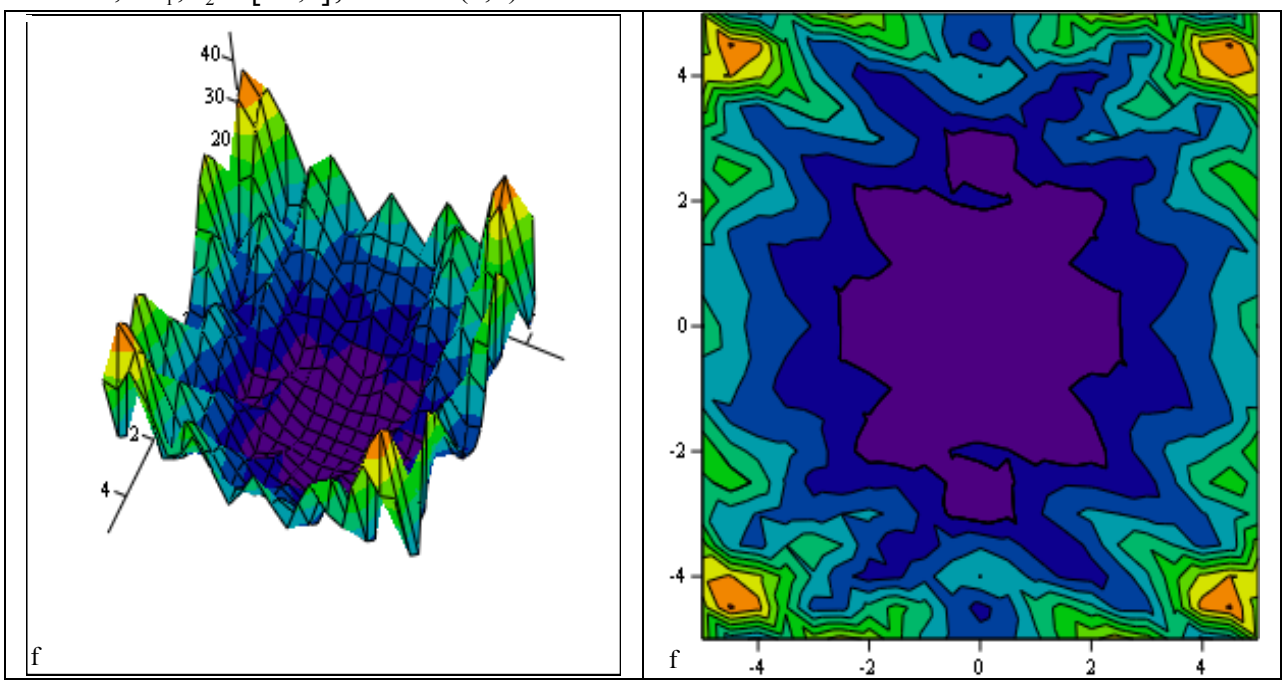


рис 10. График функции 10.

Функция 11.

$$I(x_1,x_2)=x_1^2\left|\sin 2x_1\right|+x_2^2\left|\sin 2x_2\right|-\frac{1}{(5x_1^2+5x_2^2+0.2)}+5$$

$x_1, x_2 \in [-4;4],\ \min = I(0,0) = 0$

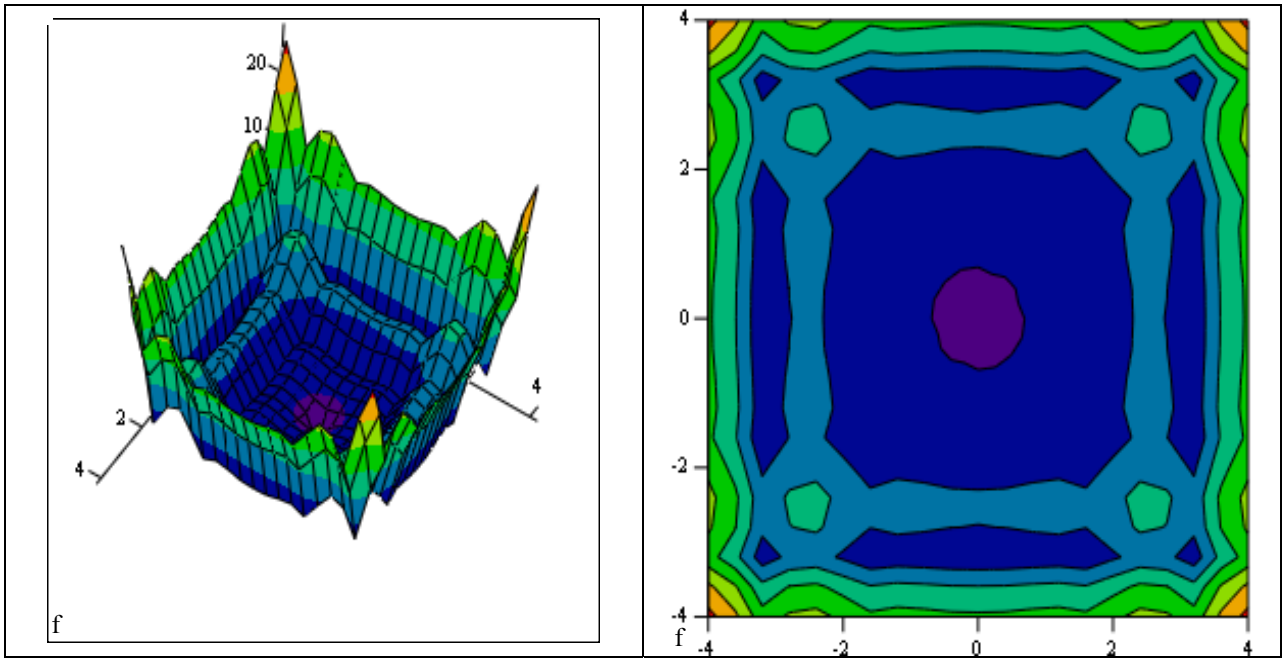


рис 11. График функции 11.

Функция 12.

$$I(x_1, x_2) = 0.5(x_1^2 + x_1 x_2 + x_2^2) \times$$
$$\times \left[1 + 0.5\cos(1.5x_1)\cos(3.2x_1 x_2)\cos(3.14x_2) + 0.5\cos(2.2x_1)\cos(4.8x_1 x_2)\cos(3.5x_2)\right]$$
$$x_1, x_2 \in [0;4],\ \min = I(0,0) = 0$$

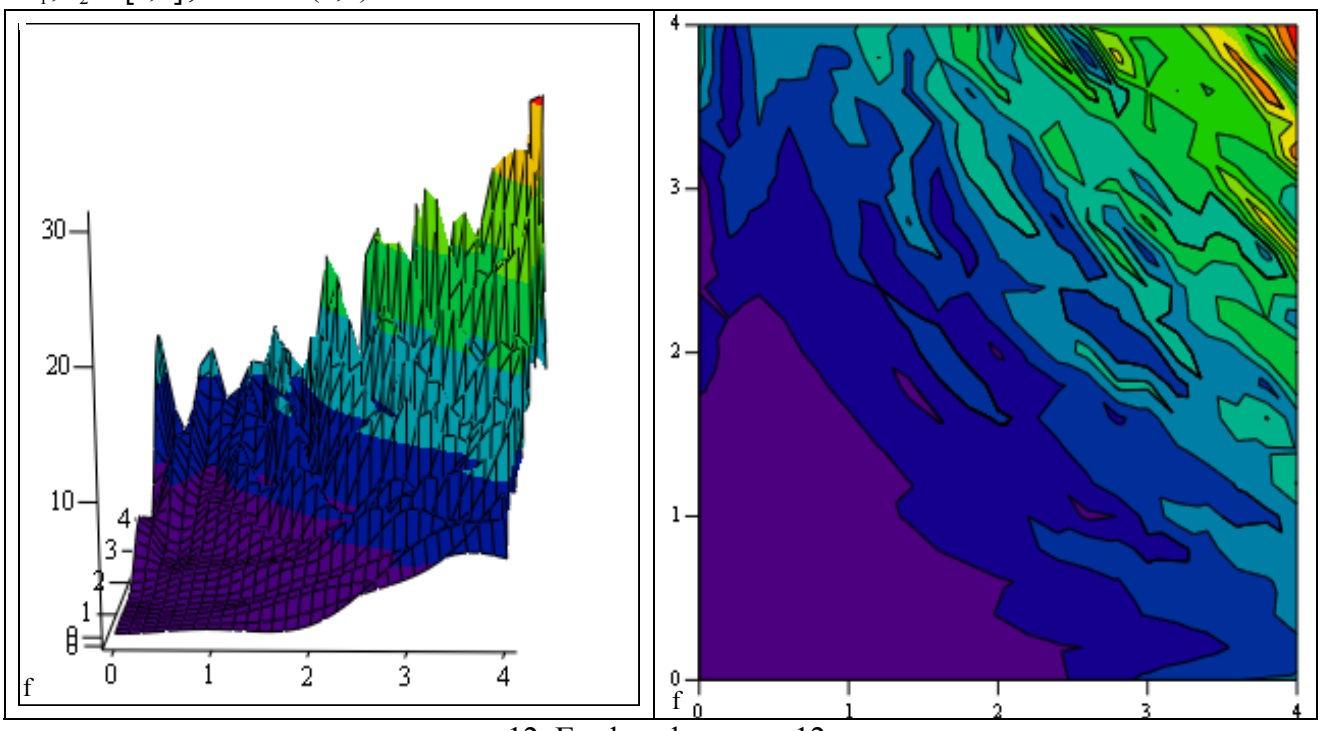


рис 12. График функции 12.

Функция 13. Мультипликативная потенциальная функция

$$I(x_1,x_2) = -z(x_1)z(x_2),\ z(x) = -\frac{1}{(x-1)^2+0.2} - \frac{1}{2(x-2)^2+0.15} - \frac{1}{3(x-3)^2+0.3}$$

$x_1, x_2 \in [0;4]$, $\min = I(2,2) = -60.8$

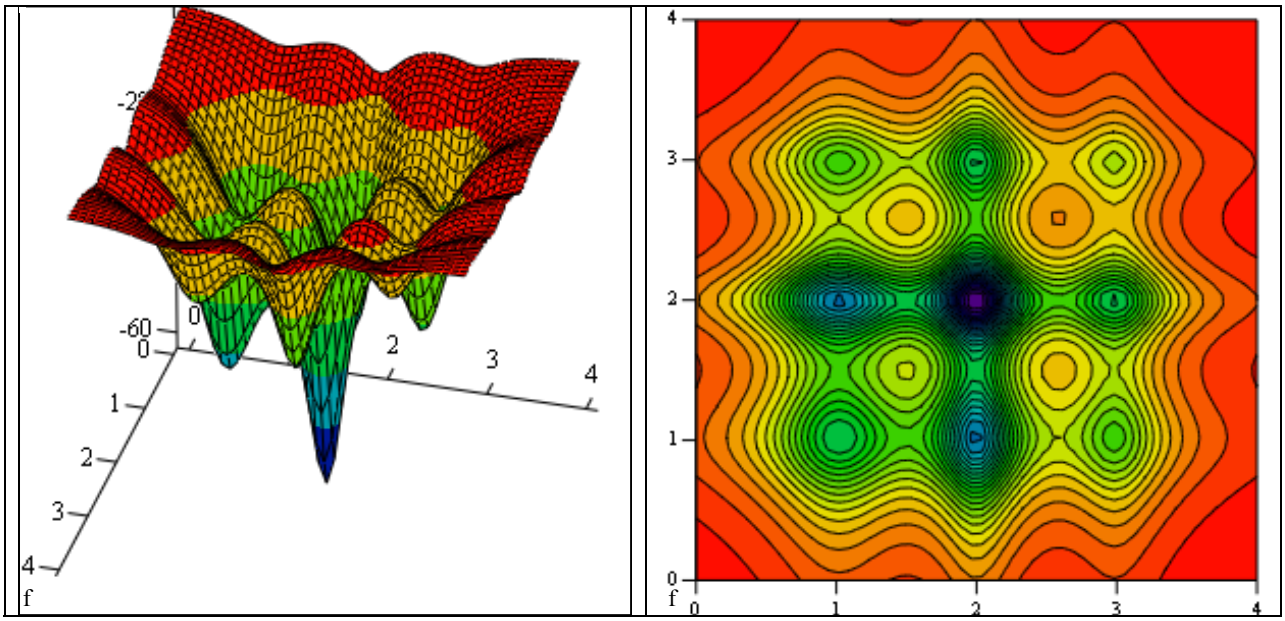


рис 13. График функции 13.

Функция 14. Аддитивная потенциальная функция

$$I(x_1,x_2) = z(x_1) + z(x_2),\ z(x) = -\frac{1}{(x-1)^2+0.2} - \frac{1}{2(x-2)^2+0.15} - \frac{1}{3(x-3)^2+0.3}$$

$x_1, x_2 \in [0;4]$, $\min = I(2,2) = -15.6$

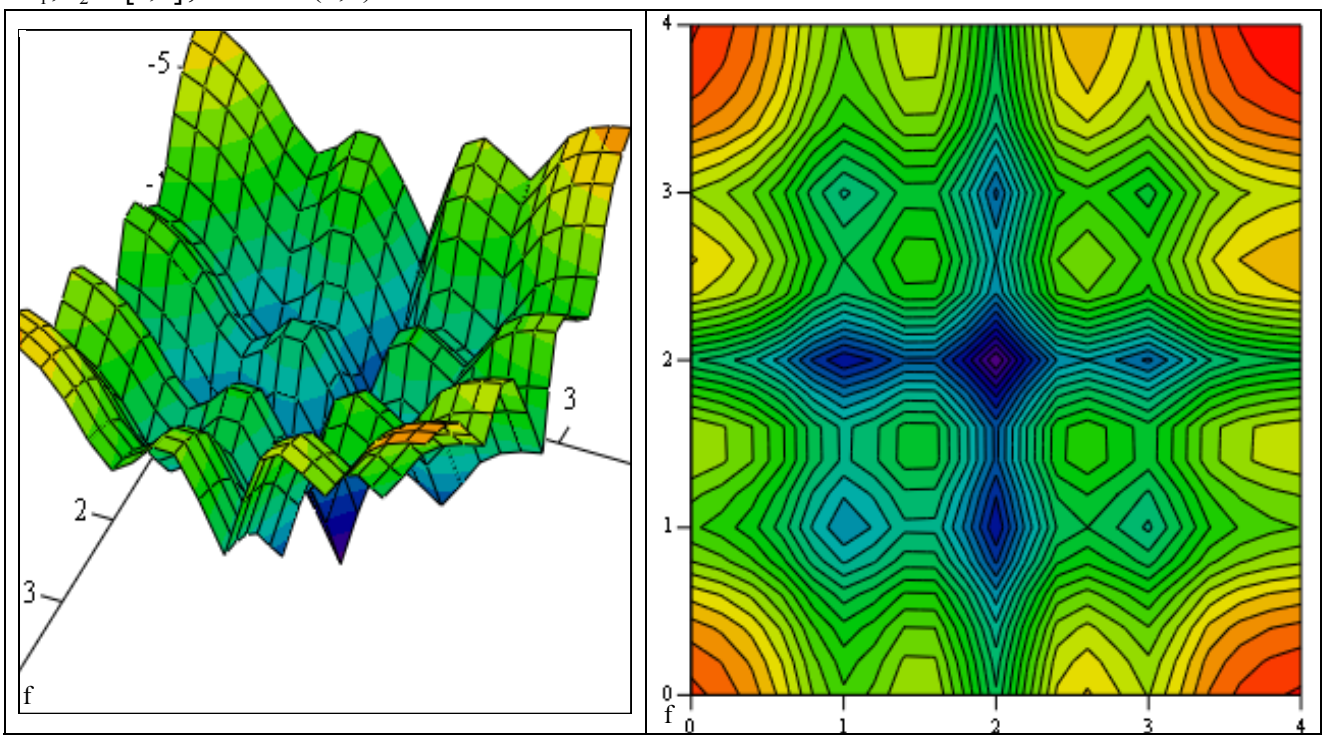


рис 14. График функции 14.

Функция 14. "Лисьи норы" Шекеля.

$$\frac{1}{I(\mathrm{x}_1,\mathrm{x}_2)}=\frac{1}{K}+\sum_{j=1}^{25}\frac{1}{c_j+\sum_{i=1}^{2}(x_i-a_{ij})^6},\quad \begin{cases} a_{1,(i-1)*5+1}=a_{2,i}=-32 \\ a_{1,(i-1)*5+2}=a_{2,i+5}=-16 \\ a_{1,(i-1)*5+3}=a_{2,i+10}=0 \\ a_{1,(i-1)*5+4}=a_{2,i+15}=16 \\ a_{1,(i-1)*5+5}=a_{2,i+20}=32 \end{cases}$$

K=500, $c_j=j,\ x_1,x_2\in[-65;65],\ \min=I(-32,32)\approx 1.$

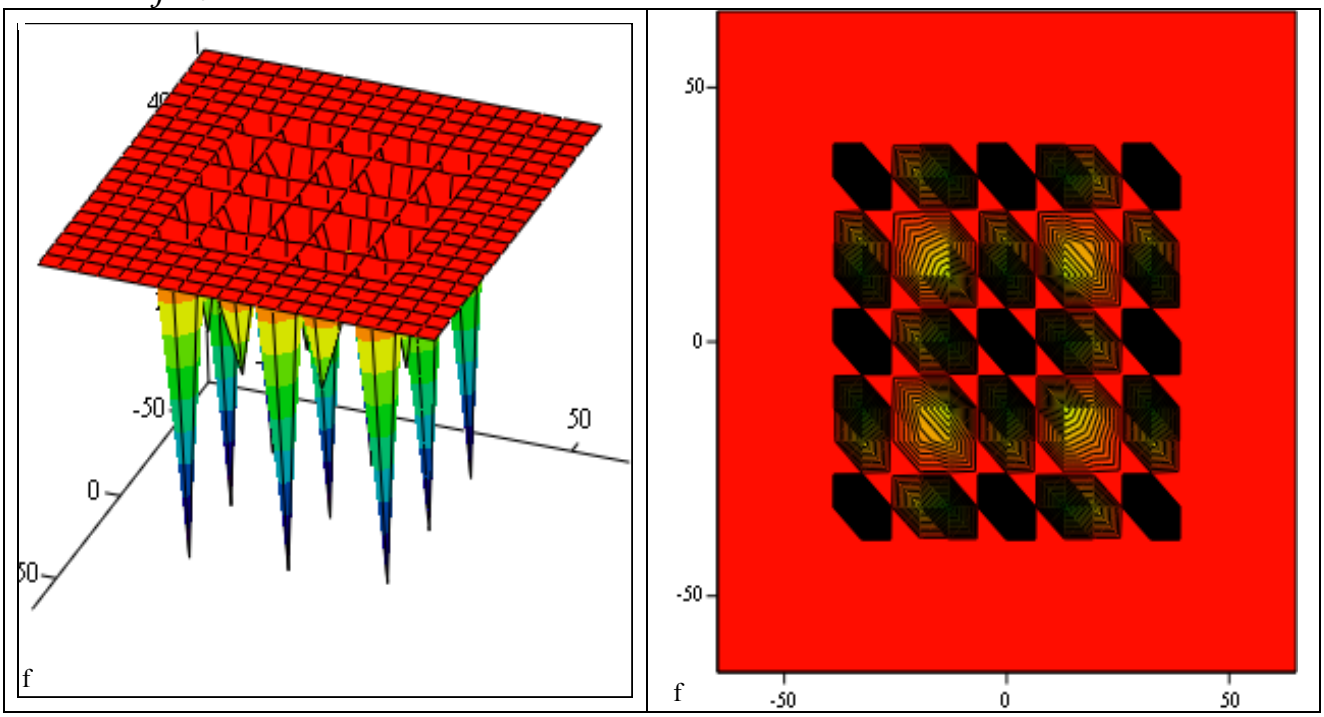


рис 15. График функции 15.

Пусть эффективность алгоритма проверяется на тестовых функциях Ф1-Ф15.Исследования проведем для ГА с одноточечным скрещиванием и ГА с равномерным скрещиванием по всей популяции. Реализованы пропорциональная, ранговая и турнирная (размер турнира – 2) селекции, низкая и высокая мутации. Использовалось стандартное бинарное и кодирование Грея. Размер популяции – 50. Количество прогонов алгоритма – 50.

Для оценки эффективности алгоритмов используем два показателя: надежность и среднее число итераций. *Надежность* – процент успешных запусков (решение найдено) к общему числу запусков алгоритма. *Среднее число итераций* – номер итерации, когда найдено решение, усредненный по успешным прогонам. Алгоритм с наибольшей надежностью всегда считается лучшим. Среди алгоритмов с одинаковой надежностью – лучший тот, у которого наименьшее среднее число итераций.

Результаты тестирования представлены в таблицах 1 и 2 (выделен алгоритм, «победивший» на данной тестовой задаче). Из таблицы видно, что для данного множества тестовых задач одноточечное и равномерное скрещивание в среднем дают похожие результаты. Наиболее эффективным оказывается использование высокой мутации, которая помогает избежать локальной сходимости даже при использовании пропорциональной селекции. Наилучший тип селекции – ранговая, практически на всех задачах эффективность алгоритма стабильно высокая. Эффективность кодирования Грея в сочетании с представленными алгоритмами сравнима с эффективностью стандартного бинарного кодирования.

Таблица 1

Тестовая задача	Тип селекции	ГА с одноточечным скрещиванием (низкая мутация)				ГА с одноточечным скрещиванием (высокая мутация)			
		Код Грея		Бинарный код		Код Грея		Бинарный код	
		Надеж-ность	Ср. число итер.	Надеж-ность	Ср. число итер.	Надеж-ность	Ср. число итер.	Надеж-ность	Ср. число итер.
Ф1	пропорциональная	24	19	54	15	82	31	76	18
	ранговая	26	32	56	20	84	32	100	18
	турнирная	16	15	46	9	34	15	46	12
Ф2	пропорциональная	68	21	50	20	84	19	94	13
	ранговая	64	30	76	14	100	30	100	19
	турнирная	60	10	70	10	72	14	68	12
Ф3	пропорциональная	74	25	20	9	82	31	26	9
	ранговая	34	29	28	12	83	28	85	28
	турнирная	24	15	10	10	44	17	24	10
Ф4	пропорциональная	2	48	6	13	44	30	8	15
	ранговая	4	25	10	10	0	0	24	20
	турнирная	6	14	4	14	2	6	8	12
Ф5	пропорциональная	2	10	18	15	38	12	35	18
	ранговая	8	16	20	16	48	28	44	16
	турнирная	4	10	12	26	8	10	8	10
Ф6	пропорциональная	4	14	18	9	28	30	20	19
	ранговая	0	0	12	9	2	36	14	18
	турнирная	2	10	10	7	0	0	4	12
Ф7	пропорциональная	0	0	0	0	0	0	7	17
	ранговая	0	0	0	0	0	0	6	30
	турнирная	0	0	2	8	0	0	0	0

Ф8	пропорциональная	10	28	20	5	48	26	58	14
	ранговая	10	18	30	7	14	28	18	30
	турнирная	16	16	14	6	10	20	22	7
Ф9	пропорциональная	12	11	50	10	36	14	60	8
	ранговая	8	25	40	9	24	19	76	12
	турнирная	0	0	38	10	8	12	44	11
Ф10	пропорциональная	40	12	46	7	70	20	40	10
	ранговая	4	28	36	10	24	33	44	16
	турнирная	6	18	22	8	18	15	34	11
Ф11	пропорциональная	16	16	48	8	50	33	32	8
	ранговая	2	13	38	8	6	28	34	9
	турнирная	2	10	14	9	6	12	12	6
Ф12	пропорциональная	96	14	86	7	100	12	98	7
	ранговая	100	8	100	8	100	8	100	8
	турнирная	38	7	86	7	60	10	82	7

Сравнительный анализ эффективности генетических алгоритмов

Продолжение таблицы 1

Ф13	пропорциональная	24	9	100	9	98	22	100	12
	ранговая	90	13	100	8	100	18	100	8
	турнирная	96	8	94	8	98	9	98	9
Ф14	пропорциональная	68	14	98	12	100	20	100	7
	ранговая	96	11	100	6	100	18	100	9
	турнирная	94	9	100	8	96	10	98	10
Ф15	пропорциональная	52	29	52	18	50	32	36	22
	ранговая	26	33	46	16	98	36	48	23
	турнирная	12	11	12	11	50	23	28	18

Таблица 2

Тестовая задача	Тип селекции	ГА с равномерным скрещиванием по всей популяции (низкая мутация)				ГА с равномерным скрещиванием по всей популяции (высокая мутация)			
		Код Грея		Бинарный код		Код Грея		Бинарный код	
		Надежность	Ср. число итер.	Надежность	Ср. число итер.	Надежность	Ср. число итер.	Надежность	Ср. число итер.
Ф1	пропорциональная	22	12	40	13	68	28	82	15
	ранговая	26	30	48	12	76	34	94	24
	турнирная	24	10	46	13	28	20	50	11
Ф2	пропорциональная	78	18	76	19	92	23	100	21
	ранговая	70	25	70	13	98	30	98	20
	турнирная	62	12	76	14	72	15	62	16
Ф3	пропорциональная	72	18	32	10	82	13	22	15
	ранговая	50	23	25	16	81	32	20	16
	турнирная	22	14	12	8	48	15	10	30
Ф4	пропорциональная	8	16	14	10	40	26	24	12
	ранговая	10	27	18	14	27	25	16	20
	турнирная	4	10	10	10	6	20	10	17
Ф5	пропорциональная	0	0	12	11	30	26	38	18
	ранговая	10	20	40	10	46	30	42	18
	турнирная	12	19	14	8	10	20	24	12
Ф6	пропорциональная	0	0	8	14	22	35	24	28
	ранговая	0	0	14	16	11	32	24	20
	турнирная	0	0	0	0	4	15	14	20
Ф7	пропорциональная	0	0	0	0	0	0	2	30
	ранговая	0	0	5	12	0	0	6	20
	турнирная	0	0	0	0	0	0	5	25
Ф8	пропорциональная	24	16	48	10	38	20	56	18
	ранговая	25	17	36	11	40	23	53	20
	турнирная	12	8	16	8	16	10	26	10
Ф9	пропорциональная	12	17	30	8	30	20	40	8
	ранговая	8	10	46	10	16	21	36	11
	турнирная	6	6	38	9	4	10	26	10
Ф10	пропорциональная	56	16	22	8	60	26	34	10
	ранговая	30	24	34	10	44	30	30	14
	турнирная	28	12	22	12	18	21	6	8

Продолжение таблицы 2

Ф11	пропорциональная	24	15	18	8	57	28	20	9
	ранговая	2	7	22	9	10	27	48	10
	турнирная	6	10	30	9	12	20	34	9
Ф12	пропорциональная	100	10	90	18	100	8	100	8
	ранговая	100	12	94	15	100	8	100	8
	турнирная	72	8	94	13	82	8	86	8
Ф13	пропорциональная	28	17	100	14	96	20	100	10
	ранговая	100	14	100	16	100	17	100	8
	турнирная	100	8	100	12	96	8	100	8
Ф14	пропорциональная	80	11	88	13	100	22	100	9
	ранговая	100	9	100	9	100	15	100	8
	турнирная	100	15	88	13	98	9	100	13
Ф15	пропорциональная	72	28	26	15	89	27	26	17
	ранговая	54	34	22	20	87	32	34	20
	турнирная	38	12	22	13	62	22	16	18

Подчеркнем, что выводы верны только в рамках данного тестирования, т.е. для использованного набора задач.

Эволюционные стратегии.

Рассмотрим эволюционные стратегии (ЭС). Так как для наших целей наиболее интересной представляется параметрическая оптимизация, то поисковые точки будут соответствовать n-мерным векторам вещественных чисел $X \in R^n$.

Далее $N(0,1)$ обозначает нормально распределенную одномерную величину с нулевым математическим ожиданием и стандартным отклонением. То же самое, но с отклонением ### обозначается как $\sigma \cdot N(0,1)$.

Впервые ссылки на ЭС появились в 1964 году в Техническом Университете Берлина. Применения были в основном экспериментальными и имели дело с гидродинамическими проблемами типа оптимизации формы сочленения трубы и горящего сопла. Различные версии ЭС были также апробированы на первом компьютере университета. И. Рехенберг развил теорию скорости сходимости для так называемого $(1+1)$-ЭС, где простой механизм селекции и мутации работал на одном индивидууме, который создавал одного потомка за поколение через мутацию гауссовского типа, и также предложил теоретически обоснованное правило изменения стандартного отклонения мутаций (1/5 - правило успеха). Он также предложил $(\mu+1)$-ЭС, где μ>1 индивидуумов рекомбинируют для формирования одного потомка, который после мутации вытесняет наихудшего родителя как и в симплексном методе. Эта стратегия, хотя и не получила широкого использования, но послужила отправной точкой для перехода к $(\mu+\lambda)$-ЭС и (μ,λ)-ЭС, введенным и исследованным Х.-П. Швефелем. Приведем описание этих стратегий в обобщенной форме.

Оценивание функции пригодности и представление поисковых точек

Поисковые точки в ЭС являются n-мерными векторами $\bar{x} \in R^n$, и значение полезности любого индивидуума идентично его значению целевой функции, т.е. $\Phi(\bar{a}) = f(X)$, где X есть компонент $\bar{a}$. Кроме того, каждый индивидуум может включать в себя до n различных вариаций $c_{ii} = \sigma_i^2 (i \in \{1,...,n\})$, а также до n(n-1)/2 ковариаций

$$c_{ij} (i \in \{1,...,n-1\}, j \in \{i+1,...,n\})$$

n-мерного нормального распределения с ожиданием $\bar{q}$, имеющим функцию распределения вероятностей

$$p(\mathrm{z}) = \sqrt{\frac{\det \mathbf{A}}{(2\pi)^n}} \cdot \exp(-\frac{1}{2}\mathrm{z}^{\mathrm{T}}\mathrm{Az}),$$

где $\mathbf{A}^{-1} = \|c_{ij}\|$ представляет собой ковариационную матрицу и z обозначает случайную переменную. В целом, до $\omega = n \cdot (n+1)/2$ стратегических параметров могут быть скомбинированы с объектными переменными для формирования индивидуума $\overline{a} \in \mathrm{I} = \mathrm{R}^{n+\omega}$. Часто, однако, принимаются в расчет только вариации, т.е. $\overline{a} \in \mathbf{I} = \mathbf{R}^{2n}$, и иногда даже полезно работать с одной общей для всех объектных переменных вариацией, т.е. $\overline{a} = \mathbf{I} = \mathbf{R}^{n+1}$.

Для работы с положительно определенной ковариационной матрицей $\mathbf{A}^{-1}$, алгоритм использует эквивалентные поворотные углы $\alpha_{ij}(\tan 2\alpha_{ij} = 2c_{ij}/(\sigma_i^2 - \sigma_j^2))$. Далее размеры шага мутации σ_i, т.е. стандартные отклонения, используются в реализации чаще чем вариации σ_i^2. Далее $\overline{\mathrm{N}}(\overline{q}, \overline{\sigma}, \overline{\alpha})$ обозначает случайный вектор, распределенный с ожиданием $\overline{q}$, стандартными отклонениями $\overline{\sigma}$, и поворотными углами $\overline{\alpha}$. $\overline{a} = (\mathrm{X}, \overline{\sigma}, \overline{\alpha}) \in \mathrm{R}^{n+\omega}$ используется для обозначения индивидуума в целом.

Оператор мутации

В общем виде (локальный) оператор мутации $m'_{\{\tau,\tau',\beta\}} : \mathrm{I} \to \mathrm{I}$ (где $\mathrm{I} = \mathrm{R}^{n+\omega}$) создает мутированного индивидуума

$$m'_{\{\tau,\tau',\beta\}}(\overline{a}) = (\mathrm{X}', \overline{\sigma}', \overline{\alpha}')$$

при первом мутировании стандартных отклонений и углов поворота, и затем, мутируя объектные переменные согласно модифицированной функции распределения вероятностей индивидуума $\overline{a}$, т.е.

$\forall i \in \{1,...,n\}$, $\forall \mathrm{j} \in \{1,...,\mathrm{n} \cdot (\mathrm{n}-1)/2\}$:

$$\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)),$$

$$\overline{\alpha}' = \alpha_j + \beta \cdot N_j(0,1),$$

$$\mathrm{X}' = \mathrm{X} + \overline{N}(\overline{q}, \overline{\alpha}, \overline{\sigma}).$$

Мутации объектных переменных теперь могут быть линейно коррелированы согласно значениям $\overline{\alpha}$ и $\overline{\sigma}$ обеспечивает масштабирование метрик. Глобальный фактор $\tau' \cdot N(0,1)$ позволяет всем индивидуумам менять мутированность, в то время как $\tau \cdot N_i(0,1)$ позволяет индивидууму менять размер шага σ_i. Факторы τ, τ' и β являются робастными параметрами,

которые Х.-П. Швефель предложил следующими: $\tau \propto (\sqrt{2\sqrt{\pi}})^{-1}$, $\tau' \propto (\sqrt{2\pi})^{-1}$, $\beta \approx 0.0873$.

Обычно, константы для τ и τ' равны 1, а значение, предложенное для β (в радианах) равно 5°.

Рекомбинация

В ЭС используются различные механизмы рекомбинации. В основном - это схема, в которой от двух случайно выбранных родителей воспроизводится один потомок, либо схема, в которой компоненты для одного индивидуума берутся, практически, ото всех индивидуумов родительской популяции (глобальная схема). Более того рекомбинация осуществляется и на стратегических параметрах так же как и на объектных переменных, и оператор рекомбинации может быть различен для объектных переменных, стандартных отклонений и поворотных углов. Правила рекомбинации для оператора $r' : \mathrm{I}^{\mu} \to \mathrm{I}$, создающие индивидуума $r'(P(t)) = \overline{\alpha}' = (\mathrm{X}', \overline{\sigma}', \overline{\alpha}') \in \mathrm{I}$ даны здесь только для объектных переменных $(\forall i \in \{1,...,n\})$:

$$
\mathrm{X_i'} = \begin{cases} \mathrm{X_{S,i}} & \text{без рекомбинации} \\ \mathrm{X_{S,i}} \text{ или } \mathrm{X_{T,i}} & \text{дискретная} \\ \mathrm{X_{S,i}} + \chi \cdot (\mathrm{X_{T,i}} - \mathrm{X_{S,i}}) & \text{промежуточная} \\ \mathrm{X}_{\mathrm{S_i},i} \text{ или } \mathrm{X}_{\mathrm{T_i},i} & \text{глобальная дискретная} \\ \mathrm{X}_{\mathrm{S_i},i} + \chi_i(\mathrm{X}_{\mathrm{S_i},i} - \mathrm{X}_{\mathrm{T_i},i}) & \text{глобальная промежуточная} \end{cases}
$$

Индексы S и T обозначают двух родителей, случайно выбранных из $P(t)$, и $\chi \in [0,1]$ есть равномерная случайная величина. Для глобальных вариантов, для каждого компонента Х родители S_i, T_i так же как и Xi определяются по-новому.

Селекция

Селекция в ЭС полностью детерминированная, выбирающая μ лучших $(1 \le \mu < \lambda)$ индивидуумов из множества λ потомков ((μ, λ)- селекция), либо из объединения родителей и потомков ($(\mu + \lambda)$- селекция). Хотя $(\mu + \lambda)$-селекция является элитной и, следовательно, гарантирует монотонное улучшение, эта схема неспособна работать с меняющейся средой и осуществлять самоадаптацию стратегических параметров (внутренняя модель), особенно внутри маленьких популяций. Следовательно, (μ, λ)- селекция рекомендуется

на сегодняшний день, причем наиболее оптимальным является отношение μ/λ=1/7. Комбинируя предшествующие разделы можно описать концептуальный алгоритм для $(\mu+\lambda)$ - ЭС и (μ,λ) - ЭС следующим образом:

Алгоритм эволюционных стратегий

$t := 0$; *initialize* $P(0) := \{\bar{a}_1(0),...,\bar{a}_\mu(0)\} \in I^\mu$, где $I = R^{n+\omega}$ и $\bar{a}_k(X_i, s_i, a_j, \forall i \in \{1,...,n\}, \forall j \in \{1,...,n\cdot(n-1)/2\})$;

evaluate $P(0) : \{\Phi(\bar{a}_1(0)),...,\Phi(\bar{a}_\mu(0))\}$

where $\Phi(\bar{a}_k(0)) = f(X_k(0))$;

while $(l(P(t)) \neq true)$ do

$recombine : \bar{a}'_k(t) := r'(P(t)), \forall k \in \{1,...,\lambda\}$

$mutate : \bar{a}''_k(t) := m'_{\{\tau,\tau',\beta\}}(\bar{a}'_k(t)), \forall k \in \{1,...,\lambda\}$

$evoluate : P''(t) := \{\bar{a}''_1(t),...,\bar{a}''_\lambda(t)\}$; $\{\Phi(\bar{a}''_1(t)),...,\Phi(\bar{a}''_\lambda(t))\}$

$select : P(t+1) :$ $if\,(\mu,\lambda) - selection$

then $s_{(\mu,\lambda)}(P''(t))$;

else $s_{(\mu+\lambda)}(P(t) \bigcup P''(t))$;

$t := t+1$;

od.

Так как сходство ГА и ЭС очевидно, а различия не выглядят существенными, то необходимо обсудить какие именно свойства этих алгоритмов должны учитываться при выборе процедуры для оптимизации. Экспериментальные исследования позволяют сделать вывод о том, поведение алгоритмов существенно зависит от модальности целевой функции и связности допустимого множества. В частности на унимодальной функции и при связном допустимом множестве более предпочтительным является алгоритм, реализующий ЭС, а на полимодальных функциях и при несвязном допустимом множестве лучше работает ГА. Это может быть объяснено представлением объектных переменных, с которыми работают алгоритмы. Так как ЭС работают на уровне фенотипа, т.е. подвергают преобразованиям сами объектные переменные, то понятно, что преодолеть "разрывы" в допустимой области для ЭС довольно трудно, также как и пройти в зону притяжения другого локального минимума. ГА, работающий с генотипом, лишен этих недостатков, так как небольшие мутации битовой строки могут означать на уровне фенотипа весьма резкие скачки по

допустимой области, которые и обеспечивают просмотр изолированных частей несвязного множества и переход в зоны притяжения других локальных минимумов. Отсюда же вытекает и менее эффективная работа ГА в более простых случаях.

Генетического программирование

Для решения многих практических задач наиболее естественным представлением решений являются компьютерные программы. Обычно компьютерные программы – иерархические композиции процедур и входных данных, отражающих состояние системы. Одна из центральных задач теории вычислительных систем (computer science) - научить компьютер решать поставленную задачу, не объясняя ему как это делать. Генетическое программирование позволяет сделать это путем создания работающих компьютерных программ исходя из высокоуровневой постановки задачи. Генетическое программирование достигает поставленной цели автоматического программирования или программного синтеза (automatic programming, program synthesis) путем выращивания популяций компьютерных программ, используя принцип естественного отбора Дарвина, и основанные на генетических принципах операторы, которые могут включать репродукцию, скрещивание и мутацию.

Каким образом генетические алгоритмы могут быть использованы для автоматического программирования?

В 1975 году Джон Холланд предложил некую модификацию генетического алгоритма известную как классификатор (classifier system), состоящий из IF…THEN правил. Например, бит равный 1 означает выполнение условия IF, бит равный 0 – не выполнение, последующие биты определяют действие при выполнении (невыполнении) условий.

В 1985 году Н. Крамер предложил вычислять непосредственно код компьютерных программ и продемонстрировал эволюцию простых арифметических выражений. Он также предложил представление решений в виде деревьев.

Джон Коза (John R. Koza) в 1987 продемонстрировал эволюцию выражений языка программирования LISP (LISP S-expression), которые по сути являются компьютерными программами. Он впервые использовал термин «генетическое программирование». В 1989 году Коза показал применение генетического программирования для решения различных задач.

Настоящее развитие генетическое программирование получило после выхода в 1992 году книги Джона Козы «Genetic Programming: On the Programming of Computers by Means of Natural Selection & Genetics», в которой он продемонстрировал области применения метода, а также численные результаты экспериментов и некоторые практические рекомендации.

Области применения метода ГП различны, тем не менее, вот не которые рекомендации, когда использование метода ГП преимущественно:

- Проблемы, в которых классический математический анализ не позволяет получить аналитическое решение;
- Проблемы, в которых связь между независимыми и зависимыми переменными не известна, либо известная связь подвергается сомнению;
- Проблемы, в которых определение формы и размера решения является основной частью данной проблемы;
- Проблемы, в которых допустимо получение приблизительного, не точного решения;
- Проблемы, в которых большие массивы данных (представленные в удобной для компьютерной обработки форме) должны обрабатываться, проверяться, классифицироваться, интегрироваться и т.д.;
- Проблемы, в которых небольшие улучшения решения легко достижимы и высоко поощряются [Koza, Future work and practical Application of Genetic Programming – Handbook of Evolutionary Computation, 1996].

Примеры конкретных задач:

- Задачи автоматического управления;
- Задачи обработки больших массивов данных (последовательности данных из биологии, астрономические, геологические наблюдения, финансовые временные ряды, базы данных маркетинговых исследований, прогнозы погоды и т.д.);
- Обработка сложных структур данных;
- Эволюция кода-ассемблера, эволюция компьютерных программ;
- Автоматическое программирование мульти-агентных систем;
- Формирование ADF-функций (автоматически определяемых) и макросов;
- Кодирование клеточных структур (cellular encoding) – выращивание архитектур нейронных сетей, электрических схем и др. сложных структур;
- Распараллеливание алгоритмов;
- Оптимизация;

- Аппроксимация, классификация, обобщение, кластеризация;
- Вычисления на аппаратном обеспечении с «мягкой архитектурой» (evolvable hardware)
- И другие…

По сути, генетическое программирование является некой модификацией генетического алгоритма, основное различие – в представлении решений. Решения в генетическом программировании могут иметь различную форму и размер, в генетическом алгоритме – это строки фиксированной длины. Наиболее распространенным является представление в виде деревьев.

Деревом будем называть направленный граф, в котором каждая последующая вершина связанна с одной и только одной предыдущей. Вершины дерева являются элементами одного из двух множеств:

Множество всех возможных внутренних вершин дерева называется *функциональным множеством* F.

Множество всех возможных внешних вершин дерева называется *терминальным множеством* T.

Элементы функционального множества обычно являются рабочими блоками программы (процедурами, функциями), элементы терминального множества – входными данными (переменными и константами). Пример дерева в методе ГП представлен на рис. 4.1.

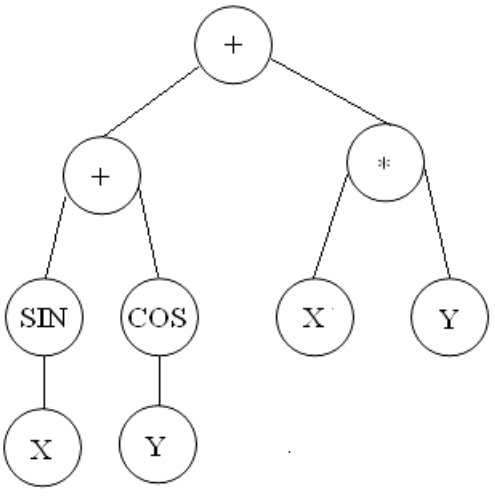


Рис. 1. Пример решения в методе ГП

Объединение функционального и терминального множеств назовем *универсальным множеством* $C = F \bigcup T$.

Для решения поставленной задачи методом генетического программирования множества F и T должны удовлетворять следующим требованиям:

Замкнутость. Для обеспечения замкнутости любой элемент из множества F должен принимать в качестве аргумента любой элемент множества C.

Например, множество $C = \{+, -, \times, \div, X, R\}$ не является замкнутым, так может привести к делению на нуль (R – множество вещественный чисел).

Достаточность. Для обеспечения достаточности элементы множества C должны быть выбраны таким образом, чтобы можно было бы решить поставленную задачу.

Пользователь метода генетического программирования должен знать или предполагать некоторую композицию элементов функционального и терминального множеств, которая должна дать решение поставленной задачи. Следует отметить, что существуют задачи, для которых такие композиции точно известны (например, задачи, использующие булевы переменные).

Следует отметить, что экспертные знания о природе задачи могут быть включены во множества F и T в виде специальных функций или термов. В частности, термами могут быть специфические для предметной области константы (например, число π, ускорение свободного падения, постоянная Планка и т.п.). Функции или комбинации функций, широко применяемые в исследуемой области, могут быть включены в функциональное множество, что избавляет от необходимости генерирования их в ходе работы алгоритма (например, факториалы, комбинаторные формулы и т.п.).

Общая схема метода генетического программирования, аналогичная схеме стандартного ГА, представлена ниже:

1. Генерируем начальную популяцию A^0 одним из методов выращивания, используя элементы множеств F и T.
2. На шаге g вычисляем пригодность решений в популяции A^g.
3. Применяем операторы селекции, клонирования, скрещивания и мутации к поколению A^g и формируем новую популяцию A^{g+1}.
4. Стоп, если допустимое решение найдено, иначе $g \leftarrow g + 1$ и переходим к шагу 2.

Рассмотрим шаги алгоритма подробней.

Для инициализации популяции используются следующие *методы выращивания* деревьев:

Полный метод (full method). Задается глубина дерева d. В вершины на глубине i ($i=\overline{1,d-1}$) случайным образом выбираются элементы из функционального множества F. На глубине d выбираются элементы из терминального множества T. Таким образом, получается полное дерево глубины d.

Метод выращивания (grow method). Задается глубина дерева d. В вершины на глубине i ($i=\overline{1,d-1}$) случайным образом выбираются элементы из функционального множества F или из терминального множества T (в этом случае рост текущей ветви заканчивается). На глубине d выбираются элементы из терминального множества T. Таким образом, выращивается дерево глубины не больше, чем d.

Комбинированный метод (ramped half and half). Часть популяции выращивается полным методом, другая - методом роста.

Функция пригодности (fitness function) является оценкой качества полученного решения, аналогична пригодности в ГА.

Оператор селекции – оператор, посредством которого индивиды выбираются для порождения потомков и формирования новой популяции. Наиболее приспособленные особи должны выбираться с большей вероятностью для сохранения своих генов в следующем поколении. Схемы селекции, применяемые в методе ГП аналогичны схемам селекции, применяемым в ГА.

Существует две схемы скрещивания в методе генетического программирования: *стандартное скрещивание* (standard crossover) и *одноточечное скрещивание* (one-point crossover).

Стандартное скрещивание осуществляется следующим образом. Выбираются родительская пара. У каждого из родителей выбирается точка скрещивания (дуга в графе). Родители обмениваются генами (поддеревьями), находящимися ниже точки скрещивания. Полученная пара является потомками.

Пример стандартного скрещивания (рис. 2):

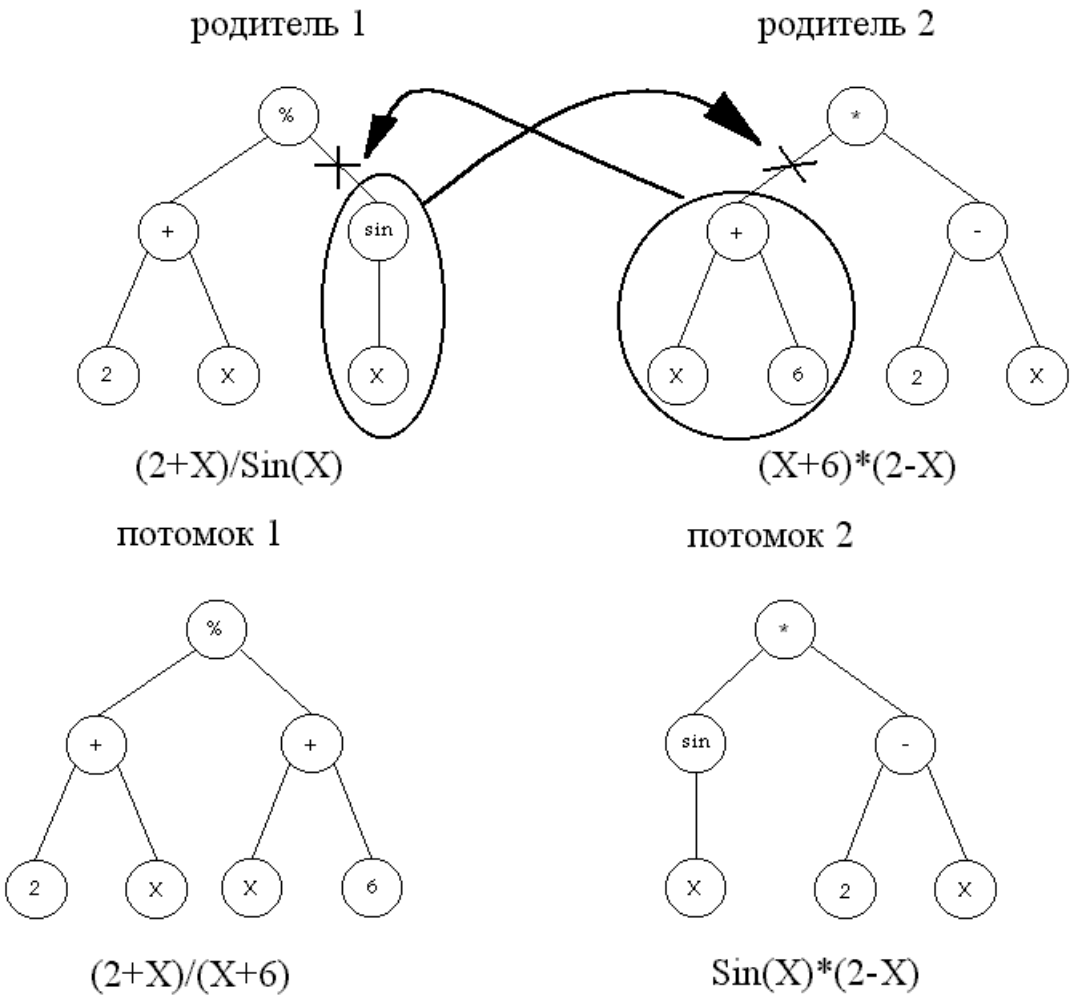


Рис. 2. Пример стандартного скрещивания в методе генетического программирования

При одноточечном скрещивании у родительской пары выбирается общая точка скрещивания, далее скрещивание осуществляется по стандартной схеме. Общая точка выбирается в общей области деревьев родителей, получаемой наложением одного дерева на другое начиная с корня.

Пример одноточечного скрещивания (рис. 3). Жирной линией отмечены связи, которые могут быть выбраны в качестве общей точки скрещивания:

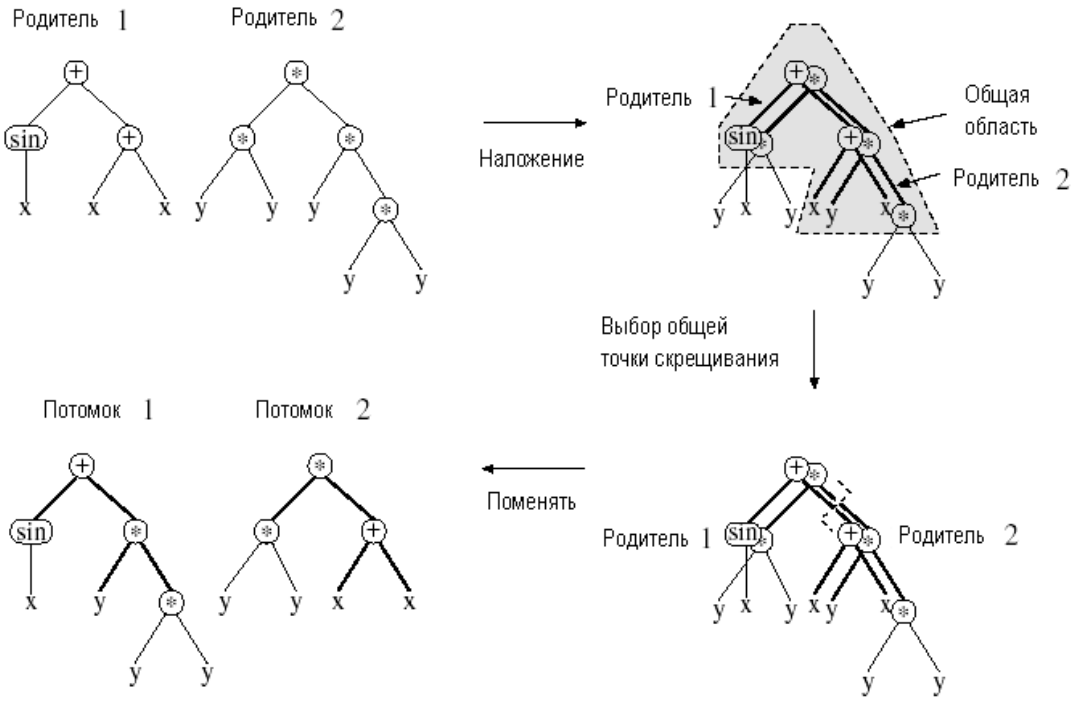


Рис. 3. Пример одноточечного скрещивания в методе генетического программирования

Мутация состоит из выполнения (обычно небольших) изменений в значениях одного или нескольких генов в хромосоме. Мутация применяется с очень низкой вероятностью $p_m \in [0.001, 0.01]$. Однако существует ряд задач, решаемых с помощью метода генетического программирования, где допустимое решение может быть найдено экстенсивным использованием оператора мутации.

Существует две схемы применения оператора мутации в генетическом программировании.

При использовании первой схемы (точечная мутация), случайно выбранный узел в дереве меняется на случайно выбранный элемент того же типа. Т.е. выбранный ген заменяется случайно выбранным элементом терминального множества, если этот ген принадлежит терминальному множеству, или элементом функционального, если ген – элемент функционального множества (рис. 4).

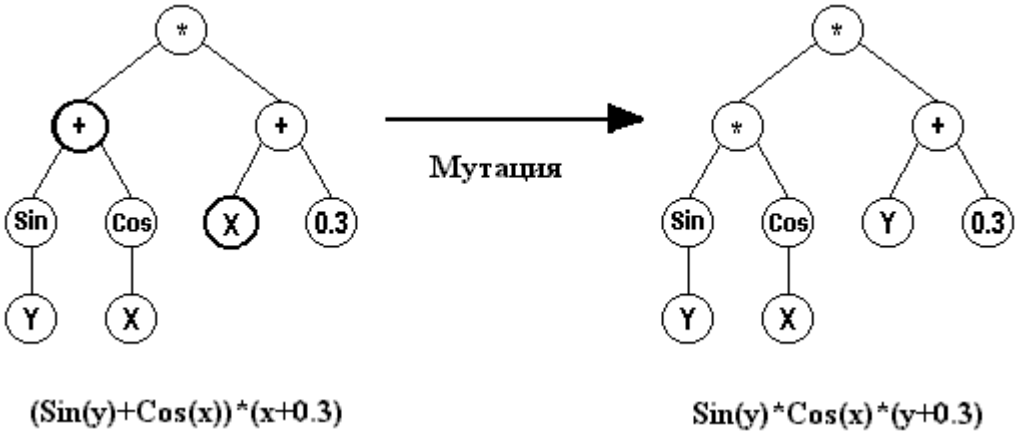


Рис. 4. Пример мутации по схеме 1

При использовании второй схемы, выбранное поддерево заменяется поддеревом, выращенным методом роста, новое поддерево имеет глубину не больше, чем глубина выбранного поддерева (рис. 5).

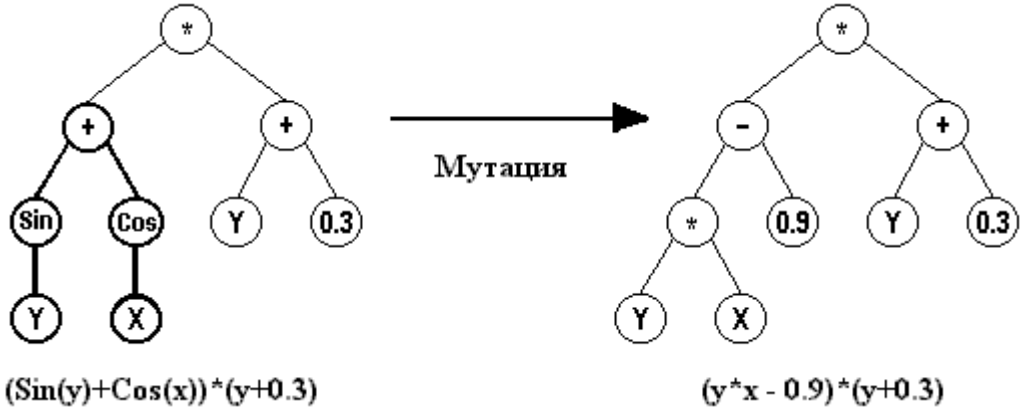


Рис. 5. Пример мутации по схеме 2

Решение задачи символьной регрессии

В большинстве численных методов идентификации для аппроксимации экспериментальных (статистических) данных используются регрессионные модели. Регрессия - это оценка функциональной зависимости условного среднего значения результативного признака Y от факторных признаков $X = (x_1, x_2, ..., x_N)$, т.е. регрессия – это некоторая усредненная количественная зависимость между выходными и входными переменными $Y = M\{Y \mid X\}$.

В **регрессионном анализе** задача регрессии решается путем выбора функциональной формы и последующим нахождением ее численных

коэффициентов (любым подходящим методом). Например, линейная ($\hat{y}_x = a_0 + a_1 x$), квадратичная ($\hat{y}_x = a_0 + a_1 x + a_2 x^2$), полиномиальная регрессия и др. [5]. Очевидно, что качество аппроксимации при данном подходе напрямую зависит от выбора конкретной параметрической модели.

Непараметрические методы не требуют дополнительной информации о структуре (параметрической) моделируемого объекта. Задача регрессии решается путем восстановления плотности вероятности [16, 23]:

$$M\{Y \mid x\} = \int_{\Omega(y)} y \cdot P(y \mid x) = \eta(x)$$

,

$$\hat{M}(Y \mid x) = \eta_S(x) = \frac{\sum_{i=1}^{S} y_i \cdot \prod_{j=1}^{N} \Phi\left(\frac{x^j - x_i^j}{C_S}\right)}{\sum_{i=1}^{S} \prod_{j=1}^{N} \Phi\left(\frac{x^j - x_i^j}{C_S}\right)},$$

где $\Phi(*)$ - заданная колоколобразная функция, C_S - параметр размытости, s – объем выборки.

Нейросетевой подход заключается в построении и дальнейшем обучении сложных математических структур – искусственных нейронных сетей, имитирующих функционирование биологических нейронов. Математическая модель нейрона имеет вид:

$$y = f(S),\ S = \sum_{i=1}^{N} w_i \cdot x_i + b_i$$

,

где x_i - независимая переменная, w_i - вес синаптической связи, b_i - смещение, $f(*)$ - функция активации нейрона.

Т.е. функция активации нейрона $f(*)$ - некоторое нелинейное преобразование взвешенной суммы входных переменных. Задавая различные архитектуры сетей, состоящие из множества нейронов можно описывать сложные нелинейные зависимости.

Недостаток численной модели, при всем ее удобстве для принятия решений, заключается в том, что она, по сути, является «черным ящиком» (моделью, в которой перечисляются входные и выходные связи системы со средой, а информация о внутренней структуре «ящика» полностью отсутствует). Решение задачи символьной регрессии могло бы значительно продвинуть ситуацию.

Задача символьной регрессии заключается в нахождении математического выражения в символьной форме, аппроксимирующего зависимость между конечным набором значений независимых переменных и соответствующими значениями зависимых переменных.

Таким образом, символьная регрессия дает нам не только вычислительную процедуру, но и формулу (символьное математическое

выражение), которую можно было бы подвергнуть содержательному анализу, упростить, а затем и уточнить. Однако на современном этапе методы символьной регрессии не разработаны достаточно хорошо. Генетическое программирование (ГП) - один из самых многообещающих подходов в данном направлении.

Для решения задачи символьной регрессии с помощью генетического программирования необходимо выполнить следующие подготовительные шаги, а именно определить:

1. терминальное множество,
2. функциональное множество,
3. функцию пригодности,
4. параметры, контролирующие работу алгоритма,
5. критерий остановки.

Первый этап определяет множество термов, из которых будет строиться решение. В задаче символьной регрессии терминальное множество содержит набор переменных $x_i, i = \overline{1,N}$ (где N - размерность поставленной задачи) и набор констант $const$, т.е.

$$T = \left\{ \left. x_i \right|_{i=1}^{N}, \left. const_j \right|_{j=1}^{K} \right\}.$$

На втором этапе пользователь метода должен определить множество функций, которые будут использованы для построения решений. Пользователь должен априори предполагать некоторую комбинацию функций, которые могли бы содержаться в решении задачи. Функциональное множество может содержать:

- арифметические операции ($+,-,\times,\div$),
- математические функции (Sin, Cos, Exp, Log и т.д.),
- булевы операции (AND, OR, XOR, NOT),
- специальные предопределенные функции (Automatically Defined Functions - ADFs).

Необходимо помнить, что заданное универсальное множество $C = F \cup T$ должно удовлетворять условию замкнутости. Например, для исключения деления на нуль вводят «защищенное деление», которое возвращает единицу, когда знаменатель равен нулю.

Третий этап – определить функцию пригодности, вычисляемую по заданной обучающей выборке. Для решения задачи символьной регрессии предложена следующая функция пригодности (4.1)

$$fitness(A_j) = \frac{1}{1 + Error(A_j)} - complexity(A_j)$$
$$Error(A_j) = \sum_{k=1}^{S} (y_k - evaluate(A_j, \overline{x_k}))^2 \quad (4.1)$$

где $fitness(A_j)$ - пригодность решения A_j, $Error(A_j)$ - квадратичная ошибка аппроксимации, вычисляемая по всем точкам обучающей выборки, S - объем обучающей выборки, $evaluate(A_j, \overline{x_k})$ - значение полученного выражения A_j в точке $\overline{x_k}$, $complexity(A_j)$ - некоторый штраф на сложность дерева, выраженный в суммарном количестве вершин.

При аппроксимации зависимостей большой размерности может возникнуть ситуация, когда в результате применения оператора скрещивания или мутации будет потеряны одна или несколько переменных. Однако при оценке по критерию (4.1) дерево, которое содержит не все переменные, может иметь ошибку аппроксимации меньше, чем дерево, содержащее все переменные. Поэтому для задач большой размерности предложены следующие модификации критерия (4.1):

$$fitness(A_j) = \left[\frac{1}{1 + Error(A_j)} - complexity(A_j) \right] \cdot \frac{n}{N},$$
$$Error(A_j) = \sum_{k=1}^{S} (y_k - evaluate(A_j, \overline{x_k}))^2 \quad (4.2)$$

$$fitness(A_j) = \left[\frac{1}{1 + Error(A_j)} - complexity(A_j) \right] \cdot k^{n-N},$$
$$Error(A_j) = \sum_{k=1}^{S} (y_k - evaluate(A_j, \overline{x_k}))^2, \quad (4.3)$$
$$k > 1$$

где $fitness(A_j)$ - пригодность решения A_j, $Error(A_j)$ - квадратичная ошибка аппроксимации, вычисляемая по всем точкам обучающей выборки, S - объем обучающей выборки, $evaluate(A_j, \overline{x_k})$ - значение полученного выражения A_j в точке $\overline{x_k}$, $complexity(A_j)$ - некоторый штраф на сложность дерева, выраженный в суммарном количестве вершин, N - число переменных задачи, n - число различных переменных, представленных в решении A_j.

В критерии (2.2) штраф за отсутствие в решении переменных пропорционален количеству отсутствующих переменных, в критерии (2.3) штраф гораздо жестче – имеет экспоненциальную зависимость.

Параметры, контролирующие работу алгоритма обычно включают:

- размер популяции,

- метод роста,
- максимальную начальную глубину деревьев,
- метод селекции и его параметры (например, размер турнира для турнирной селекции),
- вероятность и тип мутации.

В качестве критерия остановки работы алгоритма можно выбрать один из следующих:

- достигнуто заданное число поколений (итераций),
- достигнута заданная точность аппроксимации,
- относительное изменение средней пригодности не превышает заданной величины.

Общая схема алгоритма для решения задачи символьной регрессии с помощью метода ГП имеет следующий вид:

1. Инициализировать популяцию A^0 случайным образом одним из методов роста.
2. Оценить популяцию, используя критерий (4.1), (4.2) или (4.3).
3. Если выполняется критерий остановки, то КОНЕЦ, иначе шаг 4.
4. На итерации g для поколения A^g произвести селекцию одним из предложенных методов отбора.
5. Отобранные особи скрещиваются, подвергаются мутации и оцениваются – создают промежуточную популяцию.
6. В поколение A^{g+1} попадает часть особей из популяции A^g и часть из промежуточной популяции.
7. Переход к шагу 2.

ЗАКЛЮЧЕНИЕ

Рассмотренные в данном разделе алгоритмы являются простейшими в том смысле, что позволяют решать относительно простые задач моделирования и оптимизации. В частности - задачи безусловной однокритериальной оптимизации и задачи моделирования в простом случае построения аналитической зависимости по репрезентативным данным. Кроме того, настройка эволюционных алгоритмов выполнялась путем исчерпывающего статистического анализа, что не может быть признано удачным решением, т.к. при этом конкретная задача решается многократно, причем в большинстве прогонов не эффективным алгоритмом.

Ниже мы рассмотрим эволюционные методы решения более сложных задач оптимизации – условных, многокритериальных, комбинаторных – и более сложные подходы к моделированию сложных систем эволюционными

алгоритмами, а также обсудим известные подходы к более эффективному выбору настроек эволюционных алгоритмов.

МОДИФИЦИРОВАННЫЕ ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ МОДЕЛИРОВАНИЯ И ОПТИМИЗАЦИИ СЛОЖНЫХ СИСТЕМ

ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧ УСЛОВНОЙ ОПТИМИЗАЦИИ

Стандартный генетический алгоритм

В общем виде работу генетического алгоритма можно представить следующим образом:

1. Инициализировать случайным образом популяцию решений.
2. С помощью оператора селекции выбрать часть популяции (родителей) для порождения потомков.
3. Применить оператор скрещивания.
4. Новые решения (потомки) подвергаются мутации.
5. Формируется новая популяция: выбрать решения из родителей и потомков.
6. Повторять шаги 2 – 5 пока не выполнится условие остановки.

Напомним, что детальное описание каждого из операторов генетических алгоритмов приводилось в предыдущих лекциях.

Условная оптимизация с ГА

На шагах 2 и 5 стандартного ГА (селекция родителей и формирование нового поколения) выбор индивида из популяции происходит в зависимости от его степени пригодности. В общем случае, чем индивид более пригоден, тем у него больше шансов быть отобранным. Степень пригодности вычисляется через функцию пригодности.

Один из недостатков классических эволюционных алгоритмов состоит в отсутствии механизма учета ограничений оптимизационной задачи. Можно указать несколько возможных методов непосредственного решения этой проблемы.

Метод статических штрафов

Данный метод предполагает формирование для каждого ограничения семейства интервалов, которые определяют соответствующие коэффициенты штрафов.

Метод работает по следующей схеме:

- Для каждого ограничения определяется несколько уровней (l) его нарушения,
- Для каждого уровня нарушений каждого ограничения определяется коэффициент штрафа $R_{ij}(i = 1,2,...,l,\ j = 1,2,...,m)$; большие значения коэффициентов соответствуют большим уровням нарушения ограничений,

- Стартовая популяция индивидов задается случайным образом (могут присутствовать допустимые и недопустимые индивиды),

- Пригодность индивидов определяется по формуле:

$$fitness(\bar{x}) = f(\bar{x}) + \sum_{j=1}^{m} R_{ij} f_j{}^2(\bar{x}),$$

где $f_j(\bar{x})$ – штраф за нарушение j-го ограничения.

Недостатком метода является большое число параметров. Для m ограничений метод требует определения $m(2l+1)$ параметров, где m задает число интервалов для каждого ограничения, $l+1$ параметров для каждого ограничения – это границы интервалов (уровней нарушения ограничений), и l параметров для каждого ограничения определяют коэффициенты штрафов R_{ij}.

Метод динамических штрафов

Пусть решается следующая задача условной оптимизации:

$$f(x) \to extr$$
$$\begin{cases} g_j(x) \le 0, j = \overline{1,r} \\ h_j(x) = 0, j = \overline{r+1,m} \end{cases}$$

В общем виде, пригодность индивида x вычисляется по формуле:

$$fitness(x) = f(x) + \delta \cdot \lambda(t) \cdot \sum_{j=1}^{m} f_j^{\beta}(x),$$

где t – номер текущего поколения, $\delta = 1$ если решается задача минимизации, $\delta = -1$ если решается задача максимизации, $f_j(x)$ - штраф за нарушение j-го ограничения, β - вещественное число.

Штрафы в данном методе вычисляются динамически, в зависимости от степени нарушения ограничений, по следующей формуле (на t-ой итерации).

$$f_j(x) = \begin{cases} \max\left\{0, g_j(x)\right\}, j = \overline{1,r} \\ \left|h_j(x)\right|, j = \overline{r+1,m} \end{cases}$$

где, $g_j(x) \le 0, h_j(x) = 0$ - ограничения задачи, $\lambda(t) = (C \cdot t)^{\alpha}$

$$fitness(x) = f(x) + \delta \cdot \left(C \cdot t\right)^{\alpha} \cdot \sum_{j=1}^{m} f_j^{\beta}(x).$$

Рекомендованные значения $C = 0.5, \alpha = \beta = 2$.

Метод требует намного меньше параметров, чем предыдущий (их число не зависит от количества ограничений). Вместо выбора из набора фиксированных уровней нарушения ограничений в данном методе штраф рассчитывается динамически.

Метод «смертельных» штрафов

Метод заключается в отсечении («убийстве») недопустимых точек, т.е. эти точки не принимают более участия в воспроизводстве. Для некоторых

задач этот простой метод может давать хорошие результаты. Для использования данного метода необходимо инициализировать стартовую популяцию допустимыми значениями.

Метод адаптивных штрафов

Экспериментирование с вышеизложенным методом показало его некоторую ограниченность (в частности, он не отличает ситуаций, когда лучшая точка допустима, от ситуации, когда она недопустима, что необходимо для алгоритмов прямого поиска). Поэтому, была предложена модификация метода динамических штрафов, названная методом адаптивных штрафов. В отличие от предыдущего, в данном методе штрафные функции зависят не только от номера итерации, но и от количества попаданий лучшего представителя популяции на каждом шаге в допустимую или недопустимую области.

Эта зависимость выглядит так:

$$\lambda(t+1,\vec{\beta}) = \begin{cases} \dfrac{\lambda(t)}{\beta_1}, & \text{если } \vec{b}^i \in F \text{ для всех } t-k+1 \le i \le t \\ \beta_2 \cdot \lambda(t), & \text{если } \vec{b}^i \in S-F \text{ для всех } t-k+1 \le i \le t \\ \lambda(t), & \text{в противном случае} \end{cases}$$

где $\vec{b}^i$ - лучший индивид i-й популяции, $\beta_1, \beta_2 > 1$ и $\beta_1 \neq \beta_2$ (для избегания зацикливания).

В данном методе происходит уменьшение штрафа на (t+1)-м шаге в случае, если лучший индивид популяции (в терминах функции пригодности) на протяжении последних k итераций принадлежал допустимой области. Если же лучший индивид популяции в течение того же промежутка времени выходил за границы допустимой области, происходит увеличение штрафа.

Метод вводит три дополнительных параметра β_1, β_2, k. Метод адаптивных штрафов, таким образом, штрафует недопустимых индивидов не только в соответствии с тем, на сколько они сами нарушают ограничения, но и с учетом того, насколько ограничения нарушались их предшественниками. Исследования данного подхода на тестовых и реальных задачах показали его превосходство над другими методами штрафных функций. Существенно то, что это превосходство возрастает с ростом сложности решаемой задачи.

В теории оптимизации известен еще один подход к решению задач с ограничениями - метод обобщенного Лагранжиана, когда условный оптимум ищется как седловая точка функции Лагранжа. При этом в точке, являющейся решением, достигается минимум по переменным задачи и максимум по коэффициентам Лагранжа (для задачи минимизации). Если бы оптимальные коэффициенты Лагранжа были известны заранее, то решение задачи условной оптимизации было бы сведено к однократному решению

задачи безусловной оптимизации. Однако эти коэффициенты неизвестны и процесс решения условной задачи состоит в одновременном решении задачи безусловной минимизации по объектным переменным и задачи безусловной максимизации по коэффициентам Лагранжа. При таком подходе может быть применен ГА, если хромосома будет представлять собой объект, составленный из объектных переменных и коэффициентов Лагранжа. Однако при первом же взгляде становится заметно, что на самом деле речь идет о двух различных индивидах (один - набор объектных переменных, другой - набор коэффициентов Лагранжа), объединенных в одну хромосому. Учитывая же то, что при применении ГА есть возможность выбора своего набора генетических операторов для каждой компоненты индивида, становится окончательно ясно, что речь идет о двух различных популяциях, каждая из которых может эволюционировать самостоятельно, используя индивидов другой популяции только на стадии расчета функции пригодности. Это и приводит к идее коэволюционного ГА, в котором эволюционируют совместно две популяции, каждая из которых имеет свое представление хромосомы, свои параметры алгоритма и свою стратегию эволюции.

Коэволюционный алгоритм может быть применен не только для задач условной оптимизации, но и для других задач, где возникает необходимость отыскания седловой точки.

Минимаксная аппроксимация. Вещественная функция $f(x)$ задана на интервале $I = [a, b]$. Необходимо найти функцию $g(x)$, принадлежащую определенному классу функций F, такую, что она минимизирует максимальное расхождение между g и f на интервале I, т. е. задача имеет вид

$$\min_{g \in F} \max_{x \in I} \left| f(x) - g(x) \right| .$$

Если каждая функция $g \in F$ может быть однозначно определена некоторым набором m параметров $a \in S \subset \mathbf{R}^m$, то получаем задачу

$$\min_{a \in S} \max_{x \in I} \left| f(x) - g(a; x) \right| ,$$

эквивалентную исходной.

Множители Лагранжа в задачах условной оптимизации

Пусть решается задача условной оптимизации в виде

$$min\, f(\mathbf{X}),\ \mathbf{X} \in \mathbf{R}^n,\ g(\mathbf{X}) \le 0,\ i = 1, 2, \ldots, m.$$

Введя множители Лагранжа $Y \in R_+^m$, преобразуем исходную задачу к эквивалентной задаче отыскания седловой точки функции Лагранжа

$$L(X, Y) = \sum_{i=1}^{m} g_i(\mathbf{X}) \cdot y_i .$$

Решением данной задачи является точка $(\mathbf{X}^*, \mathbf{Y}^*)$, удовлетворяющая условию

$$L(\mathbf{X}^*,\mathbf{Y}) \le L(\mathbf{X}^*,\mathbf{Y}^*) \le L(\mathbf{X},\mathbf{Y}^*)\ \forall\ \mathbf{X} \in \mathbf{R}^n,\ \mathbf{Y} \in \mathbf{R}_+^m,$$

причем первая компонента $\mathbf{X}^*$ этой точки является решением исходной задачи. Таким образом, решение минимаксной задачи

$$\min_{\mathbf{X}\in\mathbf{R}^n}\ \max_{\mathbf{Y}\in\mathbf{R}_+^m} L(X,Y)$$

обеспечивает не только точку условного минимума $\mathbf{X}^*$, но и множители Лагранжа $\mathbf{Y}^*$, которые в некоторых приложениях не менее важны, чем объектные переменные сами по себе. Ограничения-равенства тоже могут рассматриваться при этом подходе с дополнением, что соответствующие им множители Лагранжа не имеют ограничения на знак.

Две популяции эволюционируют, используя каждая свою собственную процедуру ГА. Размеры и представление популяций, а также параметры ГА устанавливаются независимо, а согласование процессов эволюции выполняется через оценивание функции пригодности. Основой для вычисления функции пригодности служит функция *f(x,y)*, у которой *x* берется из популяции A, а *y* - из популяции B. Поэтому функция пригодности индивида из популяции A зависит от популяции B и наоборот. Работа ГА в популяции A (соответственно B) представляет собой минимизацию (максимизацию), а индивиды этой популяции представляют собой кодированные переменные *x* (переменные *y*), принадлежащие соответствующему множеству **X** (множеству **Y**).

Пригодность индивида $x \in \mathbf{X}$ из популяции A определяется выражением $f_{\mathrm{A}}(x) = \max_{y\in B} f(x,y)$, а пригодность индивида $y \in \mathbf{Y}$ популяции B – выражением $f_{\mathrm{B}}(y) = \min_{x\in A} f(x,y)$.

В обоих случаях лучше использовать ранговую схему селекции с тем, чтобы не возникало необходимости в масштабировании исходной функции *f*(*x*,*y*) при вычислении пригодности индивидов обеих популяций.

Общая структура коэволюционного ГА:

```
Procedure co-evolutionary_GA
begin
инициализировать популяцию A
инициализировать популяцию B
for k = 1,2,... c_max do
for i = 1,2,... a_max do
сгенерировать новую популяцию A
оценить A
for j = 1,2,... b_max do
сгенерировать новую популяцию B
оценить B
```

end

Процесс решения задачи состоит в поочередной работе ГА с обеими популяциями. Сначала ГА работает с популяцией A, давая ей эволюционировать определенное пользователем количество поколений a_{max}. Затем ГА работает с популяцией B в течение b_{max} поколений. Этот цикл повторяется c_{max} раз.

Общее число поколений вычисляется как $ng = c_{max} \cdot (a_{max} + b_{max})$. При этом можно рассмотреть два крайних варианта:

$$a_{max} = b_{max} = 1,\ c_{max} = ng/2 \text{ и}$$
$$a_{max} = b_{max} = ng/2,\ c_{max} = 1.$$

Оба варианта не очень похожи на процедуру, которую можно было бы порекомендовать. В первом случае обе популяции вынуждены эволюционировать в условиях постоянно изменяющейся поверхности отклика так, что каждое новое поколение будет сталкиваться с существенно другой ситуацией. Во втором случае популяция B начнет меняться только после того, как популяция A уже достигла сходимости при неизменной исходной популяции B и более меняться не будет. Интуитивно ясно, что наилучший результат будет достигаться, если каждой популяции предоставить возможность пройти несколько поколений перед тем, как она будет остановлена для работы с другой популяцией. Таким образом, a_{max} и b_{max} являются параметрами алгоритма, от которых зависит его эффективность и которые должны быть настроены в процессе работы.

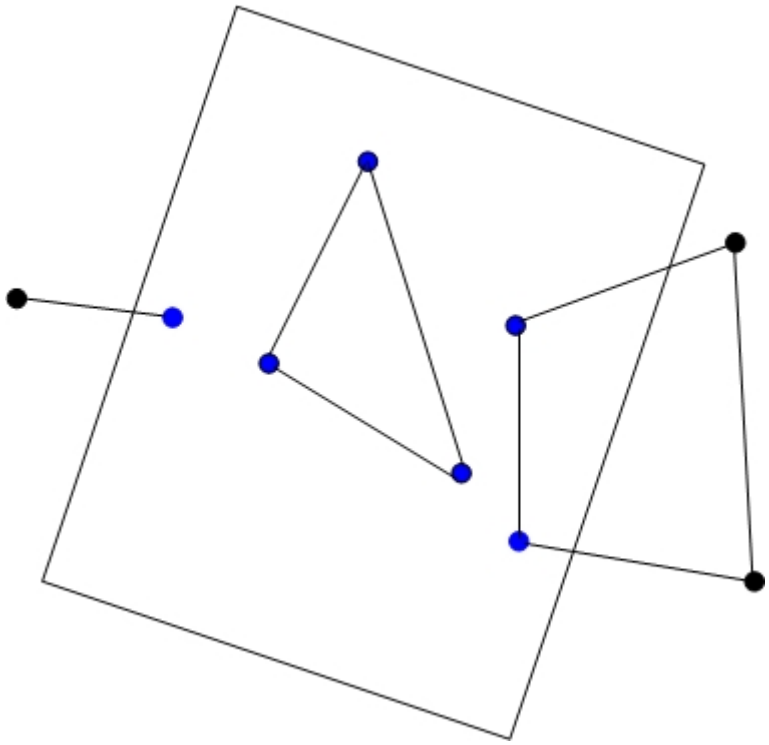
Специализированные генетические операторы

Все описанные выше подходы к обработке ограничений эволюционными алгоритмами обладают существенным недостатком - они очень трудоемки. При анализе реальных практических задач можно видеть, что значительная часть ограничений является линейной относительно переменных. Это позволяет предложить оператор скрещивания, автоматически обеспечивающий допустимость потомка при допустимых родителях. Этот же оператор позволит довольно быстро получить пригодного или почти пригодного потомка в случае, когда часть родителей оказывается непригодной.

Данный оператор селекции возвращает в качестве потомков выпуклую линейную комбинацию родителей. Как известно из теории выпуклых множеств, выпуклая линейная оболочка конечного числа точек в пространстве является многогранником, вершинами которого являются эти точки. Иллюстрация подхода приведена на рисунке.

Из теории линейного программирования известно, что совокупность линейных ограничений образует выпуклый многогранник. Это означает, что если взять два или более решения-родителя, удовлетворяющих линейным

ограничениям, то их выпуклая линейная оболочка целиком содержится в допустимом множестве. Т.е. каждая линейная комбинация родителей будет давать пригодного потомка. Достаточно указать лишь коэффициенты линейной комбинации. Предлагается выбирать их пропорционально пригодности родителей, с тем, чтобы потомок размещался ближе к более пригодному родителю и сам обладал более высокой пригодностью.



Если же часть решений-родителей не принадлежит допустимому множеству, то выбирать потомка из их выпуклой линейной оболочки можно с коэффициентами, пропорциональными степени допустимости родителей, с тем, чтобы потомок размещался ближе к допустимым родителям и сам оказывался допустимым или почти допустимым.

После получения потомков автоматически удовлетворяющих линейным ограничениям, удовлетворение остальных ограничений обеспечивается методами штрафных функций, описанными выше.

Метод поведенческой памяти

В некоторых задачах бывает достаточно трудно получить даже допустимое решение. Зачастую это происходит из-за того, что в задаче большое количество ограничений и/или допустимая область является «достаточной малой» по отношению ко всему поисковому пространству. Для преодоления этой трудности существует метод поведенческой памяти:

1. Генерация начальной популяции (допустимые и недопустимые индивиды).

2. Счетчик учитываемых ограничений j=1.

3. Текущая популяция является стартовой точкой для следующей фазы эволюции, в течение которой все точки, не удовлетворяющие одному из j-1 учитываемых ограничений, отбрасываются. Критерий остановки – удовлетворение j-му критерию заданной части популяции.

4. j=j+1; повторять два последних шага пока j не равно числу ограничений +1.

Для данного алгоритма очень важен порядок добавления учитываемых ограничений.

Во многих случаях этот метод работает лучше штрафных методов, т.к. зачастую небольшое по сравнению со значением целевой функции значение штрафа оставляет допустимую область скрытой среди пиков “оштрафованной” целевой функции, находящихся в недопустимых областях.

С другой стороны, как показали эксперименты, для данного метода не составляет труда быстро локализовать популяцию в рамках допустимой области при условии удачного выбора последовательности учета ограничений.

Выбор наиболее подходящего метода учета ограничений для конкретной задачи является самостоятельной проблемой, решаемой в большинстве случаев интуитивно и субъективно. Для определения наилучшего метода обычно приходится проводить полное тестирование с многократным решением задачи, как описано выше в модуле 1.

ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧ МНОГОКРИТЕРИАЛЬНОЙ БЕЗУСЛОВНОЙ ОПТИМИЗАЦИИ

Постановка многокритериальной задачи

Понятие «решение многокритериальной задачи» в известной мере субъективно и зависит от шкалы ценностей лица, принимающего решение (ЛПР). ЛПР не только принимает решение, но и готов нести ответственность за его последствия. Формально любая концепция выбора описывается функцией выбора. Поэтому можно связывать многокритериальную оптимизацию не с предпочтением ЛПР, а с функцией выбора на допустимом множестве значений векторного критерия.

Формулировка задачи векторной оптимизации

Многокритериальная оптимизация, также известная как векторная оптимизация, может быть определена как задача нахождения вектора переменных-решений, которые удовлетворяют ограничениям и доставляют оптимум вектор-функции, чьи элементы представляют собой целевые функции. Эти функции формируют математическое описание представления критериев, которые обычно находятся в конфликте друг с другом. Поэтому термин «оптимизировать» означает нахождение такого рода решения, которое бы давало значения всех целевых функций, приемлемых для ЛПР.

В общем виде многокритериальная задача оптимизации включает набор N параметров (переменных), множество K целевых функций и множество M ограничений. Целевые функции и ограничения являются функциями переменных.

Таким образом, при решении многокритериальной задачи необходимо найти оптимум по K критериям, а сама задача формально записывается следующим образом:

$$y = f(x) = (f_1(x), f_2(x), ..., f_K(x)) \to opt,$$

где $x = (x_1, x_2, ..., x_N) \in X$ – вектор решений, удовлетворяющий M ограничениям $g(x) = (g_1(x), g_2(x), ..., g_M(x)) \geq 0$, $y = (y_1, y_2, ..., y_K) \in Y$ – вектор целевых функций.

При этом X обозначает пространство решений, а Y – пространство целей или критериальное пространство. Ограничения $g(x) \geq 0$ определяют множество допустимых решений задачи.

Помимо самой постановки задачи многокритериальной оптимизации необходимо также ввести еще ряд определений и понятий, характерных для теории решения многокритериальных задач.

Допустимое множество D определяется как множество векторов-решений x, которые удовлетворяют ограничениям $g(x)$:

$$D = \{x \in X \mid g(x) \geq 0\}.$$

Тогда допустимая область в пространстве целей обозначается через:

$$Y_f = f(D) = \bigcup_{x \in D} \{f(x)\}.$$

Парето-оптимальность

Критерии могут быть согласованными, нейтральными или противоречивыми. В первом случае оптимизация одного из критериев приводит к улучшению других. Во втором случае оптимизация одного критерия никак не влияет на другие. Интерес представляет случай конфликтующих (противоречивых) критериев, когда попытка улучшить один из них приводит к ухудшению других. В таком случае решение возможно только на основе компромисса. Математическая модель компромисса в оптимизации обычно строится на основе понятия множества Парето, названного так в честь итальянского экономиста, первым исследовавшего такие модели в начале 20-го века.

Решение $x^0 \in D$ называется *эффективным* (недоминируемым, паретовским, неулучшаемым), если во множестве допустимых альтернатив D не существует решения, которое по целевым функциям было бы не хуже, чем x^0, и, по крайней мере по одной целевой функции было бы строго лучше, чем x^0. (Паретовская точка не может быть улучшена по совокупности всех целевых функций).

Представленное определение эффективного решения можно конкретизировать, введя понятие доминирования по Парето, которое является основополагающим в теории многокритериального выбора и более точно раскрывает сущность формулировки «одно решение лучше другого».

Концепция доминирования по Парето

Будем считать, что решается задача минимизации. Тогда понятие *Парето-доминирования* для двух любых векторов определяется тремя возможными вариантами:

1. Решение a доминирует решение b: $a \succ b$, если $f(a) < f(b)$.
2. Решение a слабо доминирует решение b: $a \succeq b$, если $f(a) \leq f(b)$.
3. Решения a и b несравнимы: $a \approx b$, если $f(a) \not\leq f(b) \wedge f(a) \not\geq f(b)$.

Таким образом, если x недоминируем относительно D, то он называется *Парето-оптимальным*, т.е. если не существует $y \in D : y \succ x$, то x является Парето-оптимальным.

При этом вектор $x \in D$, называется *недоминируемым* относительно некоторого множества $A \subseteq D$, если не существует $a \in A : a \succ x$.

Множество и фронт Парето

Множество всех эффективных точек называется *множеством Парето* в пространстве переменных (альтернатив), а их образ в критериальном пространстве – *фронтом Парето*.

С точки зрения концепции Парето-доминирования получаем следующую формулировку этого определения.

Функция $p(A)$ дает множество недоминируемых решений в A, $A \subseteq D$: $p(A)=\{a \in A \mid a -$ недоминируем относительно $A\}$. Тогда множество $p(A)$ назовем *недоминируемым множеством* относительно A, а соответствующее множество целевых векторов $f(p(A))$ – *недоминируемым фронтом* относительно A.

Множество $X_p = p(D)$ называется *Парето-оптимальным множеством*, а множество $Y_p = f(X_p)$ определяет *Парето-оптимальный фронт*.

Таким образом, имея множество Парето можно построить фронт Парето. Из всего вышесказанного можно сделать следующий вывод. Для любой допустимой точки, лежащей вне множества Парето, найдется точка во множестве Парето, дающая по всем целевым функциям значения не хуже, чем в этой точке и хотя бы по одной целевой функции – строго лучше. Отсюда следует, что решение многокритериальной задачи оптимизации целесообразно выбирать из множества Парето, т.к. любое другое, очевидно, может быть улучшено некоторой точкой Парето как минимум по одному критерию без ухудшения других критериев. С точки зрения математики решения из множества Парето не могут быть предпочтены друг другу, поэтому после формирования множества Парето задача может считаться математически решенной.

Многокритериальная оптимизация с ГА

Общая схема ГА

Напомним общую схему генетического алгоритма:

1. Инициализировать случайным образом популяцию решений.
2. С помощью оператора селекции выбрать часть популяции (родителей) для порождения потомков.
3. Применить оператор скрещивания.
4. Новые решения (потомки) подвергаются мутации.

5. Формируется новая популяция: выбрать решения из родителей и потомков.

6. Повторять шаги 2 – 5 пока не выполнится условие остановки.

Детальное описание каждого из генетических операторов приводилось в предыдущих лекциях.

Далее пойдет речь о четырех наиболее распространенных методах многокритериальной оптимизации с ГА:

1. VEGA – Vector Evaluated Genetic Algorithm;
2. FFGA – Fonseca and Fleming's Multiobjective Genetic Algorithm;
3. NPGA – Niched Pareto Genetic Algorithm;
4. SPEA – Strength Pareto Evolutionary Algorithm.

Эти методы реализуют различные схемы назначения пригодности и селекции. Рассмотрим каждую из них в отдельности.

Метод VEGA (Vector Evaluated Genetic Algorithm)

Метод VEGA, впервые предложенный Шафером в 1984 году, относится к категории селекции по переключающимся целевым функциям. Это означает, что селекция производится по пригодности индивидов для каждого из K критериев в отдельности. Тем самым промежуточная популяция заполняется равными порциями индивидов, отобранных по каждому из частных критериев.

Назначение пригодности и селекция в методе VEGA:

Входные данные: P_t (текущая популяция).

Выход: P' (промежуточная популяция).

1. Положить $k=1$, $k=\overline{1,K}$ и $P'=\varnothing$.

2. Для каждого индивида $i \in P_t$, $i=\overline{1,N}$, выполнить его оценивание, вычислив функцию пригодности используя значение k-го критерия.

3. Для $s=\overline{1,N/K}$ выбрать индивида $i \in P_t$, используя выбранный механизм селекции, основываясь на значении пригодности по данному критерию, и скопировать его в $P' : P' = P' + \{i\}$.

4. Положить $k = k+1$.

5. Если $k \le K$, перейти на шаг 2, иначе P' – результирующая промежуточная популяция.

Рассмотренный механизм селекции графически представлен на рисунке 1. Таким образом, для каждого из K критериев создается подпопуляция размером N/K, где N – размер всей популяции. В эти подпопуляции индивиды отбираются с помощью выбранной селекции относительно пригодности по каждому критерию в отдельности. Затем все подпопуляции смешиваются для получения популяции размером N. Далее, как обычно, осуществляются скрещивание и мутация согласно общей схеме ГА.

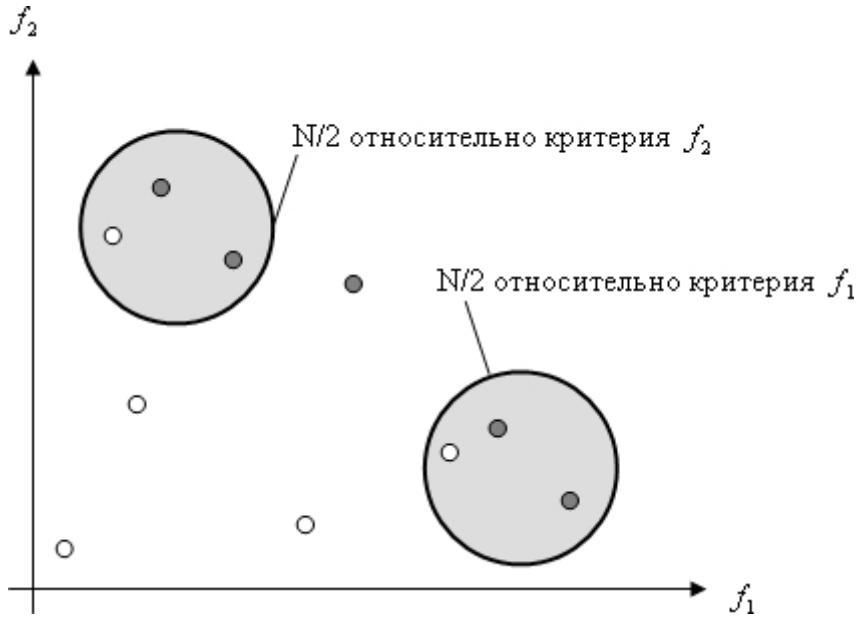


Рисунок 1 – Механизм селекции в методе VEGA

Метод FFGA (Fonseca and Fleming's Multiobjective Genetic Algorithm)

Метод FFGA (1993) использует основанную на Парето-доминировании процедуру ранжирования индивидов, где ранг каждого индивида определяется числом доминирующих его индивидов.

Назначение пригодности в методе FFGA:

Вход: P_t (популяция).

Выход: F (значения пригодности).

1. Для каждого индивида $i \in P_t$ вычислить его ранг $r(i) = 1 + \left|\{j : j \in P_t \wedge j \succ i\}\right|$.

2. Отсортировать популяцию согласно рангам *r*. Назначить каждому индивиду $i \in P_t$ посредством интерполирования от лучшего индивида к худшему "сырую" пригодность $F'(i)$, то есть лучшему индивиду $(r(i) = 1)$ назначается "сырая" пригодность $F'(i) = N$, а худшему $(r(i) \le N)$ – "сырая" пригодность $F'(i) = 1$.

3. Вычислить значение пригодности для каждого индивида $i \in P_t$ посредством усреднения и деления "сырой" пригодности между индивидами, имеющими одинаковый ранг:

$$F(i_r) = \sum_{i_r=1}^{n_r} F'(i_r) / n_r ,$$

i_r – индивиды, имеющие одинаковый ранг r, n_r – количество индивидов, имеющих ранг r. Деление пригодности осуществляется в критериальном пространстве.

Метод FFGA реализует лишь схему назначения пригодности, а для отбора индивидов в следующее поколение используется выбранная процедура селекции, которая осуществляется на основании полученных значений пригодности каждого отдельного индивида $F(i)$, $i = \overline{1, N}$.

На рисунке 2 представлены гипотетическая популяция и соответствующие ранги индивидов, назначенные согласно рассмотренной схеме метода FFGA. Индивиды, чьи соответствующие решения недоминируемы относительно всей популяции $m(P)$, имеют ранг 1, в то время как наихудший индивид, доминируемый всеми членами популяции получает ранг 10.

Таким образом, в методе FFGA используется понятие пригодности индивидов, но, в отличие от пригодности в алгоритме VEGA, она назначается не для каждого критерия в отдельности, а для индивида в целом и определяется не значениями целевых функций, а рангом каждого индивида в популяции, который в свою очередь основан на понятии Парето-доминирования.

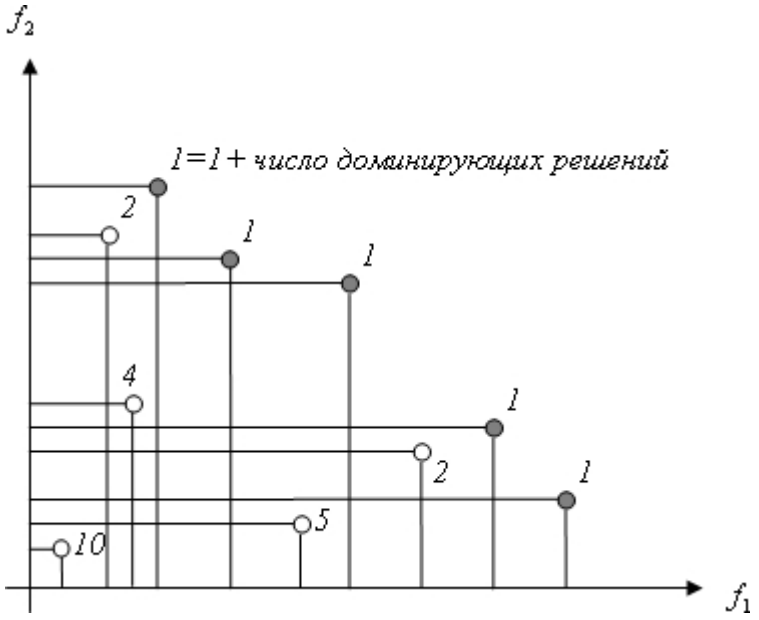


Рисунок 2 – Механизм селекции в методе FFGA

Метод NPGA (Niched Pareto Genetic Algorithm)

Третий рассматриваемый метод – NPGA (Horn, Nafpliotis and Goldberg, 1994, В.Р. Гарипов, 1998) – принципиально отличается от двух предыдущих, так как в нем заложен механизм поддержания разнообразия. Метод NPGA

представляет собой комбинацию турнирной селекции и концепции доминирования по Парето.

Назначение пригодности и селекция в методе NPGA:

Входными данными для этого метода являются:

P_t (популяция),

σ_{share} (радиус ниши),

t_{dom} (давление доминирования – количество индивидов в сравнительном множестве P_{dom}).

Выход: P' (промежуточная популяция).

1. Положить $s=1$ и промежуточную популяцию $P'=\varnothing$.

2. Выбрать двух индивидов $x, y \in P_t$ и сравнительное множество $P_{dom} \subseteq P_t$, состоящее из t_{dom} случайно выбранных индивидов популяции.

3. Проверить условие: если x недоминируем относительно P_{dom}, а y доминируется индивидами сравнительного множества, то победителем турнира является индивид x: $P'=P'+\{x\}$.

4. Иначе, если y недоминируем относительно P_{dom}, а x – доминируем, то победителем турнира становится индивид y: $P'=P'+\{y\}$.

5. Если победитель турнира не определился, то турнир разрешается делением общей пригодности:

а) вычислить количество индивидов в частично заполненной промежуточной популяции, которые находятся от индивида x на расстоянии, не превышающем радиус ниши σ_{share}: $n(x)=\left|\{l : l \in P' \wedge d(x,l) < \sigma_{share}\}\right|$. Проделать то же для индивида y.

б) если $n(x) < n(y)$, то $P'=P'+\{x\}$, иначе $P'=P'+\{y\}$, то есть, если количество индивидов, близких к индивиду y, больше числа индивидов, близких к индивиду x, то в промежуточную популяцию переходит индивид x, иначе там оказывается индивид y. Последней процедурой и обеспечивается разнообразие популяции.

6. Положить $s=s+1$. Если $s \le N$, перейти на шаг 2, иначе остановка.

На рисунке 3 проиллюстрирован механизм Парето-турнира: сопоставление 2-х индивидов популяции со сравнительным множеством t_{dom} индивидов. Участник турнира, обозначенный серой точкой, является его победителем, так как в отличие от другого участника, закодированный вектор данного индивида не доминируется представителями сравнительного множества.

Итак, в методе NPGA этап назначения пригодности заменяется модифицированной схемой деления пригодности с использованием понятия ниши, которая определяется для индивидов в пространстве целевых функций и обеспечивает возможность поддержания разнообразия, позволяя получить представительное множество Парето.

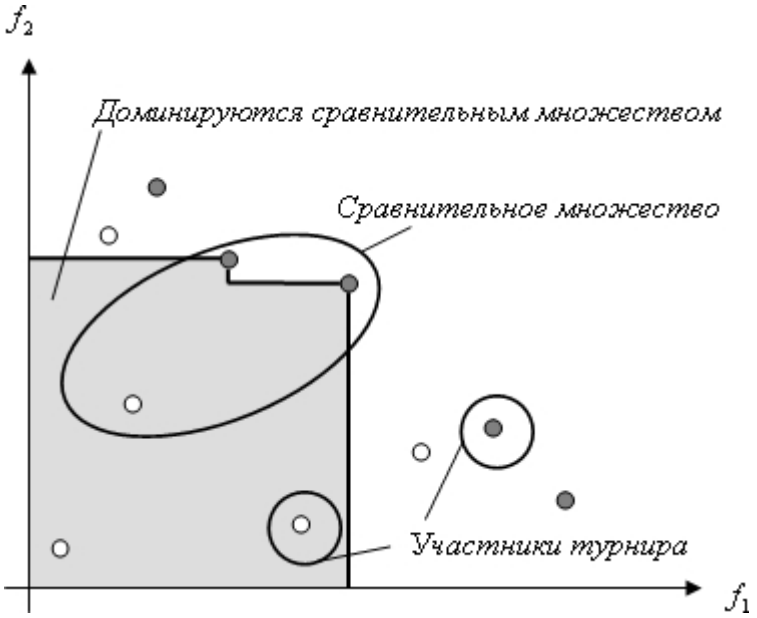


Рисунок 3 – Механизм селекции в методе NPGA

Метод SPEA (Strength Pareto Evolutionary Algorithm)

Метод SPEA (Zitzler and Thiele, 1998) в корне отличается от рассмотренных ранее методов, так как в нем:

- для назначения индивидам скалярного значения пригодности используется концепция Парето-доминирования;
- индивиды, недоминируемые относительно других членов популяции, хранятся внешне в специальном внешнем множестве;
- для уменьшения количества индивидов, хранящихся во внешнем множестве, выполняется кластеризация, что в свою очередь никак не влияет на приобретенные в процессе поиска свойства индивидов.

Уникальность и преимущества метода SPEA заключаются в том, что:

- он сочетает вышеперечисленные подходы в одном алгоритме;
- пригодность каждого индивида популяции в данном методе определяется только относительно индивидов внешнего множества, независимо от того, доминируют ли индивиды популяции друг друга;
- несмотря на то, что «лучшие» индивиды, полученные в предыдущих поколениях, хранятся отдельно – во внешнем множестве, все они принимают участие в селекции;
- для предотвращения преждевременной сходимости, в методе SPEA используется особый механизм образования ниш, где деление общей пригодности осуществляется не в смысле расстояния между индивидами, а на основе Парето-доминирования.

Процедура метода SPEA:

Вход:

N (размер популяции),

$\overline{N}$ (размер внешнего множества),

T (максимальное число поколений),

p_c (вероятность скрещивания),

p_m (вероятность мутации).

Выход: A (недоминируемое множество).

1. Инициализация:

Создать начальную популяцию P_0 и пустое внешнее множество $\overline{P_0}=\varnothing$. Положить $t=0$.

2. Модернизация внешнего множества:

Положить промежуточное внешнее множество $\overline{P'} = \overline{P_t}$.

а) скопировать индивидов, чьи векторы решений недоминируемы относительно P_t в $\overline{P'}$: $\overline{P'} = \overline{P'} + \{x \mid x \in P_t \wedge x \in p(P_t)\}$.

б) удалить тех индивидов из $\overline{P'}$, чьи соответствующие векторы решений слабо доминируемы относительно $\overline{P'}$, то есть, если существует пара индивидов $x, y \in \overline{P'}$ и $x \succeq y$, то $\overline{P'} = \overline{P'} - \{y\}$.

в) уменьшить посредством кластеризации (с параметрами $\overline{P'}$ и $\overline{N}$) число индивидов, хранящихся во внешнем множестве, и поместить результирующее уменьшенное множество в $\overline{P}_{t+1}$.

3. Назначение пригодности:

Вычислить значения пригодности индивидов в P_t и $\overline{P_t}$.

4. Селекция:

Положить $P'=\varnothing$ и для $s = \overline{1, N}$ выполнить:

а) случайным образом выбрать двух индивидов $x, y \in P_t + \overline{P}_t$,

б) если $F(x) < F(y)$, то $P' = P' + \{x\}$, иначе $P' = P' + \{y\}$ (в случае задачи минимизации).

5. Рекомбинация:

Рекомбинация осуществляется согласно общей схеме ГА.

6. Мутация:

Мутация осуществляется согласно общей схеме ГА.

7. Окончание:

Положить $P_{t+1} = P'''$ и $t = t+1$. Если $t \geq T$ или выполняется какой-то другой критерий остановки, тогда $A = p\left(\overline{P}_t\right)$ – есть искомое недоминируемое множество, иначе перейти на шаг 2.

Основной цикл описанного алгоритма схематично показан на рисунке 4. Шаг 1 не обозначен на представленном рисунке, так как инициализация начальной популяции осуществляется один раз и является обязательным первичным этапом любого ГА. Далее, в начале каждой итерации (Шаг 2) происходит модернизация внешнего множества $\overline{P}$ и его уменьшение, если

максимальный размер $\overline{N}$ был превышен. После этого происходит оценка индивидов из P относительно индивидов внешнего множества $\overline{P}$ для назначения индивидам популяции значений пригодности (Шаг 3). Следующий шаг (Шаг 4) представляет фазу селекции, когда отбираются индивиды из объединенного множества $\overline{P}+P$ для заполнения промежуточной популяции (в данной схеме используется турнирная селекция с удалением индивида, «проигравшего» турнир). Наконец, скрещивание и мутация (Шаг 5 и Шаг 6) осуществляются как соответствующие этапы общего эволюционно алгоритма.

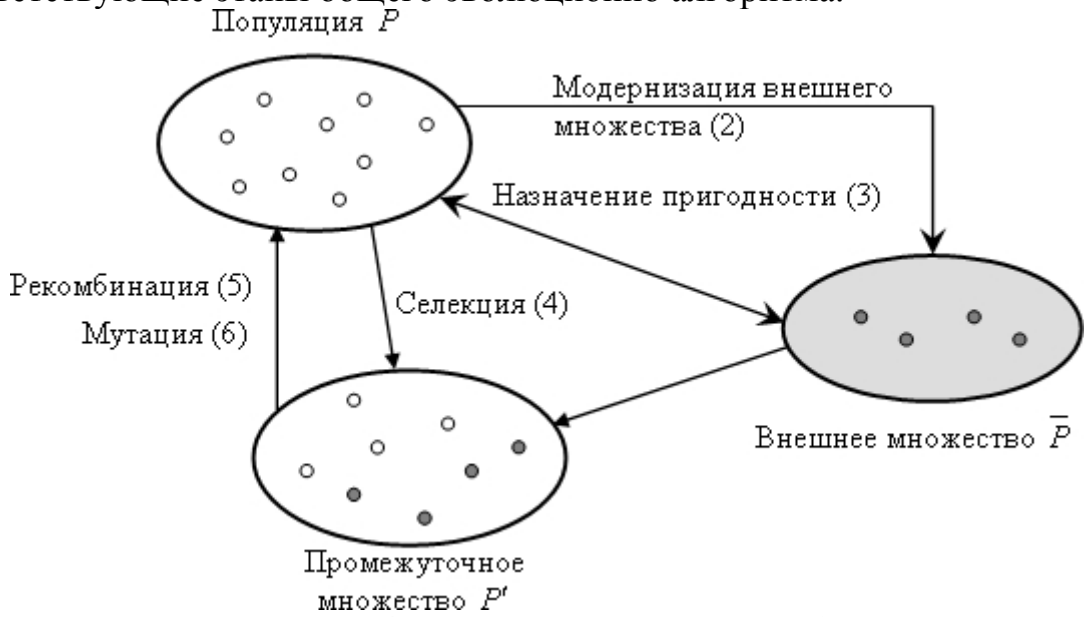


Рисунок 4 – Основные шаги метода SPEA

Однако стоит отметить, что этап назначения пригодности в схеме метода SPEA осуществляется не так, как в общем эволюционном алгоритме, а представляет собой модифицированную процедуру, основанную на концепции Парето-доминирования. Наряду с этим этапом, механизм кластеризации, используемый для уменьшения количества индивидов во внешнем множестве, также осуществляется по особой схеме. Алгоритмы пошаговой реализации обеих процедур представлены ниже.

Назначение пригодности в методе SPEA:

Вход:

P_t (популяция),

$\overline{P_t}$ (внешнее множество).

Выход: F (значения пригодности).

1. Каждому индивиду $x \in \overline{P_t}$ присваивается значение $S(x) \in [0,1)$, называемое «силой» индивида (отражает пригодность недоминируемого решения), которое пропорционально количеству членов популяции $y \in P_t$, для которых $x \succeq y$:

$$S(x) = \frac{\left|\{y : y \in P_t \wedge x \succeq y\}\right|}{N+1}.$$

Таким образом, пригодность индивида *x*, принадлежащего внешнему множеству, равна его «силе»: $F(x) = S(x)$.

2. Пригодность индивида $y \in P_t$ вычисляется посредством суммирования «сил» всех индивидов $x \in \overline{P_t}$, хранящихся во внешнем множестве, чьи векторы решений слабо доминируют *y*. К общему полученному числу добавляется единица, чтобы гарантировать, что индивиды внешнего множества $\overline{P_t}$ имеют лучшую пригодность, по сравнению с индивидами из P_t. Чем меньше пригодность, тем больше шансов у индивида j перейти в следующее поколение:

$$F(y) = 1 + \sum_{x \in \overline{P}_t, x \succeq y} S(x), \text{ где } F(y) \in [1, N).$$

Чтобы механизм ранжирования стал более понятным, обратимся к рисунку 5. Пять недоминируемых решений, обозначенных серыми точками, разбивают область критериального пространства на прямоугольники. Каждое подмножество $\overline{P'}$ внешнего множества $\overline{P}$ определяет один такой прямоугольник, представляющий область в критериальном пространстве. Эта область в общем доминируется всеми векторами решений в $\overline{P'}$. Например, на представленном рисунке светлый прямоугольник в нижнем левом углу доминируется всеми векторами решений в $\overline{P}$. Верхний левый светлый прямоугольник доминируется только $x \in \overline{P}$. Прямоугольники здесь рассматриваются как ниши, а цель состоит в том, чтобы так распределить индивидов по искусственно созданной сетке, чтобы прямоугольники (темные), доминируемые только несколькими $x \in \overline{P}$, содержали больше индивидов, чем прямоугольники (светлые), доминируемые большим количеством индивидов из $\overline{P}$, а также два прямоугольника (одинаково окрашенные) должны содержать одинаковое количество индивидов, если они доминируются равным количеством решений. Этот механизм отражает идею предпочтения индивидов, расположенных вблизи Парето-оптимального фронта и, в то же время, их распределения вдоль всего пространства поиска. Это можно видеть на рисунке 5: индивиды, сосредоточенные в светлых прямоугольниках, вдоль недоминируемых точек, получают лучшую пригодность, чем оставшиеся члены популяции. При удалении от фронта Парето, пригодность индивидов ухудшается.

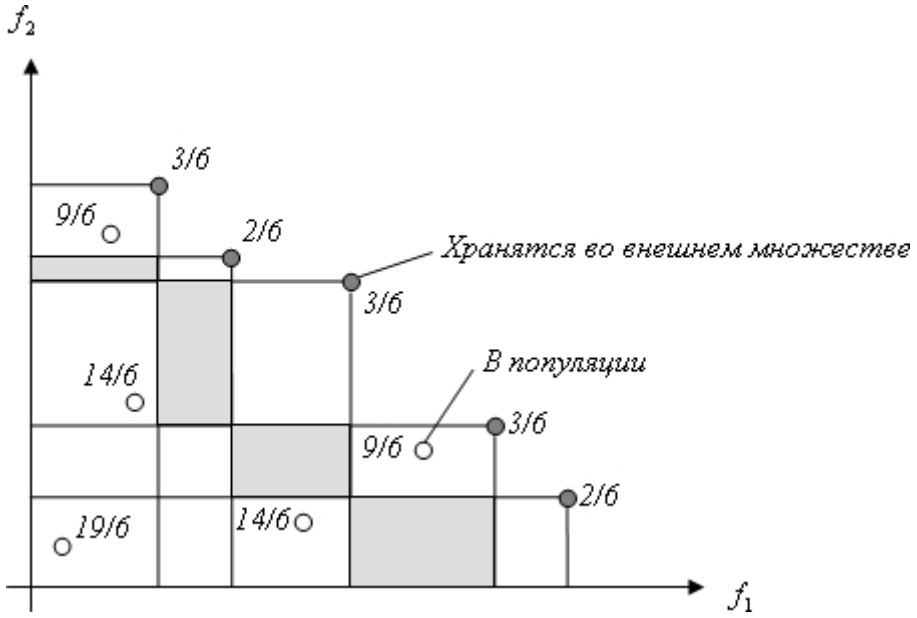


Рисунок 5 – Механизм селекции в методе SPEA

Механизм кластеризации в методе SPEA:

Вход:

$\overline{P'}$ (внешнее множество),

$\overline{N}$ (размер внешнего множества).

Выход: $\overline{P_{t+1}}$ (модернизированное внешнее множество).

1. Инициализировать множество кластеров C. Каждый индивид $i \in \overline{P'}$ образует отдельный кластер: $C = \bigcup_{i \in \overline{P'}} \{\{i\}\}$.

2. Если $|C| \le \overline{N}$, перейти на Шаг 5, иначе перейти на Шаг 3.

3. Вычислить расстояние между всеми возможными парами кластеров. Отдаленность d_c двух кластеров c_1 и $c_2 \in C$ друг от друга определяется как среднее расстояние между парами индивидов, принадлежащих этим кластерам:

$$d_c = \frac{1}{|c_1| \cdot |c_2|} \cdot \sum_{i_1 \in c_1, i_2 \in c_2} d(i_1, i_2),$$

где функция d отражает расстояние (в пространстве целей) между двумя индивидами i_1 и i_2.

4. Определить два кластера c_1 и $c_2 \in C$ с минимальным расстоянием d_c. Эти кластеры объединяются в один больший по размеру кластер:

$$C = C \setminus \{c_1, c_2\} \bigcup \{c_1 \cup c_2\}.$$

Перейти на Шаг 2.

5. Для каждого кластера выбрать репрезентативного индивида, а всех остальных индивидов из него удалить. (Таким образом, репрезентативный индивид – это точка с минимальным средним расстоянием до всех остальных

точек кластера.) Определить уменьшенное недоминируемое множество путем объединения репрезентативных индивидов всех кластеров: $\overline{P}_{t+1} = \bigcup_{c \in C} c$.

При решении некоторых задач, размер множества Парето может быть довольно значительным и даже содержать бесконечное число решений. Однако, с точки зрения ЛПР, нахождение всех недоминируемых решений, когда их количество превышает разумные пределы, является бесполезным. Более того, размер внешнего множества влияет на поведение метода SPEA. С одной стороны, так как все индивиды внешнего множества $\overline{P}$ участвуют в селекции, слишком большое количество хранящихся внешне индивидов может уменьшить селективное давление и замедлить поиск. С другой стороны, усиленный механизм ниширования основывается на образовании так называемой «сетки», определяемой индивидами внешнего множества и, если индивиды из $\overline{P}$ разбросаны неравномерно, представленный выше метод назначения пригодности направит алгоритм в определенные области поискового пространства, что, в свою очередь, приведет к несбалансированности в популяции. Поэтому, порой может быть необходимым и даже обязательным уменьшение размеров внешнего множества, при сохранении его основных характеристик.

Стоит также заметить, что возможность априорного задания количества итоговых точек Парето является очень полезной характеристикой представленного подхода и, в ряде случаев, определяющим фактором, значительно облегчающим для ЛПР окончательный выбор альтернатив.

Отметим также, что вопрос выбора наилучшего метода многокритериальной оптимизации в конкретном случае также пока остается открытым.

ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ РЕШЕНИЯ НЕСТАЦИОНАРНЫХ ЗАДАЧ ОПТИМИЗАЦИИ

Введение

Генетические алгоритмы хорошо зарекомендовали себя на определенном классе задач, решение которых стандартными подходами не принесло существенного повышения эффективности. Большинство методов оптимизации, в том числе и эволюционные алгоритмы, предполагают целевую функцию статической, тогда как большинство процессов реального мира носят динамический характер. Задача оптимизации этих процессов ввиду их распространенности является актуальной на сегодняшний день. Особенностью алгоритмов, предназначенных для работы в условиях нестационарности, является наличие механизмов адаптации к происходящим изменениям. Эволюционные алгоритмы в этом смысле являются благодатной почвой для работы в данном направлении, так как и сама природа не отличается постоянством, и умение адаптироваться к изменениям окружающей среды, особенно в наше время, является определяющим для выживаемости живых существ.

Причина нестационарности кроется в специфике окружающего нас мира, и с повышением уровня развития науки и техники, с усложнением функциональной, структурной и других составляющих технологических систем и взаимосвязей между ними, все чаще в инженерной практике встречаются задачи, в которых приходится учитывать эту особенность. В процессе работы целевая функция претерпевает моделируемый дрейф параметров, который задает характер смещения глобального оптимума. При этом глобальный оптимум может при наступлении изменения может как сместиться в случайном направлении на некоторое расстояние, так и "перепрыгнуть" в другой оптимум, который до того был локальным. Таким образом, в зависимости от того, какие именно свойства поискового алгоритма нужно исследовать, можно получить как простую, так и сложную задачу.

Адаптация генетического алгоритма в условиях нестационарности целевой функции

В общем случае постановка задачи оптимизации нестационарной функции формализуется следующим образом: задана функция $F(\bar{x}(t),\bar{\alpha}(t),t)$, где: $\bar{x}(t)$ – вектор состояния, определенный на интервале $t \in [t_0, t_N]$;

$\bar{\alpha}(t)$ – вектор параметров модели, то есть параметрическая характеристика функции F;

t – время.

Необходимо найти такую функцию $x(t)$, которая в каждый момент времени $t \in [t_0, t_N]$ приносит функции $F(\bar{x}(t),\bar{\alpha}(t),t)$ оптимальное значение:

$$F(\bar{x}(t),\bar{\alpha}(t),t) \rightarrow \min_{x(t)\in X} \quad (1)$$

Под эту формулировку попадают практически все задачи, связанные с

оптимизацией функции, значения параметрических характеристик $\overline{\alpha}(t)$ которой заданы как функции времени. Задачи оптимизации, попадающие под введенное выше определение, в абсолютном большинстве представляют собой задачи регулирования состояния динамического объекта.

Использование генетических алгоритмов в подобных задачах кажется вполне обоснованным: основанные на моделировании процесса развития популяции индивидов по аналогии с тем, как это происходит в природе, генетические алгоритмы принципиально приспособлены к работе в условиях постоянно меняющейся внешней среды. Однако при более глубоком рассмотрении этой проблемы можно заметить ряд особенностей, которые существенно облегчают задачу приспосабливаемости (адаптации) к возникающим изменениям, так как, если не учитывать влияния факторов антропогенного характера, то нетрудно заметить, что большинство изменений условий окружающей среды носит либо постоянный характер (например, сила притяжения Земли), либо циклический характер (приливы и отливы, суточное колебание параметров температуры, освещенности и т.д.).

Без применения специальных методов, при недостаточном объеме выделяемых вычислительных ресурсов наблюдается быстрая потеря разнообразия в популяции и, как следствие, преждевременная его сходимость. Одно из основных преимуществ генетического алгоритма – в любой момент времени наличие множества альтернативных решений – сводится, таким образом, к нулю.

Указанная проблема приобретает особую значимость, когда от алгоритма в течение длительного времени ожидается высокая точность отслеживания оптимума, в то время как последний претерпевает дрейф в пространстве, то есть в задачах динамической оптимизации. Поисковый алгоритм должен не просто в некоторой области глобального оптимума иметь допустимый вариант решения, но и иметь некоторое множество решений, равномерно распределенных в пространстве поиска для того, чтобы в случае значительного изменения быть способным быстро локализовать его новое положение и получить новое допустимое решение.

Таким образом, на первое место “выходит” проблема поддержания разнообразия в популяции. Для решения этой задачи исследователи применяют различные подходы. Согласно широко распространенной классификации применяемые решения данной проблемы можно отнести к следующим классам:

- механизмы адаптации;
- механизмы памяти;
- механизмы поддержания разнообразия.

Рассмотрим каждый класс более подробно:

1. Сущность механизмов адаптации состоит в динамическом изменении параметров генетического алгоритма на основе эвристических или статистических правил во время процесса поиска.

2. Идея использования механизмов памяти не нова. Одним из основных свойств адаптивных подходов является возможность вывода результатов приобретенного ранее опыта. Подходы к реализации памяти разделяются на 2 основных класса:

- неявная память,
- явная память.

К первому классу отнесены методы, основанные на концепции избыточности генетического материала. Идея заимствована из природы: парность хромосом – диплоидность. Хромосома предполагается состоящей из двух наборов генов, каждый из которых однозначно кодирует решение. При этом один из наборов генов является доминантным и участвует полностью во всех шагах генетического алгоритма, а второй

набор – рецессивным, который меняется местами с доминантным только если произошло изменение окружающей среды (целевой функции). В дальнейшем алгоритм работает уже с новым набором генов, который теперь считается доминантным, при этом старый набор не откидывается, а сохраняется как набор рецессивных генов. Преимущество данного шага становится очевидным в случае, когда целевая функция при последующем изменении возвращается в своё первоначальное состояние.

Концепция использования явной памяти состоит в том, что алгоритм целенаправленно отбирает и использует решения, получаемые популяцией в процессе поиска. Согласно обзору работ, в большинстве случаев реализации явной памяти алгоритм в том или ином виде оснащается буфером памяти, куда, согласно определенной исследователем стратегии, в заданные моменты времени, сохраняются хромосомы индивидов. Стратегия определяет, при выполнении каких условий и хромосомы каких индивидов должны быть сохранены в памяти, каким образом в процессе поиска будет использована эта информация, и какие хромосомы должны быть удалены из памяти для того, чтобы освободить место для вновь добавляемых.

Принцип работы механизма достаточно прост: при изменении окружающей среды в память заносится наилучший, найденный на прошедших поколениях, индивид. Затем, после переоценки индивидов рабочей популяции с учетом новых условий окружающей среды, а также оценки всех индивидов, хранящихся в памяти, в популяцию из буфера памяти восстанавливается индивид с наивысшей пригодностью к новым условиям, замещая при этом индивида популяции с наихудшей пригодностью.

Так как размер буфера памяти задается исследователем, то при работе алгоритма возможны две ситуации:

a) размер буфера памяти превышает количество изменений целевой функции, возможное при данном размере вычислительного ресурса. В таком случае после изменения целевой функции лучший найденный индивид просто сохраняется в буфер.

b) размер буфера памяти меньше, чем количество изменений целевой функции. В таком случае если к моменту сохранения индивида буфер памяти был заполнен, то добавляемый индивид заменял наиболее старого. После переоценки индивидов популяции и буфера памяти копия наилучшего индивида из буфера памяти замещала индивида популяции с наихудшей пригодностью.

3. Механизмы поддержания разнообразия основаны на использовании некоторой обоснованной стратегии замещения индивидов рабочей популяции случайно сгенерированными, или же полученными определенным образом во время работы алгоритма. Сюда относятся как подходы, основанные на стохастических стратегиях генерации и замещения индивидов, так и эвристические решения, такие как, например, использование субпопуляций, отслеживающих локальные экстремумы в процессе дрейфа их параметров, попытки использовать прогнозирование будущего расположения глобального оптимума и др.

Рассмотрим так называемую гипермутационную схему (как одну из наиболее простых для реализации). Сущность работы схемы состоит в том, что каждые n поколений некоторое число индивидов замещается случайно сгенерированными. Продолжительность интервала между включениями схемы и число генерируемых (и замещаемых) индивидов задаются исследователем. Таким образом, после наступления изменения и переоценки значений пригодности индивидов рабочей популяции заданное число индивидов, выбранных случайным образом, замещается случайно сгенерированными. При этом учитывается свойство элитизма одного индивида, то есть наилучший индивид популяции не замещается.

Исследование эффективности генетических алгоритмов решения нестационарных задач оптимизации на множестве тестовых функций

Прежде всего, следует отметить, что очень трудно найти стандартные тестовые функции, так как практически в каждой работе предлагается свой вариант тестовой задачи, продолжительность и характер периода стационарности, а также характер дрейфа параметров целевой функции, размерность пространства поиска, конфигурация генетического алгоритма, средствами которого задача решалась. Однако, в результате анализа работ, посвященных исследованиям в данной области, было определено некоторое множество тестовых функций, практически специально разработанных для эволюционных алгоритмов нестационарной оптимизации. Ниже приведен перечень тестовых функций, используемых в работе:

1. Наиболее часто используемой тестовой задачей является семейство динамических функций Осмера. Функция в общем виде задается в следующем виде: $q_1(x,t) = 1 - e^{-200\times(x-c(t))}$, где функция *c(t)* задана как *c(t) = 0.04([t/20])*, при $x \in [1.000; 2.000]$

При этом задача состоит в том, чтобы найти значение *x*, которое минимизировало бы функцию $q_1(x,t)$ в момент времени *t*.

2. В результате исследований адаптивных свойств генетического алгоритма, оснащенного специальными ускоряющими процесс адаптации механизмами, Трояновский и Михалевич выбрали следующую функцию:

$G(\vec{x}) = f_k(\vec{x})$, если $\vec{x} \in D_k$

Данная функция полимодальная, с настраиваемым числом локальных минимумов. Основная идея при генерации целевой функции такова: задан гиперкуб в пространстве поиска, который определен как разрешенная область поискового пространства:

$D = \prod_{i=1}^{n}[0;1)$, где *n* – размерность пространства. По каждой из координат разрешенная область разбивается на *w* равных интервалов, таким образом, что вся разрешенная область разбивается на w^n гиперкубиков, в каждом из которых определена унимодальная функция:

$f_{\bar{v}}(\bar{x}) = a_{\bar{v}} \prod_{i=1}^{n}(r_i - x_i)(x_i - q_i)$, где

q_i и r_i – соответственно нижняя и верхняя границы текущего гиперкубика по *i*-ой компоненте поискового пространства; $a_{\bar{v}}$ - коэффициент высоты экстремума в центре текущего гиперкубика.

Таким образом, реализация тестовой задачи позволяет контролировать не только число локальных экстремумов и положение глобального экстремума, но и моделировать требуемый характер изменений целевой функции.

Ниже приведена таблица (табл. 1) настроек параметров описанной выше тестовой функции Трояновского-Михалевича, используемых при тестировании алгоритмов.

Таблица 1

Настройки параметров тестовой функции Трояновского-Михалевича

Номер изменения функции	Нижняя граница гиперкуба	Верхняя граница гиперкуба	Высота экстремума	Значение F^*
1	-10	-2	0.001	-1.571
2	5	10	0.001	0

3	0.1	0.3	0.1	0
4	0	44	0.0001	-4.777
5	-5	-1	0.0001	-0.024

Замечание: в процессе работы алгоритма варианты изменения целевой функции не обязательно должны следовать друг за другом как в табл. 1. Чередование может происходит произвольным образом в любой последовательности. Естественно, что при чередовании вариантов изменений целевой функции общее количество изменений целевой функции оказывается гораздо больше пяти.

Для сравнения алгоритмов используются следующие показатели: надежность – процент успешных запусков алгоритма (глобальный оптимум найден) от общего числа запусков алгоритма; среднее число итераций до нахождения решения (скорость) – номер итерации, на которой найден глобальный оптимум, усредненный по числу успешных запусков алгоритма. У наилучшего ГА должна быть наибольшая надежность, при равных показателях надежности – наибольшая скорость (наименьшее количество вычислений до обнаружения экстремума).

Пример результатов тестирования приведены в таблице 2.

Таблица 2

Результаты экспериментального исследования алгоритмов на множестве тестовых функций

Алгоритмы		Параметры оценки	Функция Осмера n=1	Функция Трояновского-Михалевича		
				n=2	n=6	n=18
Глобальный характер	1	надежность	100%	80%	55%	30%
		скорость	12	15	15	20
	2	надежность	80%	100%	40%	40%
		скорость	11	16	20	20
	3	надежность	95%	85%	35%	20%
		скорость	9	18	20	20
Локальный характер	4	надежность	65%	65%	20%	20%
		скорость	16	18	20	20
	5	надежность	70%	75%	35%	30%
		скорость	15	19	20	19
	6	надежность	95%	45%	30%	5%
		скорость	18	20	20	20
Неизвестный характер	7	надежность	100%	90%	35%	15%
		скорость	10	15	18	20
	8	надежность	70%	65%	55%	10%
		скорость	15	15	17	20
	9	надежность	60%	55%	35%	30%
		скорость	17	14	18	20

где:

- алгоритмы 1, 4, 7 – оснащены механизмом диплоидности хромосом,
- алгоритмы 2, 5, 8 – оснащены буфером памяти,
- алгоритмы 3, 6, 9 – включают в себя гипермутационную схему.

Использовались следующие настройки алгоритмов: количество изменений целевой функции – 10, размер популяции – 100, максимальное число итераций – 200, количество поколений до наступления изменения целевой функции – 20, прогонов – 20.

Выводы

Несколько модифицировав классификацию подходов к улучшению адаптивных свойств генетического алгоритма методики можно разбить на два больших класса.

Первый класс представляют собой способы, действие которых направлено на повышение адаптивных свойств ГА в общем. Это такие механизмы, задача которых состоит в том, чтобы стимулировать глобальный характер поиска. Подходы, относящиеся к данному классу, характеризует независимость от свойств оптимизируемой функции. К этому классу относятся механизмы поддержания и повышения разнообразия в популяции. Ориентированные на изучение характера поверхности целевой функции, эти подходы способны повысить эффективность работы генетического алгоритма не только на нестационарном классе задач, но и на других классах, таких, например, как зашумленные функции.

К другому классу относятся подходы, которые основаны на адаптации к известным (априори, или эмпирически) свойствам целевой функции. Основываясь на этом знании не составляет труда выдвинуть идею уменьшения влияния данного фактора. Примером механизмов адаптации данного класса может служить механизм памяти. Иначе, какой смысл в оснащении генетического алгоритма дополнительными структурами, требующими привлечения ресурсов, если нет обоснования использования этих структур? Так, например, если характер дрейфа параметров нестационарной функции позволяет сделать вывод о том, что вероятность посещения объектом регулирования состояния, которое он уже принимал ранее, ниже, скажем, 50%, то в дело вступает экономический аспект проблемы. Другой пример — оснащение поискового алгоритма средствами фильтрации высокочастотного шума имеет смысл при наличии таковых. Если использование средств поддержания в популяции необходимого уровня разнообразия может быть обоснованным при неопределенности характера поведения целевой функции, и даже более того, является необходимостью для эффективной работы генетического алгоритма, то оснащение последнего средствами, например, обработки сложных нелинейных ограничений на разрешенную область поискового пространства без обоснования наличия последних едва ли будет экономически выгодно (с учетом затрат времени на их разработку и реализацию).

Гибридные эволюционные алгоритмы

Несмотря на свою универсальность, генетический алгоритм (ГА) имеет два существенных недостатка.

Первым недостатком генетического алгоритма является то, что нет гарантии того, что найденное ГА решение является хотя бы локально-оптимальным, несмотря на глобальную сходимость алгоритма.

Ко второму недостатку можно отнести то, что для эффективной работы генетического алгоритма необходимо тонко настроить его параметры. И этот вопрос является актуальным не только для обычного пользователя или инженера-конструктора, который не владеет эволюционными алгоритмами, но он также не тривиален и для специалиста в области эволюционного поиска. То есть даже специалист в области эволюционного поиска не может дать точного ответа на вопрос, какие настройки генетического алгоритма будут оптимальными для решения конкретной задачи оптимизации.

Для устранения первого недостатка предлагаются различные способы: коэволюционные алгоритмы, гибридные алгоритмы и др. Например, можно предложить классическую схему гибридизации генетического алгоритма с алгоритмом локального поиска, в которой генетический алгоритм являлся бы способом генерации множества точек мультистарта, из которых затем производился локальный спуск. То есть локальный поиск выступает в роли процедуры уточнения найденных решений ГА. Данная схема подтвердила эффективность гибридизации ГА с локальным спуском. Существует другой способ гибридизации ГА и локального спуска - метод Хука-Дживса, Нелдера Мида, но с алгоритмами, предназначенными только для работы с непрерывными функциями.

Для устранения первого недостатка можно также использовать гибридный генетический алгоритм с локальным поиском. Здесь подразумевается не классический вид гибридизации, когда локальный поиск на заключительном этапе всего процесса оптимизации уточняет решение найденное ГА, а другой вид гибридизации. Предлагаемый вид гибридизации представляет собой комбинацию работы локального поиска и генетического алгоритма в ходе всего процесса оптимизации. То есть локальный поиск используется как один из генетических операторов, имитирующий прижизненную адаптацию индивидов генетического алгоритма, тем самым, ускоряя процесс оптимизации, и одновременно способствует локальному уточнению решений, найденных генетическим алгоритмом.

В области эволюционного поиска для гибридных ГА с локальным поиском в ходе процесса оптимизации существует две концепции эволюции.

Один из видов гибридизации ГА заключается в следующем: на каждом поколении ГА отбираются несколько (перспективных) индивидов для того, чтобы из них были выполнены несколько шагов локального поиска (ЛП)

(моделирование прижизненной адаптации индивидов). В зависимости от типа моделируемой эволюции, индивид после локального спуска в следующую популяцию переходит: либо таким же, каким он был перед ЛП, но с «новой» пригодностью (прижизненная адаптация делает индивида более пригодным, но по наследству не передается – эволюция по Дарвину), или таким, каким он стал после ЛП (прижизненная адаптация делает индивида более пригодным и передается по наследству – эволюция по Ламарку).

Таким образом, для устранения первого недостатка используется специфическая схема гибридизации ГА и локального поиска названная эволюцией по Ламарку. Данная схема гибридизации позволяет увеличить вероятность нахождения оптимального оптимума для любой комбинации параметров ГА.

Представим пошаговые процедуры для стандартного и гибридного генетических алгоритмов.

Обобщенная пошаговая структура стандартного ГА безусловной оптимизации

1. Сгенерировать случайным образом начальную популяцию.
2. Оценить полученную популяцию.
3. Сгенерировать популяцию потомков.
   - Селекция (выбор двух индивидов из текущей популяции).
   - Рекомбинация (скрещивание выбранных индивидов).
   - Мутация (генетическое изменение полученного потомка).
4. Если не все поколения пройдены, то перейти на шаг 2, иначе выдать наилучшего найденного индивида и его значение целевой функции в качестве решения оптимизации.

Обобщенная пошаговая структура гибридного ГА безусловной оптимизации

1. Сгенерировать случайным образом начальную популяцию.
2. Оценить полученную популяцию.
3. Сгенерировать популяцию потомков.
   - Селекция (выбор двух индивидов из текущей популяции).
   - Рекомбинация (скрещивание выбранных индивидов).
   - Мутация (генетическое изменение полученного потомка).
4. Выполнить локальный поиск из нескольких индивидов текущего поколения в соответствии с концепцией эволюции по Дарвину или Ламарку.
5. Если не все поколения пройдены, то перейти на шаг 2, иначе выдать наилучшего найденного индивида и его значение целевой функции в качестве решения оптимизации.

В силу того, что в реальных практических задачах необходимо учитывать ограничения, накладываемые на искомые переменные, то

целесообразным является применение специальных методов учета ограничений в предлагаемых гибридном и стандартном генетических алгоритмах. Кроме рассмотренных выше методов может быть применен еще один подход.

Лечение локальным поиском

В данном методе все индивиды, нарушившие хотя бы одно из ограничений на текущем поколении «вылечиваются» и становятся допустимыми. Метод работает по следующей схеме:

1. Выбирается допустимый индивид (индивид-донор) из текущей популяции (если такого индивида нет в популяции, то он генерируется – создается).
2. Вычисляется разница в битах между текущим, нарушившим ограничения индивидом и индивидом, выбранным на шаге 1.
3. Далее бит за битом сокращается разница, путем изменения битов в «лечащемся» индивиде на соответствующие биты из индивида-донора, до тех пор, пока «лечащийся» индивид не перейдет в допустимую область.

Используя способы учета ограничений при решении условной задачи оптимизации с помощью стандартного и гибридного ГА, пошаговые структуры алгоритмов будут выглядеть следующим образом.

Обобщенная пошаговая структура стандартного ГА условной оптимизации

1. Сгенерировать случайным образом начальную популяцию.
2. Если установлено лечение локальным поиском – выполнить лечение.
3. Оценить полученную популяцию (пригодность индивидов зависит от способа обработки ограничений).
4. Сгенерировать популяцию потомков.
   - Селекция (выбор двух индивидов из текущей популяции).
   - Рекомбинация (скрещивание выбранных индивидов).
   - Мутация (генетическое изменение полученного потомка).
5. Выполнить лечение недопустимых индивидов, если это установлено пользователем.
6. Если не все поколения пройдены, то перейти на шаг 2, иначе выдать наилучшего найденного индивида и его значение целевой функции в качестве решения оптимизации.

Обобщенная пошаговая структура гибридного ГА условной оптимизации

1. Сгенерировать случайным образом начальную популяцию.
2. Оценить полученную популяцию (с использованием штрафных

функций).

3. Генерировать популяцию потомков.
   - Селекция (выбор двух индивидов из текущей популяции).
   - Рекомбинация (скрещивание выбранных индивидов).
   - Мутация (генетическое изменение полученного потомка).
   - Выполнить локальный поиск из нескольких индивидов текущего поколения в соответствии с концепцией эволюции по Дарвину или Ламарку.
4. Выполнить лечение недопустимых индивидов, если это установлено пользователем.
5. Если не все поколения пройдены, то перейти на шаг 2, иначе выдать наилучшего найденного индивида и его значение целевой функции в качестве решения оптимизации.

Пример тестирования гибридных эволюционных алгоритмов на практических задачах

В качестве примера практического применения и тестирования гибридных ГА рассмотрим задачи выбора эффективного варианта распределенной системы управления сложным объектом. Речь идет о выборе эффективных вариантов технологического, командно-программного и целевого контура системы управления косическим аппаратом и системы управления дорожным движением по сети автомобильных дорог. Не вдаваясь в подробности скажем, что модели оценки эффективности этих систем являются Марковскими процессами, интенсивности переходов между состояниями которых и надо оптимизировать выбором варианта аппаратно-программных комплексов, реализующих алгоритм управления. Целевые функции и ограничения в этих задачах заданы алгоритмически – для их вычисления необходимо многократно решать систему уравнений Колмогорова-Чэпмена с большим числом уравнений. Необходимо сказать, что выбор генетического алгоритма как основной оптимизационной процедуры предопределил исследование свойств целевых функций с помощью алгоритма прямого локального поиска. Результаты исследования полезных для оптимизации свойств целевых функций для различных задач, представлены в таблице 1.

Таблица 1

Результаты выполнения целочисленной оптимизации для определения полезных свойств целевых функций

Показатель эффективности работы РСОИиУ	Свойства целевых функций
Технологический контур КА	Унимодальность и монотонность коэффициента готовности КА. Полимодальность и немонотонность

	коэффициентов готовностей ЦА и БКУ
Командно-программный контур КА	Многоэкстремальность с несколькими множествами постоянства для показателя, характеризующего время автономной работы КА
Целевой контур КА	Унимодальность и монотонность показателя, характеризующего среднего времени на реакцию поступившей заявки
Управление дорожным трафиком	Полимодальность и немонотонность коэффициентов готовностей ЦУ и ЛКУ

Из результатов проведенных исследований видно, что большинство целевых функций являются многоэкстремальными и немонотонными. Этот факт предопределил выбор в пользу эволюционного алгоритма.

Разработанный гибридный генетический алгоритм тестировался и сравнивался со стандартным генетическим алгоритмом. Для этого необходимо установить оптимальные параметры структуры алгоритма.

Структурой ГА будем считать набор параметров алгоритма: вид селекции, мутации, рекомбинации. Качество работы ГА будет оцениваться по трем критериям в порядке их важности: надежность, скорость, разброс. Надежность это отношение эффективных запусков (на которых было найдено решение) к общему числу запусков алгоритма с фиксированными параметрами. Скорость – это среднеарифметическое значение (по нескольким прогонам алгоритма с фиксированными параметрами) поколения на котором найдено решение впервые. Разброс - интервал поколений, левая граница которого есть наименьший, а правая наибольший номер поколения, на которых найдено решение.

Постоянные параметры стандартного генетического алгоритма: численность популяции равна 25, число поколений равно 30. То есть число вычислений целевой функции равно 750, что составило порядка 0.3% от всей области определения функции (т.к. аргумент функции был представлен в виде булевого вектора (хромосомы) длиной 18 бит.) Размер в элитарной и турнирной группах был равен 5.

Постоянные параметры гибридного генетического алгоритма: численность популяции равна 24, число поколений равно 25. Настройки локального поиска: 10% от популяции, 3 шага для каждого выбранного индивида для локального поиска. То есть число вычислений целевой функции также равно 750, что составило порядка 0.3% от всей области определения функции (т.к. аргумент функции был представлен в виде булевого вектора (хромосомы) длиной 18 бит.) Размер в элитарной и турнирной группах был равен 5.

В приведенных ниже таблицах выделены параметры ГА, которые оказались оптимальными с точки зрения тех критериев, о которых уже было упомянуто выше: надежность, скорость, разброс. В таблицах 2 и 3 представлены сводные результаты тестирования алгоритмов для задач оптимизации показателей эффективности функционирования технологического контура, соответственно по надежности и скорости.

Таблица 2

Надежность стандартного и гибридных схем ГА для технологического контура КА

	Коэффициент готовности КА			Коэффициент готовности ЦА			Коэффициент готовности БКУ		
Вид ГА	Мин.	Макс.	Сред.	Мин.	Макс.	Сред.	Мин.	Макс.	Сред.
Стандартный	0	1	0,28	0,03	1	0,52	0,98	1	0,99
Эволюция по Дарвину	0,01	0,39	0,17	0,05	0,63	0,35	0,62	1	0,90
Эволюция по Ламарку	**0,87**	**1**	**0,97**	**0,97**	**1**	**0,998**	**1**	**1**	**1**

Таблица 3

Сравнение по показателю скорости стандартного и гибридных схем ГА для технологического контура КА (где скорость - номер поколения, на котором впервые найдено оптимальное решение)

	Коэффициент готовности КА			Коэффициент готовности ЦА			Коэффициент готовности БКУ		
Вид ГА	Мин.	Макс.	Сред.	Мин.	Макс.	Сред.	Мин.	Макс.	Сред.
Стандартный	-	25	18	7	22	16	4	11	7
Эволюция по Дарвину	7	32	20	11	22	16	4	12	9
Эволюция по Ламарку	**4**	**10**	**6**	**3**	**9**	**5**	**2**	**3**	**2**

Оптимальной структурой ГА в данной задаче является: турнирная селекция, слабая (средняя) мутация, равномерная рекомбинация. Оптимальной структурой гибридного ГА при использовании концепции ЛП по Ламарку является: турнирная селекция, слабая или средняя мутация, рекомбинация может быть различной.

Результаты выбора оптимального метода ограничений

В силу того, что более эффективным алгоритмом оптимизации оказался генетический алгоритм, который совместно с локальным поиском, имитирует эволюцию по Ламарку, именно он тестировался и для условной задачи оптимизации.

Число индивидов в популяции равно 25, а число поколений равно 35. Величины ограничений на объем оперативной памяти M0 = 20, и на стоимость C0 = 0,07. Число допустимых точек в пространстве в количестве

4729 точек, что составило около 2% от всего поискового пространства. Число запусков программы для вычисления статистических характеристик равно 100.

Использование штрафных функций:

Метод динамических штрафов: основным параметром является коэффициент, относительно которого увеличивается степень штрафа накладываемого на недопустимое решение с течением времени (поколений). Величина коэффициента изменялась в интервале [0.005; 0.015] с шагом 0,001. Результаты приведены в таблице 4.

Таблица 4

Настройка параметров динамической штрафной функции

Величина коэффициента	Надежность	Скорость	Разброс
0.005	0.68	21	[9;35]
0.006	0.78	21	[7;33]
0.007	0.72	19	[5;35]
0.008	0.67	20	[9;34]
0.009	**0.85**	**21**	**[8;35]**
0.010	0,76	22	[10;35]
0.011	0,66	21	[9;35]
0.012	0,65	21	[9;35]
0.013	0,64	24	[8;35]
0.014	0,77	24	[6;34]
0.015	0,58	21	[9;34]

Метод адаптивных штрафов. Настраиваемыми параметрами являются коэффициент уменьшения и соответственно увеличения штрафа за то, что наилучший индивид в популяции представляет собой допустимое решение на протяжении ряда поколений и соответственно недопустимое. Основным параметром настройки является количество поколений, в течении которых индивид принадлежит к области допустимых решений или недопустимых. Величина поколений изменяется в интервале, а коэффициенты увеличения и уменьшения были неизменны и равны 2, 4, соответственно. Результаты настройки приведены в таблице 5.

Таблица 5

Настройка параметров адаптивной штрафной функции

Число поколений	Надежность	Скорость	Разброс
2	0.63	22	[3;35]
3	0.69	20	[4;35]
4	0.64	22	[9;35]
5	0.72	24	[8;35]
6	**0.79**	**22**	**[6;33]**
7	0.66	23	[8;34]
8	0.76	21	[3;32]

Метод статических штрафов. В данной штрафной функции параметры настройки следующие: величина штрафной области (поля), в которой значение величины штрафа неизменно; коэффициент увеличения штрафа; минимальная величина штрафа.

В данной работе представлены настройки величины штрафной области в интервале [0.01; 0.1] с шагом 0.01, при неизменном коэффициенте увеличения штрафа, который равен 2 и минимальной величине штрафа равной 0.01.Результаты настройки представлены в таблице 6.

Таблица 6

Настройка параметров статической штрафной функции

Величина штрафной области	Надежность	Скорость	Разброс
0.01	0.59	23	[10;35]
0.02	0.4	19	[7;33]
0.03	0.42	22	[9;35]
0.04	0.56	22	[10;34]
0.05	0.51	22	[5;35]
0.06	**0.7**	**22**	**[8;32]**
0.07	0.57	19	[7;35]
0.08	0.66	23	[5;35]
0.09	0.68	19	[8;35]
0.1	0.64	22	[8;34]

Лечение локальным поиском. Использование локального поиска для лечения недопустимых решений. В данной работе представлены два способа использования локального поиска.

Индивид, относительно которого выполняется лечение, выбирается случайно или берется наилучший индивид. Результаты применения локального поиска представлены в таблице 7.

Таблица 7

Использование локального поиска для лечения недопустимых индивидов

Способ выбора допустимого индивида	Надежность	Скорость	Разброс
Случайный	**0.78**	**14**	**[4;34]**
Произвольный	0.56	18	[2;35]

Метод барьерной штрафной функции с локальным поиском. Настройки барьерной функции были неизменны. Коэффициент понижения величины барьера равен 0,1. Результаты использования барьерной функции представлены в таблице 8.

Таблица 8

Использование барьерной функции

Способ выбора допустимого индивида в локальном поиске	Надежность	Скорость	Разброс
Случайный	**0.73**	**16**	**[5;34]**
Произвольный	0.52	16	[3;34]

Для наглядности результатов тестирования способов учета ограничений в ГА на эффективность при оптимизации показателей эффективности функционирования КА приведем сводную таблицу 9., в которой выделены наилучшие варианты среди каждого способа учета ограничений.

Таблица 9

Надежность гибридного ГА в условной задаче оптимизации коэффициента готовности КА

	Коэффициент готовности КА		
Способ учета ограничений	Минимальная Надежность	Максимальная Надежность	Средняя надежность
Барьерная функция + «Лечение»	0,52	0,73	0,63
«Лечение» локальным поиском	0,56	0,78	0,67
Статическая штрафная функция	0,40	0,70	0,57
Адаптивная штрафная функция	**0,63**	**0,79**	**0,70**
Динамическая штрафная функция	0,58	0,85	0,71

Известно, что на практике приходится решать задачи условной оптимизации. В связи с этим, было необходимо протестировать гибридные генетические алгоритмы, имитирующие эволюции по Дарвину и Ламарку и сравнить с стандартным генетическим алгоритмом по эффективности при решении задач условной оптимизации для задач поставленных в первой главе.

Алгоритмы в силу своей стохастической природы сравнивались по трем показателям качества работы: надежность, скорость и интервал на котором был найден объективный оптимум. Данные показатели были усреднены по 100 прогонам каждой комбинации алгоритма. Сравнительный анализ стандартного и гибридного генетических алгоритмов представлен в следующих таблицах 10, 11 и 12.

Из результатов приведенных в таблицах, видно, что большее значение средней надежности по всем структурам у гибридного ГА (эволюция по Ламарку), при этом с меньшей дисперсией. Таким образом, гибридный ГА (эволюция по Ламарку) подтверждает свою эффективность и при решении данной задачи.

Таблица 10

Сравнение показателей эффективности ГА и гибридного алгоритма при оптимизации показателя, характеризующего коэффициент готовности КА

	Надежность лучшей структуры ГА	Скорость	Разброс	Оценка мат. ожидания надежности по всем алгоритмам	Оценка дисперсии надежности по всем структурам ГА
Гибридный ГА (эволюция по Ламарку)	0.80	22	[6;40]	0,55	0,0250
Гибридный ГА (эволюция по Дарвину)	0,70	25	[8; 42]	0,33	0,0234
Стандартный ГА	0.72	20	[3;39]	0,36	0,0334

Таблица 11

Сравнение показателей эффективности ГА и гибридного алгоритма при оптимизации показателя автономной работы КА

	Надежность лучшей структуры ГА	Скорость	Разброс	Оценка мат. ожидания надежности по всем алгоритмам	Оценка дисперсии надежности по всем структурам ГА
Гибридный ГА (эволюция по Ламарку)	1	3	[1; 8]	**0,88**	**0,0075**
Гибридный ГА (эволюция по Дарвину)	0,94	4	[1; 9]	**0,77**	**0,0118**
Стандартный ГА	1	4	[1;10]	**0,83**	**0,0128**

Таблица 12

Сравнение показателей эффективности ГА и гибридного алгоритма при оптимизации показателя, характеризующего среднее время реакции системы управления КА

	Надежность лучшей структуры ГА	Скорость	Разброс	Оценка мат. ожидания надежности по всем алгоритмам	Оценка дисперсии надежности по всем структурам ГА
Гибридный ГА (эволюция	0,89	23	[14; 48]	0,83	0,112

по Ламарку)					
Гибридный ГА (эволюция по Дарвину)	0,53	24	[11; 49]	0,45	0,131
Стандартный ГА	0,65	24	[12;55]	0,54	0,182

После анализа результатов, приведенных в таблицах, можно сделать вывод, о том, что гибридный генетический алгоритм, моделирующий эволюцию по Ламарку является эффективным методом оптимизации алгоритмически заданных целевых функций на дискретной решетке.

Лучшая структура стандартного эволюционного алгоритма такова: равномерная или двухточечная рекомбинация, турнирная селекция, слабая мутация. Лучшая структура гибридного эволюционного алгоритма: равномерная рекомбинация, слабая мутация и пропорциональная селекция.

Выводы

Гибридные генетические алгоритмы, разработанные на основе стандартного генетического алгоритма, позволяют устранить необходимость тонкой настройки параметров генетического алгоритма на решаемую задачу. Гибридный генетический алгоритм, имитирующий эволюцию по Ламарку, превосходит по надежности и скорости стандартный генетический алгоритм и гибридный генетический алгоритм (по Дарвину).

Кроме этого, модифицированный гибридный генетический алгоритм, оснащенный способами учета ограничений, является эффективным инструментом для решения условных задач оптимизации.

Все данные выводы имеют смысл для решения той конкрентной задачи, на которой тестировались алгоритмы.

Отметим также, что хотя и получены однозначные выводы об эффективных настройках алгоритмов, все же задача по прежнему решалась очень много раз для получения статистически достоверных выводов. Более практичным подходом была бы разработка алгоритмов, самостоятельно настраивающих свои параметры в ходе однократного решения задачи.

САМОНАСТРАИВАЮЩИЕСЯ КОЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ ОПТИМИЗАЦИИ

Введение

Известно, что возможность многократного запуска эволюционного алгоритма для нахождения более точного решения не всегда существует по многим причинам: высокая стоимость вычислений целевой функции; отсутствие времени на перезапуск алгоритма и т. д. Поэтому необходимо, исходя из какой-либо априорной информации или из накопленного опыта, выбирать конкретный алгоритм и настраивать его параметры под имеющуюся оптимизационную функцию.

При этом подобная задача - выбор конкретного алгоритма и настройка его параметров – сама по себе является очень сложной, при неудачном решении которой алгоритм может не справиться с оптимизацией. Таким образом, возникает необходимость разработки процедур, автоматизирующих выбор и настройку эволюционных алгоритмов. Одним из наиболее перспективных в этом отношении подходов является коэволюционный алгоритм.

Идея алгоритма

Основная идея коэволюционного алгоритма состоит в следующем: одновременно эволюционируют несколько популяций, каждая из которых оптимизирует заданную функцию и обладает своей стратегией оптимизации. При этом популяции «борются» за ресурс, который в течение работы алгоритма перераспределяется в пользу более эффективной из них.

Схематично работу коэволюционного алгоритма можно представить следующим образом (рис .1).

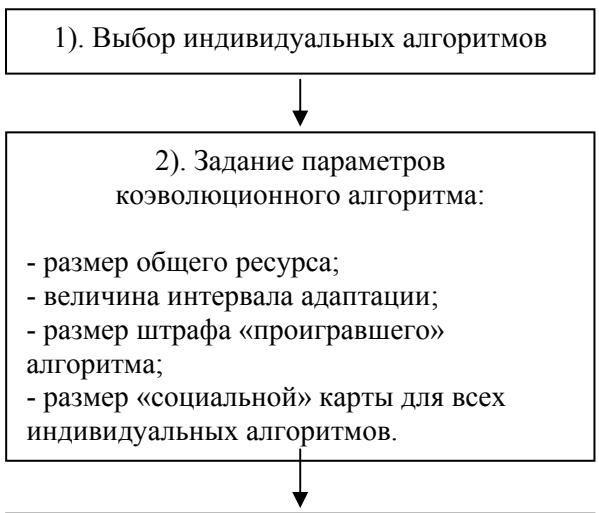
Рассмотрим шаги схемы более подробно:

1. Выбор индивидуальных алгоритмов

На данном этапе необходимо выбрать индивидуальные алгоритмы (из множества имеющихся в распоряжении алгоритмов) для включения их в состав коэволюционного алгоритма.

В качестве индивидуальных алгоритмов обычно берут множество обыкновенных генетических алгоритмов, у которых закреплены одинаковыми все параметры, кроме схем селекции и формирования нового поколения. Все виды применяемых в индивидуальных алгоритмах схем селекции и формирования нового поколения подробно были Вами рассмотрены в лекциях «Стандартный генетический алгоритм» и «Эволюционные стратегии».

Практика применения коэволюционного алгоритма для решения сложных задач оптимизации показывает, что лучший результат показывают комбинации алгоритмов, обладающих тремя типами характеров поиска: глобальным, локальным и неопределенным.



2. Задание параметров алгоритма

а) Задание общего ресурса

Ресурсом является количество вычислений целевой функции – произведение размера популяции на число итераций.

Изначально ресурс предполагается общим и поровну разбивается на каждую популяцию.

б) Задание величины интервала адаптации

В течение определенного количества шагов (так называемого интервала адаптации) каждый алгоритм работает по отдельности. Величина интервала адаптации определяется и выставляется исследователем. Естественно, что его значение не должно быть маленьким (алгоритмы не успеют проявить себя) и большим (будет мало времени на адаптацию).

с) Задание величины штрафа проигравшего алгоритма

Штраф - это некоторый процент от размера популяции индивидуального алгоритма, на который мы будем сокращать размеры популяции проигравших алгоритмов.

Естественно, что его значение также не должно быть слишком маленьким (алгоритмы не почувствуют изменений) и большим (процедура поиска у алгоритма с маленьким размером популяции становится бессмысленной).

д) Задание размера “социальной карточки”

“Социальная карточка” – это некоторый процент от размера популяции индивидуального алгоритма, являющийся минимальным гарантированным размером популяции. То есть, мы можем сокращать проигравшую популяцию только до тех пор, пока она не достигнет минимального гарантированного размера – “социальной карточки” (необходимость данного ограничения доказана практически).

3. Работа алгоритмов в течение интервала адаптации

В течение заданного количества шагов (так называемого интервала адаптации) каждый алгоритм работает по отдельности, самостоятельно выполняя оптимизацию целевой функции.

4. Оценка алгоритмов

Так как коэволюционный алгоритм основывается на конкурирующих стратегиях алгоритмов, то для субпопуляций необходимо ввести функцию пригодности. С помощью этой функции определяется лучшая популяция, и ей дается больше возможностей для воспроизводства.

Пусть T – интервал адаптации, $b_i(k)$=1, если i-я популяция в момент k содержит наилучшего (по всем популяциям) индивида, k=0 – означает текущую ситуацию, k=1 – предыдущую и т. д. Тогда качество i-й популяции можно вычислить по формуле (1):

$$q_i = \sum_{k=0}^{T-1} \frac{T-k}{k+1} \cdot b_i(k) \qquad (1)$$

В дальнейшем, полученные оценки индивидуальных алгоритмов, входящих в состав коэволюции, являются главным критерием оценки работы алгоритмов и используются в процессе перераспределение ресурса.

5. Проверка критерия остановки

В случае выполнения одного из критериев остановки работа алгоритма останавливается и выводится наилучшее найденное решение, иначе, на основе оценок индивидуальных алгоритмов, полученных на шаге 4 происходит перераспределение ресурса (шаг 6).

Как правило, критерии останова могут быть следующими: закончилось количество ресурса, выделенное на вычисление целевой функции; закончилось время, выделяемое на решение задачи; стагнация и т.д.

6. Перераспределение ресурса

Изменение размеров ресурсов происходит путем сокращения каждой проигравшей популяции на некоторый процент (определенный заранее размер штрафа) и увеличением победившей популяции на число, равное сумме потерь проигравших. Таким образом, общее число индивидов остается неизменным. Однако необходимо не забывать, что мы можем сокращать проигравшую популяцию только до тех пор, пока она не достигнет минимального гарантированного размера – “социальной карточки”.

После перераспределения ресурсов алгоритмы продолжают свою работу с популяциями измененных размеров, в которые входят только лучшие индивиды из предыдущих популяций. Далее, повторяем шаги 3 - 6, пока не выполнится критерий останова, либо не кончится ресурс, выделенный на решение задачи.

Показатели эффективности работы коэволюционного алгоритма

В качестве критериев сравнения были взяты такие показатели как: надежность и скорость. Надежность – количество (в процентном соотношении) успешных запусков алгоритма из общего числа запусков. Скорость оценивается номером поколения, в котором алгоритм находит решение с заданной точностью.

Исследование эффективности было проведено на том же множестве тестовых функций, что предлагается для исследования индивидуальных алгоритмов (см. модуль 1 «Стандартные эволюционные алгоритмы»).

Исследования работы коэволюционного алгоритма и алгоритмов по отдельности проводились при различных размерах популяций и количествах поколений.

Здесь приводится пример результатов исследований при размере вычислительных ресурсов равном 5000. Показаны два «средних» варианта комбинации вычислительных затрат и количества поколений: N (размер популяции) =100 и n (количество поколений) = 50, а также N = 50 и n = 100. На рисунках слева, показывающих поведение лучшего

индивида популяции, по оси абсцисс - количество поколений, по оси ординат – значения функции пригодности. Коэволюционный алгоритм показан сплошной жирной линией. На рисунках справа, показывающих происходящее при этом изменение размеров популяций, по оси абсцисс - количество поколений, по оси ординат – размер популяции алгоритма.

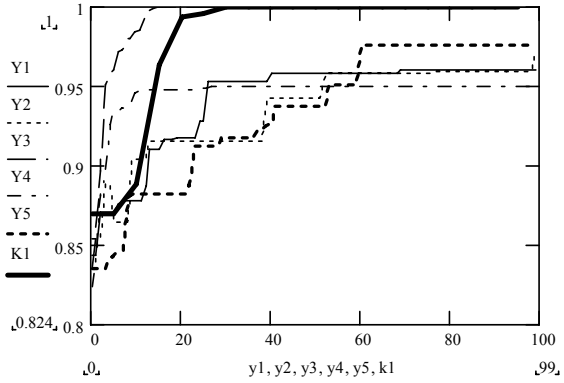


Рис. 2 Поведение лучшего индивида

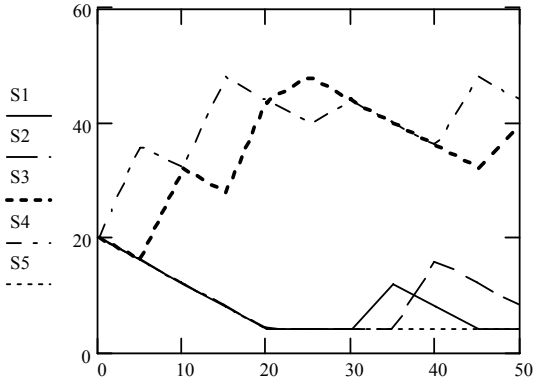


Рис. 3 Процесс перераспределения ресурса

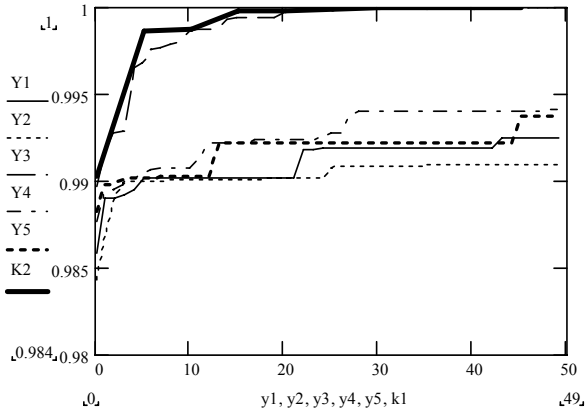


Рис. 4 Поведение лучшего индивида

Рис. 5 Процесс перераспределения ресурса

На всех рисунках показан усредненный результат по 20 прогонам алгоритмов.

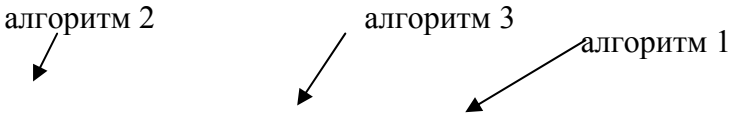
При этом в состав коэволюционного алгоритма индивидуальные алгоритмы включены следующим образом:

- 2 алгоритма с глобальным характером поиска;
- 2 алгоритма с локальным характером поиска;
- 1 алгоритм с неизвестным характером поиска.

Алгоритмы выбраны случайным образом каждый из своей подгруппы, соответствующей характеру поиска.

В результате исследований можно сделать следующий вывод: коэволюционный алгоритм лучше, чем «худший» из индивидуальных алгоритмов, и даже лучше чем «средний» алгоритм, но не всегда лучше чем «лучший» из алгоритмов (в случае сравнения по показателю надежности алгоритмов).

Полученный результат объясняется также тем, что в состав коэволюционного алгоритма в данном исследовании входил алгоритм с локальным характером поиска. Происходящий в ходе поиска процесс перераспределения ресурсов между алгоритмами можно увидеть на рис. 6.



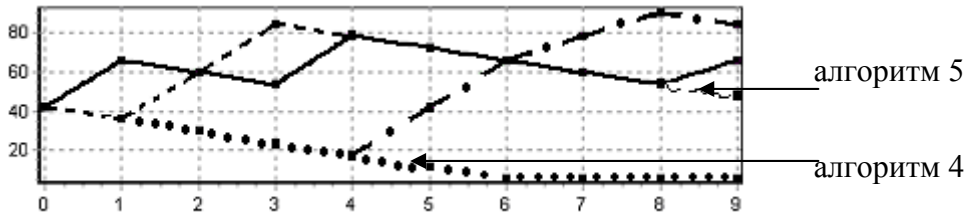


Рис. 6 Перераспределение ресурса в ходе поиска глобального оптимума многоэкстремальной функции «Сомбреро»

Здесь алгоритмы 1 и 2 – с глобальным характером поиска, алгоритмы 3 и 4 – с локальным характером поиска и алгоритм 5 – с неопределенным характером поиска. На рисунке видно, как на первых шагах оптимизации оба алгоритма с локальным характером проигрывают, но затем один из них (алгоритм 3) попадает в хорошую область, найденную с помощью алгоритмов с глобальными свойствами (алгоритмы 1 и 2), которые лидировали на первых шагах. Затем алгоритмом с локальными свойствами осуществляется быстрый спуск к более точному значению глобального оптимума.

Таким образом, подтверждается предположение о том, что нельзя все отнимать у алгоритма (выгодность «социальной карточки»), так как проигрывающие на начальных шагах алгоритмы могут в дальнейшем помочь найти лучшее решение.

Зависимость свойств коэволюционного алгоритма от выбора его параметров

При рассмотрении основных шагов коэволюционного алгоритма видно, что на эффективность коэволюционного алгоритма могут оказать влияние величины его параметров: количество индивидуальных алгоритмов, включенных в коэволюцию; величина интервала адаптации; размер штрафа и размер «социальной карточки». Для проверки данного предположения были проведены следующие исследования работы коэволюционного алгоритма:

1). Исследование эффективности коэволюционного алгоритма в зависимости от величины штрафа и социальной карточки.

Исследование проводилось в случае включения в состав коэволюционного алгоритма трех индивидуальных алгоритмов, показавших наибольшую эффективность на тестовых задачах. Использовались следующие настройки коэволюционного алгоритма: размер популяции – 210, максимальное число вычислений целевой функции – 10000, интервал адаптации – 5, прогонов – 20.

В результате можно сделать вывод:

- размер социальной карты необходимо выбирать в пределах 5–35% от размера популяции индивидуального алгоритма.
- размер штрафа необходимо выбирать в пределах 2–15% от размера популяции индивидуального алгоритма.

Из всего этого можно сделать еще один вывод: не нужно сильно штрафовать проигравшие алгоритмы, но в то же время нельзя оставлять им больших «социальных» гарантий.

Наибольшая эффективность коэволюционного алгоритма была получена при размере штрафа = 15% и размере социальной карты = 15% от размера популяции индивидуального алгоритма. В процессе дальнейшего исследования будут использоваться данные значения параметров.

2). Другим не менее важным параметром является величина интервала адаптации.

При исследовании использовались следующие настройки коэволюционного алгоритма: 1 алгоритм с глобальным характером поиска, 1 алгоритм с локальным

характером поиска, 1 алгоритм с неизвестным характером поиска, общий размер популяции – 300, максимальное число итераций целевой функции при каждом интервале адаптации – 200, размер штрафа – 15%, размер соц. карты – 15%, прогонов – 20.

По результатам исследований можно сделать следующий вывод: интервал адаптации оказывает определенное влияние на эффективность работы коэволюционного алгоритма и можно дать определенные рекомендации по его величине.

Хорошую надежность алгоритм показывает при величине интервала адаптации от 5-ти до 7-ми поколений популяции. Причем надежность, показанная алгоритмом при данных размерах интервала примерно одинаковая, что позволяет не заострять особого внимания на выставлении какой-либо определенной величины интервала адаптации для каждой конкретной задачи, а взять за основу размер интервала адаптации равным 6 поколений.

3). При работе с коэволюционным алгоритмом важным является вопрос: сколько и какого характера алгоритмы надо включать в состав коэволюционного алгоритма.

Для ответа на этот вопрос необходимо провести исследования:

3.1. Зависимость эффективности коэволюционного алгоритма от количества включенных алгоритмов.

Для проверки эффективности в состав коэволюционного алгоритма по одному включались алгоритмы (начиная с наиболее эффективных). В табл. 1 приведен пример результатов исследования эффективности КА от величины интервала адаптации на трех тестовых функциях.

Таблица 1

Пример результатов экспериментальных исследований зависимости надежности коэволюционного алгоритма от количества индивидуальных алгоритмов

Кол-во	Функция Шекеля (n=2)		Функция Сомбреро (n=2)		Функция Гриванка (n=10)	
	надежность	скорость	надежность	скорость	надежность	скорость
2	10%	3	30%	3	-	-
3	15%	2	35%	4	10%	3
4	25%	5	40%	5	15%	4
5	10%	3	30%	3	10%	2
6	10%	4	20%	2	5%	2
7	10%	8	20%	4	-	-
8	5%	3	10%	3	-	-
9	5%	2	15%	6	-	-
10	5%	2	10%	3	-	-
11	-	-	10%	2	-	-
12	-	-	5%	1	-	-
13	-	-	5%	1	-	-

Использовались следующие настройки коэволюционного алгоритма: размер популяции – 210, максимальное число вычислений целевой функции – 10000, интервал адаптации – 5, прогонов – 20.

Из табл. 1 видно, что хороший результат дает комбинация из 3-х, 4-х или 5-ти алгоритмов. Однако надежность, показанная в таблице не очень высокая – это из-за того что в состав коэволюционного алгоритма по очереди включались самые лучшие алгоритмы и иногда получалось так, что при комбинации 4-х алгоритмов поиск вели только алгоритмы с глобальным характером поиска и алгоритмы с неизвестным характером поиска. А для тестовой функции Шекеля, например, очень важно

использование также и алгоритма с локальным характером поиска – это позволяет более точно найти глобальный оптимум.

Таким образом, с помощью проведенного исследования мы узнали рекомендуемое количество алгоритмов.

3.2. Теперь для более полной оценки эффективности коэволюционного алгоритма необходимо выяснить алгоритмы какого характера необходимо включать в состав коэволюционного алгоритма. Для этого проверялись следующие комбинации алгоритмов (выбиралось 5 алгоритмов):

- комбинация 1: выбор алгоритмов делался исходя из “здравого смысла”: в состав коэволюционного алгоритма включались 2 лучших алгоритма с глобальным характером поиска, 2 лучших алгоритма с локальным характером поиска и 1 лучший алгоритм с неизвестными свойствами;
- комбинация 2: выбор алгоритмов производился случайно (но при этом исключался вариант включения в коэволюционный алгоритм одного и того же алгоритма 2 раза);
- комбинация 3: в состав коэволюционного алгоритма были включены только лучшие алгоритмы (при этом не учитывался характер поиска);
- комбинация 4: в состав коэволюционного алгоритма были включены только худшие алгоритмы (при этом не учитывался характер поиска);
- комбинация 5: в состав коэволюционного алгоритма были включены алгоритмы, обладающие глобальным характером поиска;
- комбинация 6: в состав коэволюционного алгоритма были включены алгоритмы, обладающие локальным характером поиска;
- комбинация 7: в состав коэволюционного алгоритма были включены алгоритмы, обладающие неизвестным характером поиска.

Таблица 2

Результаты экспериментального исследования эффективности коэволюционного алгоритма от характера включенных алгоритмов

Номер комбинации	Функция Шекеля (n=2)		Функция Сомбреро (n=2)		Функция Гриванка (n=10)	
	надежность	скорость	надежность	скорость	надежность	скорость
1	85%	4	75%	6	55%	3
2	30%	5	35%	5	30%	6
3	10%	3	30%	3	10%	2
4	10%	5	10%	4	-	-
5	25%	2	25%	4	20%	6
6	-	-	5%	7	-	-
7	20%	7	25%	4	10%	5

По результатам исследования можно сделать следующие выводы:

1. Наибольшая надежность коэволюционного алгоритма получена при выборе алгоритмов исходя из “здравого смысла”. При сравнении с показателями надежности индивидуальных алгоритмов было установлено, что коэволюционный алгоритм не всегда лучше самого лучшего из индивидуальных алгоритмов, но всегда лучше худшего из алгоритмов и даже лучше чем средний алгоритм.

2. При комбинации наихудших алгоритмов в виде коэволюционного алгоритма получен показатель надежности не уступающий надежности «лучшего» из худших алгоритмов. То есть, можно сказать, что комбинация из практически неработающих алгоритмов имеет большую надежность, чем надежность каждого из этих алгоритмов по отдельности.

3. При комбинации алгоритмов, выбранных случайным образом, показатель надежности практически одинаков для всех тестовых функций и выше чем для всех остальных проверенных комбинаций выбора (кроме «здравого смысла»). Это позволяет надеяться на то, что даже при полном отсутствии знаний и опыта у ЛПР (когда не получается возможность выбора исходя из “здравого смысла”) можно выбирать комбинацию алгоритмов случайным образом и она даст не самый худший результат.

Выводы

В данной лекции рассмотрены идея и основные шаги коэволюционного алгоритма, и представлены результаты численного исследования зависимости коэволюционного алгоритма от значений его параметров.

Сравнительный анализ эффективности коэволюционного алгоритма и индивидуальных генетических алгоритмов показал, что коэволюционный алгоритм лучше, чем «худший» из индивидуальных алгоритмов, и даже лучше чем «средний» алгоритм, но не всегда лучше чем «лучший» из алгоритмов.

В результате исследования зависимости эффективности коэволюционного алгоритма от величин его параметров было установлено, что для настройки коэволюционного алгоритма можно дать несколько общие рекомендации:

1. Не нужно сильно штрафовать проигравшие алгоритмы, но в то же время нельзя оставлять им больших «социальных» гарантий.
2. Наибольшую эффективность (надежность) работы коэволюционного алгоритма дает комбинация из 4-х или 5-ти алгоритмов.
3. Интервал адаптации следует брать равным 5-6 поколениям коэволюции.
4. Выбор алгоритмов необходимо делать исходя из «здравого смысла», то есть заведомо создавать комбинации из алгоритмов с глобальным и локальным характерами поиска.

Выработанные рекомендации по настройке параметров коэволюционного алгоритма позволяют конечным пользователям, не владеющим аппаратом эволюционной оптимизации, решать сложные задачи оптимизации, возникающие в реальной практике.

Самонастраивающиеся коэволюционные алгоритмы моделирования

Исследование с помощью математических моделей зачастую является единственно возможным способом изучения сложных систем и решения важнейших практических задач управления. Однако на практике сложно зафиксировать свойства функциональной зависимости выходных параметров от входных величин, еще сложнее привести аналитическое описание такой зависимости. Одним из способов решения данной проблемы является применение эволюционных алгоритмов путем решения задачи символьной регрессии методом генетического программирования. Символьная регрессия дает математическое выражение в символьной форме, которое можно подвергнуть содержательному анализу, упростить и далее использовать для моделирования или оптимизации. Эффективность применения алгоритмов генетического программирования определяется тщательной настройкой их параметров, что препятствует их более широкому распространению, т.к. требует от конечных пользователей высокой квалификации и большого опыта в применении эволюционного моделирования, что редко наблюдается на практике. Поэтому необходимо специальные алгоритмы, целью которых является совершенствование процесса моделирования сложных систем с помощью эволюционных алгоритмов, направленное на обеспечение возможности адаптивного

выбора эффективного алгоритма моделирования в ходе решения задачи символьной регрессии.

Обоснование коэволюционного алгоритма

Путем комбинации различных параметров настройки алгоритма генетического программирования можно получить множество индивидуальных алгоритмов.

В связи с тем, что алгоритм генетического программирования является стохастической процедурой, оценка его эффективности (надежности) осуществляется усреднением количества найденных решений по многократным запускам.

Базовые принципы коэволюции также применимы и для задачи моделирования. Поэтому вспомним только основную идею и схему коэволюционного алгоритма.

Коэволюционный алгоритм генетического программирования имеет следующую схему, представленную на рис. 1

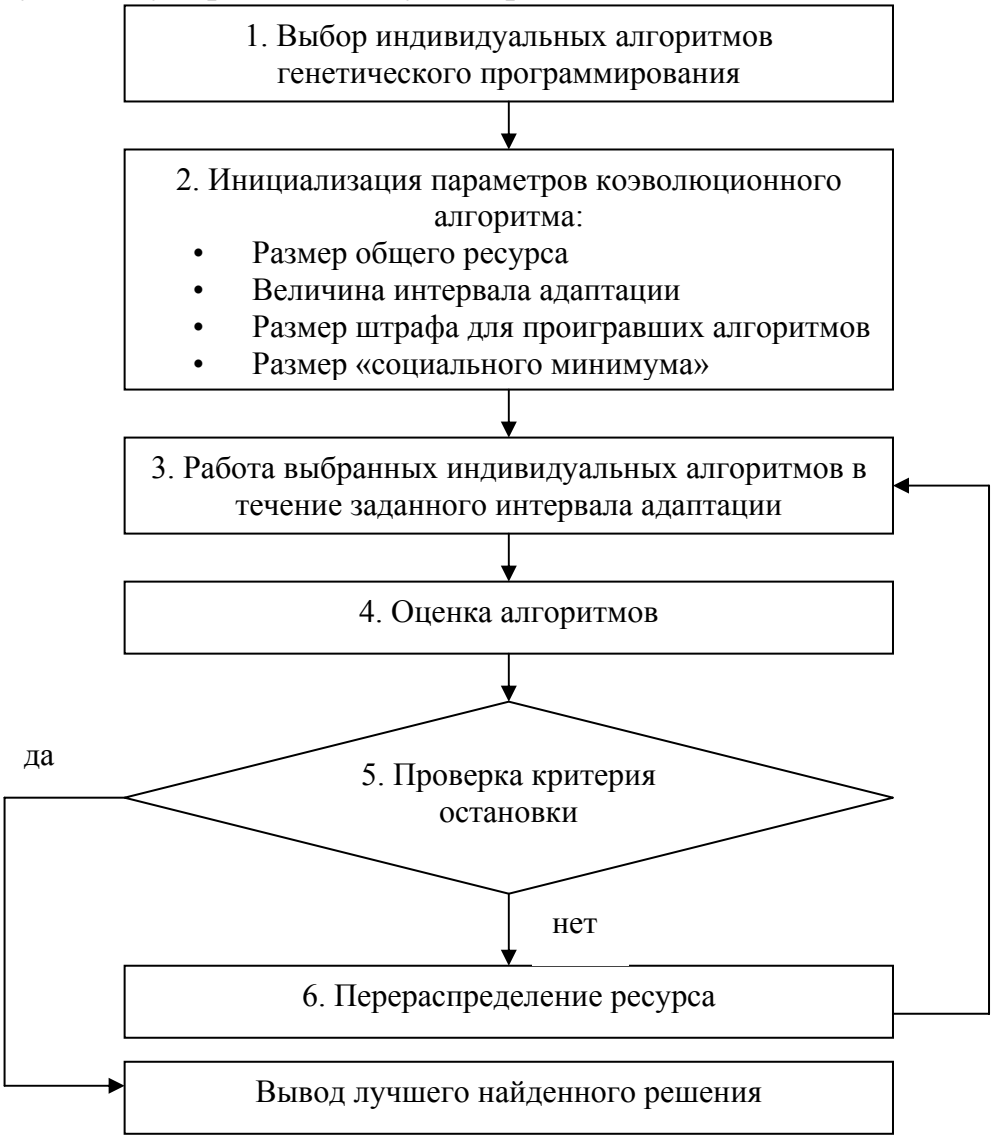


Рисунок 1. Схема коэволюционного алгоритма

Рассмотрим шаги схемы более подробно:

1. Выбор индивидуальных алгоритмов

На данном этапе необходимо выбрать индивидуальные алгоритмы (из множества имеющихся в распоряжении алгоритмов) для включения их в состав коэволюционного алгоритма. В качестве индивидуальных алгоритмов берется множество алгоритмов генетического программирования.

Таким образом, для включения в коэволюционный алгоритм предлагается максимум 8 алгоритмов, так как:

- Во-первых, большее количество алгоритмов нецелесообразно, потому что для эффективного применения стратегии поиска каждому алгоритму необходим некоторый минимум вычислительного ресурса;
- Во-вторых, конкуренция за вычислительный ресурс большего числа алгоритмов приведет к эффекту «качелей», т.е. необоснованному переключению между разными стратегиями поиска.

Если ни один из восьми представленных алгоритмов с предустановленными параметрами не отвечает требованиям практики, то их можно настроить, изменив параметры.

2. Задание параметров коэволюционного алгоритма

- *Задание общего ресурса*

Задаем ресурс, который является общим и сначала поровну разбивается на каждую популяцию. Ресурсом является количество индивидов.

- *Задание величины интервала адаптации*

В течение определенного количества шагов (так называемого интервала адаптации) каждый алгоритм работает по отдельности. Величина интервала адаптации вводится исследователем. Естественно, что его значение не должно быть маленьким (алгоритмы не успеют проявить себя) и большим (будет мало времени на адаптацию).

- *Задание величины штрафа проигравшего алгоритма*

Это некоторый процент от размера популяции индивидуального алгоритма, на который мы будем сокращать размеры популяции проигравших алгоритмов. Задается исследователем.

- *Задание размера «социального минимума»*

«Социальный минимум» – это некоторый процент от размера популяции индивидуального алгоритма, являющийся минимальным гарантированным размером популяции. То есть, мы можем сокращать проигравшую популяцию только до тех пор, пока она не достигнет минимального гарантированного размера – «социального минимума». Задается исследователем.

3. Работа алгоритмов в течение интервала адаптации

В течение заданного количества шагов (так называемого интервала адаптации) каждый алгоритм работает по отдельности, самостоятельно решая задачу моделирования.

4. Оценка алгоритмов

Качество популяции конкретного алгоритма можно вычислить по формуле, рассмотренной в лекции «Самонастраивающиеся коэволюционные алгоритмы оптимизации».:

$$q_i = \sum_{k=0}^{T-1} \frac{T-k}{k+1} \cdot b_i(k)$$

В дальнейшем, полученные оценки индивидуальных алгоритмов, входящих в состав коэволюции, используются в процессе перераспределения ресурса.

5. Проверка критерия остановки

В случае выполнения одного из критериев остановки работа алгоритма прекращается и выводится наилучшее найденное решение, иначе, на основе оценок

индивидуальных алгоритмов, полученных на шаге 4, происходит перераспределение ресурса (шаг 6).

6. Перераспределение ресурса

Изменение размера ресурса происходит путем сокращения каждой проигравшей популяции на некоторый процент (определенный заранее размер штрафа) и увеличением победившей популяции на число, равное сумме потерь проигравших. Таким образом, общее число индивидов остается неизменным. Однако необходимо не забывать, что мы можем сокращать проигравшую популяцию только до тех пор, пока она не достигнет минимального гарантированного размера – «социального минимума».

После перераспределения ресурсов алгоритмы продолжают свою работу с популяциями измененных размеров, в которые входят только лучшие индивиды из предыдущих популяций. Далее, повторяем шаги 3 - 6, пока не выполнится критерий останова, либо не кончится ресурс, выделенный на решение задачи.

Исследование зависимости свойств коэволюционного алгоритма генетического программирования от выбора его параметров

На рис. 2-3, показаны примеры поведения лучшего индивида популяции, по оси абсцисс - количество поколений, по оси ординат – значения функции пригодности. Коэволюционный алгоритм показан сплошной жирной красной линией. А на рис. 4 и 5, показано происходящее при этом изменение размеров популяций, по оси абсцисс - количество поколений, по оси ординат – размер популяции алгоритма. На всех рисунках показан усредненный результат по 20 прогонам алгоритмов.

В результате исследований нетрудно заметить, что коэволюционный алгоритм лучше, чем «худший» из индивидуальных алгоритмов, и даже лучше чем «средний» алгоритм, но не всегда лучше чем «лучший» из алгоритмов (сравнение производилось по показателю надежности алгоритмов).

Таким образом, подтверждается предположение о том, что нельзя все отнимать у алгоритма (выгодность «социального минимума»), так как проигрывающие на начальных шагах алгоритмы могут в дальнейшем помочь найти лучшее решение.

Эффективность коэволюционного алгоритма генетического программирования проверена на множестве тестовых функций с усреднением по многим прогонам.

Использовались следующие настройки коэволюционного алгоритма:

- Размер ресурса – 400;
- Начальная глубина деревьев – 3;
- Интервал адаптации – 5;
- Размер штрафа – 10% от величины популяции индивидуального алгоритма;
- Размер социального минимума – 10% от величины популяции индивидуального алгоритма;
- Количество индивидуальных алгоритмов – 4.

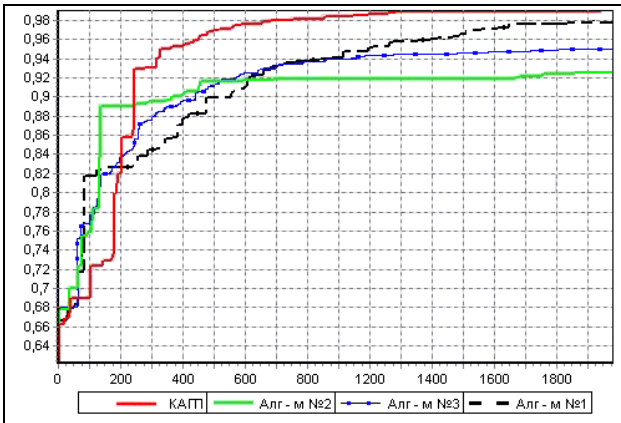


Рисунок 2. Поведение лучшего индивида

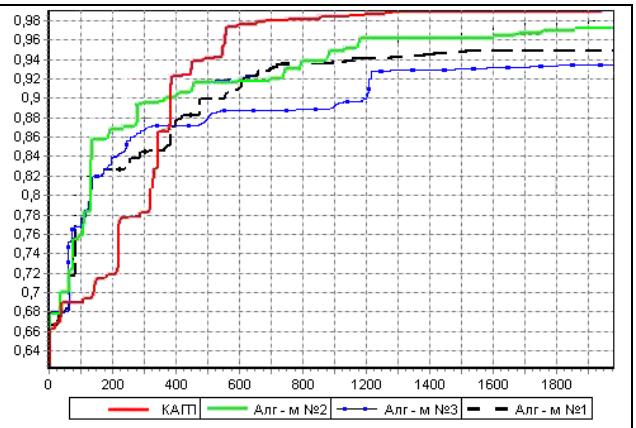


Рисунок 3. Поведение лучшего индивида

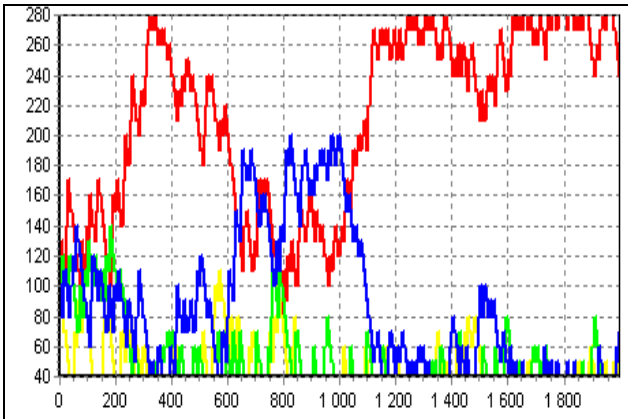


Рисунок 4. Перераспределения ресурса

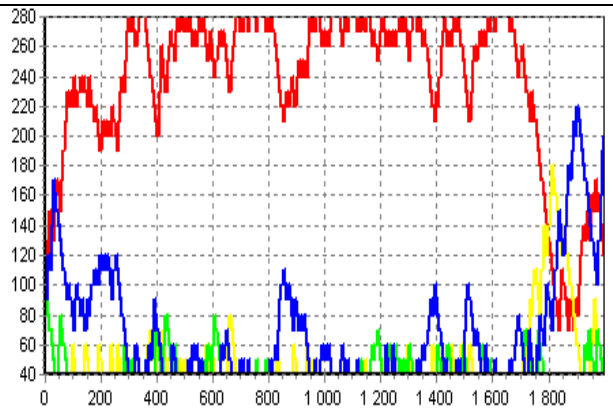


Рисунок 5. Перераспределения ресурса

В процессе исследования изменялись различные параметры алгоритма, с каждой комбинацией проводилось 20 прогонов по 2000 поколений и вычислялись следующие характеристики:

1. *E* – ошибка аппроксимации лучшего индивида (учитываются только те прогоны, в которых было получено решение с E < 1%):

$$\mathrm{Error}(A_j) = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} \left(y_i - \mathrm{evaluate}(A_j, \overline{x_i}) \right)^2}$$ (Евклидово расстояние)

2. *G* – среднее поколение на котором было получено решение с E < 1%.

3. C_n – среднее количество узлов в лучшем индивиде (учитываются только те прогоны, в которых было получено решение с E < 1%).

4. N_1 – количество прогонов, в которых было получено решение с E >5%, в процентах от общего количества прогонов.

5. N_2 – количество прогонов, в которых было получено решение с 1% < E < 5%, в процентах от общего количества прогонов.

6. N_3 – количество прогонов, в которых было получено решение с E < 1%, в процентах от общего количества прогонов.

Для решения задачи символьной регрессии предложена следующая функция пригодности:

$$\text{fitness}(A_j) = \frac{Z}{Z + \text{Error}(A_j)} \cdot \text{complexity}(A_j) \qquad (*)$$

$$\text{if} \quad \min(\bar{y}) \geq 0, \text{then} \quad Z = \max(\bar{y})$$

$$\text{if} \quad \min(\bar{y}) < 0 \,\&\&\, \max(\bar{y}) > 0, \text{then} \quad Z = (\max(\bar{y}) - \min(\bar{y}))$$

$$\text{if} \quad \min(\bar{y}) < 0 \,\&\&\, \max(\bar{y}) \leq 0, \text{then} \quad Z = \min(|\bar{y}|)$$

$$\text{Error}(A_j) = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} \left(y_i - \text{evaluate}(A_j, \overline{x_i})\right)^2}$$

$$\text{complexity}(A_j) = 1 - \frac{1}{C_j(n) \cdot k^2} \cdot e^{\frac{C_j(n)}{k}}$$

где $\text{fitness}(A_j)$ - пригодность решения A_j, $\text{Error}(A_j)$ - норма погрешности (квадратичная ошибка аппроксимации, вычисляемая по всем точкам обучающей выборки), N - объем обучающей выборки, $\text{evaluate}(A_j, \overline{x_i})$ - значение полученного выражения A_j в точке $\overline{x_i}$, $\text{complexity}(A_j)$ - некоторый штраф на сложность дерева, выраженный в общем количестве вершин, $C_j(n)$- количество вершин решения A_j, k - коэффициент штрафа, задаваемый пользователем.

При рассмотрении основных шагов коэволюционного алгоритма видно, что на эффективность коэволюционного алгоритма могут оказать влияние величины его параметров: размер штрафа и размер “социального минимума” и т.д. Для проверки данного предположения были проведены следующие исследования работы коэволюционного алгоритма:

а). Исследование эффективности коэволюционного алгоритма в зависимости от величины штрафа и социального минимума. Исследование проводилось в случае включения в состав коэволюционного алгоритма четырех индивидуальных алгоритмов, определенных случайным образом.

Таблица 1.

Результаты экспериментального исследования эффективности коэволюционного алгоритма генетического программирования в зависимости от размеров штрафа и социального минимума по показателю N_1

Соц. минимум \штраф	2%	4%	6%	8%	10%	12%	14%	16%
2%	20	25	20	15	15	20	20	25
4%	20	15	20	10	15	10	15	20

6%	20	15	15	15	5	10	10	15
8%	15	20	15	10	5	15	10	20
10%	25	15	10	10	0	5	5	20
12%	20	20	15	5	0	5	15	25
14%	20	25	15	10	10	15	25	20
16%	25	25	20	15	20	15	20	25

Таблица 2.

Результаты экспериментального исследования эффективности коэволюционного алгоритма генетического программирования в зависимости от размеров штрафа и социального минимума по показателю N_2

Соц. минимум \штраф	2%	4%	6%	8%	10%	12%	14%	16%
2%	65	55	60	80	75	75	60	65
4%	70	70	70	70	75	65	75	75
6%	75	75	60	65	70	70	75	80
8%	70	75	70	75	80	65	85	65
10%	55	60	75	70	55	55	75	65
12%	70	75	65	70	70	70	70	70
14%	65	65	70	80	70	70	60	80
16%	75	70	75	75	65	75	75	65

Таблица 3.

Результаты экспериментального исследования эффективности коэволюционного алгоритма генетического программирования в зависимости от размеров штрафа и социального минимума по показателю N_3.

Соц. минимум \штраф	2%	4%	6%	8%	10%	12%	14%	16%
2%	15	20	20	5	10	5	20	10
4%	10	15	10	20	10	25	10	5
6%	5	10	25	20	25	20	15	5
8%	15	5	15	15	15	20	5	15
10%	20	25	15	20	45	40	20	15
12%	10	5	20	25	30	25	15	5
14%	15	10	15	10	20	15	15	0
16%	0	5	5	10	15	10	5	10

Таким образом, можно сделать вывод: для коэволюционного алгоритма генетического программирования можно дать следующие рекомендации:

- размер социального минимума необходимо выбирать в пределах 10–12% от размера популяции индивидуального алгоритма.
- размер штрафа необходимо выбирать в пределах 10–12% от размера популяции индивидуального алгоритма.

Из всего вышесказанного можно сделать еще один вывод: не нужно сильно штрафовать проигравшие алгоритмы, но в то же время нельзя оставлять им больших “социальных” гарантий.
Наибольшая эффективность коэволюционного алгоритма была получена при величине штрафа 10% и социального минимума 10% от размера популяции индивидуального алгоритма. В дальнейшем в работе будут использоваться эти значения параметров.
б) Другим не менее важным параметром является величина интервала адаптации.
Результаты исследования показаны в таб. 4 и на рис. 6.

Таблица 4.

Результаты экспериментального исследования эффективности коэволюционного алгоритма генетического программирования в зависимости от величины интервала адаптации

Интервал адаптации	N_1, %	N_2, %	N_3, %
2	10	75	15
3	30	60	10
4	10	55	35
5	5	50	45
6	10	65	25
7	5	85	10
8	10	75	15
9	15	80	5
10	15	75	10

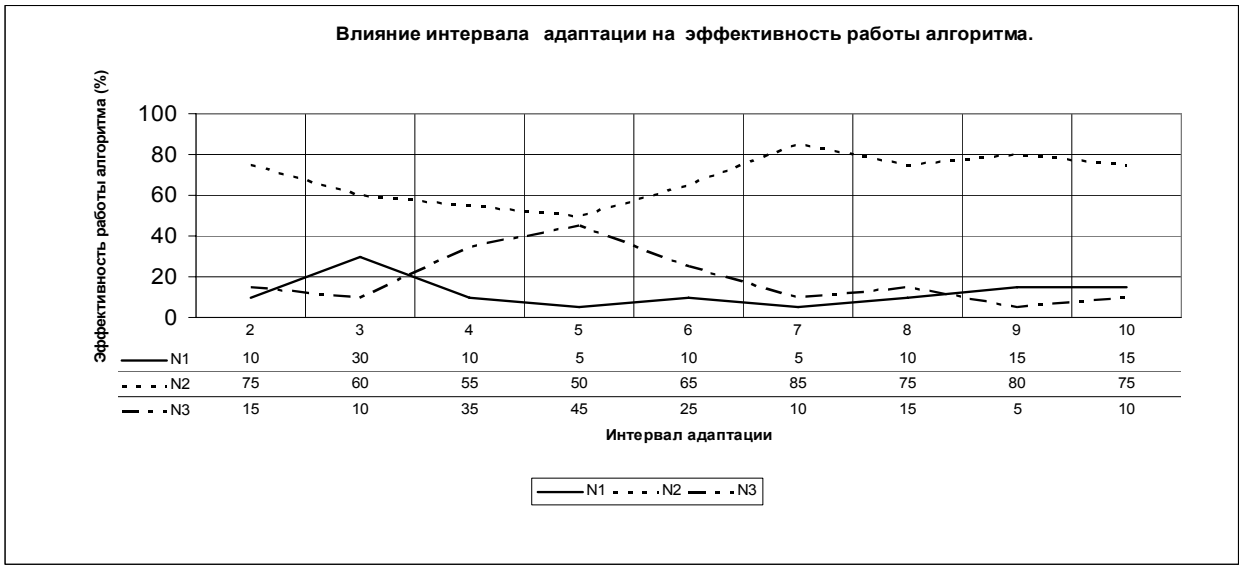


Рисунок 6. Влияние интервала адаптации на эффективность работы алгоритма

Таким образом, можно сделать вывод: интервал адаптации необходимо выбирать равным 4-5.

в). Еще одним важным параметром, от которого зависит эффективность работы алгоритма, является количество индивидуальных алгоритмов генетического программирования выбранных для включения в коэволюцию.

В ходе исследований было установлено, что использование параметра - штрафа, равного некоторому проценту от величины популяции индивидуального алгоритма, является неэффективным, так как данная величина является фиксированной и не зависит от количества индивидуальных алгоритмов. Применение размера штрафа, равного некоторому проценту от величины популяции индивидуального алгоритма, является эффективным только в том случае, когда размер популяции индивидуального алгоритма равен 100 индивидам. Уменьшение или увеличение этой величины отрицательно сказывается на эффективности работы коэволюционного алгоритма. Для устранения этой проблемы предложена следующая процедура расчета размера штрафа в зависимости от количества индивидуальных алгоритмов ГП и общего числа индивидов (рис. 7) :

$$Z = \left(\frac{SpCA}{100} \cdot zh \right) \cdot 2^{-\frac{N}{2}}$$

где Z – количество индивидов, на которое будут штрафоваться проигравшие индивидуальные алгоритмы, SpCA – общее число индивидов, zh – устанавливаемый пользователем размер штрафа в процентах, N – количество индивидуальных алгоритмов метода генетического программирования.

Так как изменилась процедура вычисления размера штрафа, то и величина «социальный минимум» также должна вычисляться по-другому и зависеть, как и размер штрафа, от общего числа индивидов. Как показало исследование, малый размер социального минимума приводит к преждевременной сходимости и стагнации. Поэтому было принято решение устанавливать величину размера социального минимума, как некоторый процент от общего числа индивидов.

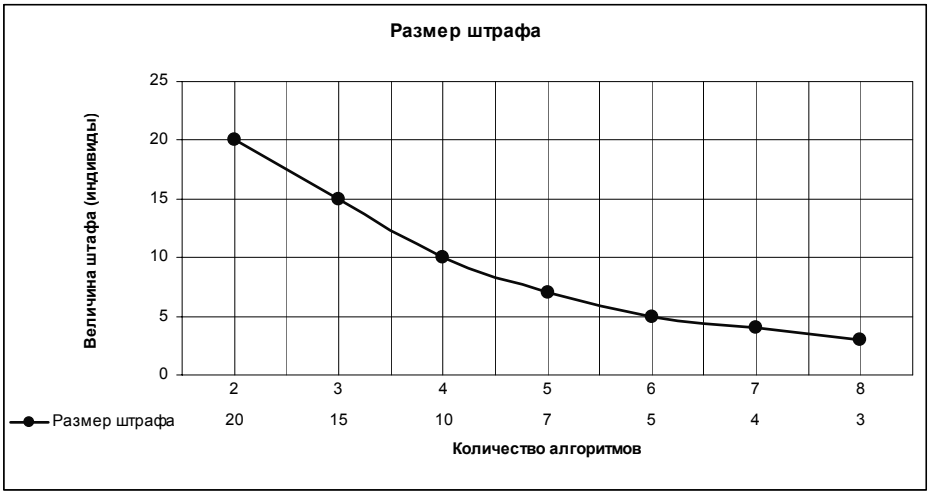


Рисунок 7. Размер штрафа, выраженный в количестве индивидов, зависящий от количества индивидуальных алгоритмов

Учитывая внесенные изменения, было проведено исследование эффективности коэволюционного алгоритма в зависимости от количества индивидуальных алгоритмов генетического программирования. Результаты исследования показаны в таб. 5 и на рис. 8.

Таблица 5.

Результаты экспериментального исследования эффективности коэволюционного алгоритма генетического программирования в зависимости от количества индивидуальных алгоритмов метода генетического программирования

Количество алгоритмов	N_1, %	N_2, %	N_3, %
2	5	80	15
3	15	50	35
4	5	50	45
5	10	50	40
6	15	50	35
7	20	50	30
8	20	60	20

В результате исследований эффективности работы коэволюционного алгоритма в зависимости от параметров, входящих в его настройку, было установлено, что:

- коэволюционный алгоритм всегда лучше самого худшего из алгоритмов и даже лучше чем средний алгоритм (по показателям надежности);
- интервал адаптации необходимо выбирать равным 4-5;
- количество индивидуальных алгоритмов должно быть 3-5;

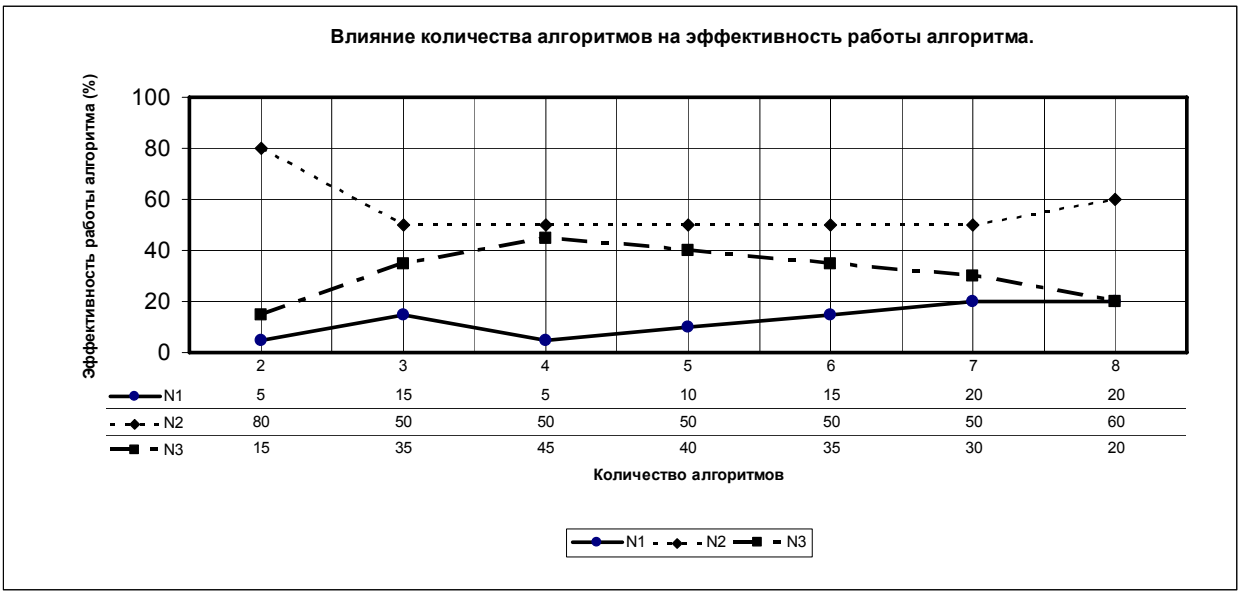


Рисунок 8. Влияние количества индивидуальных алгоритмов на работу алгоритма

- не нужно сильно штрафовать (размер штрафа 10-12%) проигравшие алгоритмы, но в то же время не оставлять им больших «социальных» гарантий (10-12%);
- при отсутствии уверенности в своих знаниях о свойствах эволюционных алгоритмов включение в коэволюцию случайно выбранных алгоритмов позволяет получать достаточно хорошие результаты;
- в случае практически полного отсутствия знаний о предпочтительности выбора индивидуальных алгоритмов даже комбинация из практически неработающих алгоритмов

(худших по надежности в сравнении с другими индивидуальными алгоритмами) имеет большую надежность, чем надежность каждого из этих алгоритмов по отдельности.

– 75% решений по показателю N_2 имеют ошибку E<1,6%. То есть коэволюционный алгоритм позволяет получать достаточно хорошие аппроксимации.

Приведенные примеры самонастраивающихся алгоритмов показывают, что в принципе возможно применение эволюционных алгоритмов конечными пользователями, не являющимися специалистами по эволюционным алгоритмам. Однако и коэволюционные алгоритмы имеют настройки, хотя их выбор намного проще, чем у стандартных эволюционных алгоритмов.

Еще одним направлением совершенствования практики использования эволюционных алгоритмов является разработка схем ЭА с меньшим числом настроек.

Вероятностный генетически алгоритм

На практике наибольшее распространение получили ГА с бинарным представлением решений. Формально они решают задачу псевдобулевой оптимизации, т.е.

$$f(X) \to \underset{X \in B_{2^n}}{opt}$$, где $f : B_{2^n} \to R^1$.

Нетрудно понять, что в процессе своей работы ГА накапливают и обрабатывают некоторую статистическую информацию о пространстве поиска, однако эта статистика в явном виде отсутствует. Можно предложить следующий способ представления накопленной генетическими алгоритмами статистики - на текущей итерации в соответствие популяции решений ставить вектор вероятностей:

$$P^k = (p_1^k, ..., p_n^k),\ p_i^k = P(x_i^k = 1) = \frac{\sum_{j=1}^{N} x_{ij}^k}{N},\ i = \overline{1,n} \qquad (3.1)$$

,

где p_i^k - вероятность присвоения *i*-й компоненте вектора решения X^k значения 1, *k* – номер итерации, *j* – номер решения в популяции, *N* – размер популяции.

Для изучения работы генетических алгоритмов изменения компонент вектора вероятностей будем представлять в виде графиков. Ниже представлены примеры типичного поведения компонент p_i для ГА (рис. 1).

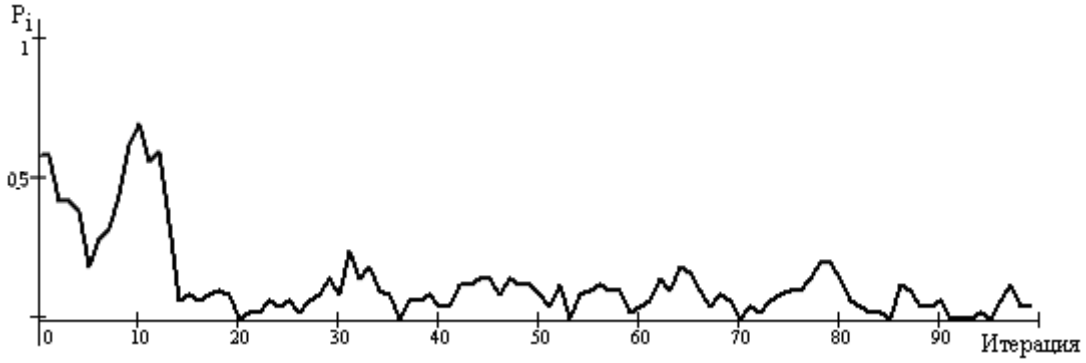


Рис. 1. Пример изменения вероятностей в ГА

Из графика видно, что вероятности меняются нелинейно, т.к. новая популяция может значительно отличаться от предыдущей. Однако, в предельном случае (мутация отсутствует, пропорциональная селекция) изменение вероятностей близко линейному.

Когда какая-либо компонента вектора вероятностей сходится к 0 или 1, поиск по этой компоненте прекращается, что может привести к «захвату» локальным минимумом. В ГА поиск не прекращается, т.к. оператор мутации «выкидывает» решения в новые регионы поискового пространства. Мутация вызывает скачкообразные изменения (скачки тем больше, чем выше мутация) компонент вектора вероятности даже после того, как компонента сходится к 0 или 1, что придает ГА глобальные свойства.

Попробуем проанализировать работу ГА с точки зрения теории псевдобулевой оптимизации, рассмотрим основные шаги ГА.

Инициализация. Если априорная информация о пространстве поиска отсутствует, то начальная популяция решений формируется случайно – точки распределяются равномерно в бинаризованном пространстве поиска, т.е. $p_i^0 = P(x_i^0 = 1) = 1/2$.

Оператор скрещивания – генетический оператор поиска, посредством которого из выбранных индивидов (родительские особи) получаются новые (потомки). Решение, представленное бинарным вектором является вершиной *n*-мерного гиперкуба в булевом пространстве. Оператор скрещивания будет генерировать точки из некоторого подмножества, размерность которого определяется числом совпадающих бит в соответствующих позициях родительских особей. Легко показать, что одноточечное или двухточечное скрещивание не может получить все точки полученного подпространства, равномерное – может.

Для примера рассмотрим действие оператора скрещивания при $n = 4$ и родителях $X = (0000)$ и $Y = (1111)$. Потомок может унаследовать любые гены родителей, т.е. решение, полученное при скрещивании может содержать 0 или 1 в любой позиции. Однако, при применении одноточечного скрещивания будут получены решения: $(0111),(1000)$, $(0011),(1100)$, $(0001),(1110)$.

В общем случае, если даны $X = (x_1, x_2, ..., x_k, x_{k+1}, ..., x_n)$ и $Y = (y_1, ..., y_k, y_{k+1}, ..., y_n)$, где $x_i = y_i, i = \overline{1,k}$ и $x_j \neq y_j, j = \overline{k+1,n}$, то с помощью равномерного скрещивания можно получить 2^{n-k} различных точек. При одноточечном скрещивании мы имеем $(n-k-1)$ «эффективных» точек разрыва, разрыв в остальных k точках не даст новых точек. При каждом «эффективном» скрещивании мы получим 2 потомка, т.е. при одноточечном скрещивании можно получить всего $2 \times (n-k-1)$ новых точек.

Очевидно, что равномерное скрещивание является наиболее полным оператором, который в частном случае может получить и одноточечное, и многоточечное скрещивание. Таким образом, с точки зрения широты охвата пространства поиска (разнообразия популяции) равномерное скрещивание предпочтительней. Исключением могут быть те случаи, когда известны свойства оптимизируемого функционала, и точечное скрещивание использует эти свойства.

Попробуем сформулировать равномерное скрещивание в «негенетических» терминах. Пусть для формирования потомка используется s родителей ($2 \le s \le r$), где r - размер популяции. Им соответствует статистика $\overline{P} = (\overline{p}_1, ..., \overline{p}_n)$, где $\overline{p}_i(x_i = 1) = \frac{1}{s} \cdot \sum_{j=1}^{S} x_i^j$. При $S = R$ статистика $\overline{P}$ совпадает с P.

Потомок формируется в соответствии со статистикой $\overline{P}$. Такой оператор поиска соответствует теории генетических алгоритмов, содержит в себе свойства оператора скрещивания и имеет математически ясную формулировку.

Оператор мутации – оператор широкого поиска, т.к. в результате его применения возможно появление решений вне данной статистики. Если компонента вектора вероятностей p_i обращается в 0 или 1, то поиск по компоненте вектора решения x_i прекращается. Мутация предотвращает появление предельных значений компонент вектора вероятностей.

Оператор селекции - оператор, который каждому решению (в зависимости от качества решения) ставит вероятность быть отобранным. Наиболее приспособленные индивиды с большей вероятностью переходят в следующее поколение. На этом шаге происходит получение апостериорной информации о поисковом пространстве. В результате статистика меняется, компоненты вектора вероятностей смещаются в сторону распределения единичных компонент наиболее приспособленных решений.

Попробуем сформулировать в «негенетических» терминах алгоритм, имеющий общую с генетическим алгоритмом схему, сохраняющий свойства генетических операторов. Такой вероятностный генетический алгоритм (ВГА) применимо к задаче (1.2) имеет следующую схему:

1. Случайным образом формируются r векторов $X_0^i \in D, i = \overline{1,r}$. Вычисляется распределение $P^0 = \{p_1^0, ..., p_n^0\}$, $p_j = P\{x_j = 1\}$, $j = \overline{1,n}$. Вычисляются соответствующие значения оптимизируемого функционала $\chi(X_0^i), i = \overline{1,r}$.

2. На k-м шаге в соответствии с распределением $P^k = \{p_1^k, ..., p_n^k\}$ формируются r векторов $X_k^i \in D$.

3. Случайным образом формируются новые решения: с вероятностью p^m компоненты векторов $X_k^i \in D$ меняются на противоположные.

4. Вычисляются соответствующие значения функционала $\chi(X_k^i), i = \overline{1,r}$.

5. Выбираются r векторов $\hat{X}_k$ из $X_{k-1} \cup X_k$ в соответствии с конкретным правилом отбора.

6. Пересчитывается распределение $P^k = \{p_1^k, ..., p_n^k\}$ единичных компонент $\hat{X}_k$.

7. Если не выполняется условие остановки, то $X_k = \hat{X}_k$, $k = k+1$, повторить шаги 2 – 6.

Если априорная информация о пространстве поиска отсутствует, $p_i^0 = 1/2, i = \overline{1,n}$. На шаге 2 выполняется поиск в соответствии с имеющейся статистикой, аналогичный скрещиванию в ГА. На шаге 3 происходит поиск вне имеющейся статистики – оператор мутации. На шаге 5 происходит получение апостериорной информации, и происходит пересчет распределения вероятностей единичных компонент.

Метод прогноза сходимости стохастических поисковых алгоритмов решения задач оптимизации с булевыми переменными

Ранее для изучения работы генетических алгоритмов было предложено изменения компонент вектора вероятностей представлять в виде графиков. Из графиков изменения вероятностей видно, что в прогонах, когда алгоритмы находят решение задачи,

компоненты вектора вероятности сходятся к соответствующим истинным значениям. Пример, такого графика в ВГА для произвольно выбранной компоненты представлен на рис. 2.

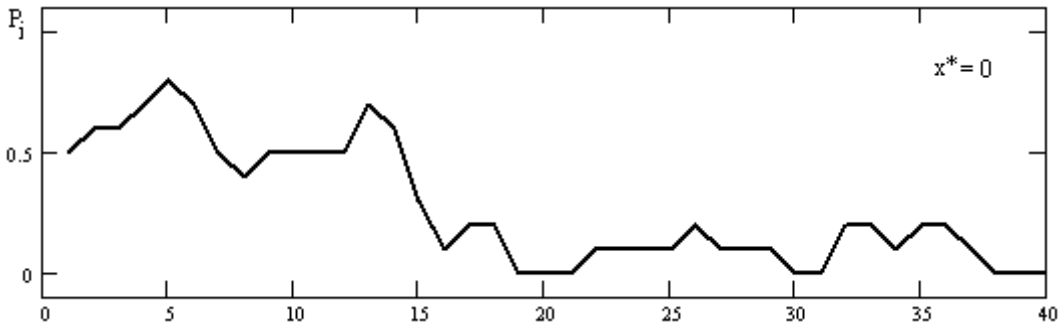


Рис. 2. Пример сходимости компоненты вектора вероятностей
к истинному значению

Поскольку ЭА являются алгоритмами случайного поиска, то принимать решение по одному запуску алгоритма нецелесообразно, т.к. в результате этого запуска решение может быть не найдено. Обычно, решение принимается после серии запусков. Усредним значения компонент вектора вероятностей по числу прогонов алгоритма. График поведения в среднем произвольно выбранной компоненты для ВГА показан на рис. 3. Из графика видно, что в среднем вероятности также сходятся к истинным значениям, однако это проявляется значительно слабее, так как в среднее значение вносят вклад и «неудачные» прогоны, в которых решение не было найдено, и вероятности принимали отличные от истинных значения.

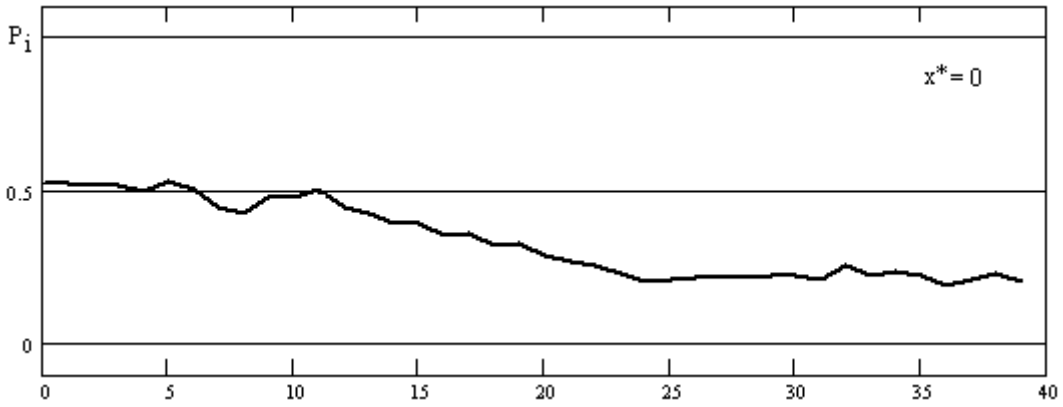


Рис. 3. Пример сходимости в среднем компоненты вектора
вероятностей к истинному значению

Таким образом, в результате анализа графиков изменения компонент вектора вероятностей в ВГА, ГА было замечено следующее свойство: изменения компонент вектора вероятностей в алгоритмах происходят целенаправленно, в среднем компоненты вектора вероятностей сходятся к истинным значениям, т.е.

$$p_i \to 1,\ если\ x_i^* = 1,$$
$$p_i \to 0, если\ x_i^* = 0 \qquad (2)$$

где p_i - *i*-я компонента вектора вероятностей, x_i^* - значение *i*-й компоненты булевого вектора решения задачи.

Если предположить, что свойство (2) справедливо для стохастических алгоритмов с бинарным представлением решений, то это свойство можно использовать для прогноза их сходимости.

На основе свойства (2) построен следующий алгоритм прогноза сходимости:

1. Для данной задачи выбрать определенную схему алгоритма, определить число итераций $i = 1,...,I$ и число прогонов алгоритма $k = 1,...,K$.

2. Набрать статистику $(p_j)_i^k, j = 1,...,n$. Усреднить $(p_j)_i$ по k. Выявить тенденцию изменения компонент p_j.

3. Считать $$x_j^{opt} = \begin{cases} 1, если\ \ p_j \to 1, \\ 0, если\ \ p_j \to 0 \end{cases}. \qquad (3)$$

Однако в результате частного прогона при малых I решение X^* может быть не найдено, и p_j может принимать значения отличные от p_j^*. Тогда среднее значение компонент может сместиться в противоположную сторону, либо слабо отличаться от $1/2$. Поэтому для обеспечения сходимости к оптимуму, было предложено придавать веса компонентам вектора вероятностей в соответствии с результатом прогона. Т.е. наилучшие прогоны вносят больший вклад в среднее значение, наихудшие – меньший. Для прогноза вместо (3) можно использовался следующий интегральный критерий:

$$x_j^{opt} = \begin{cases} 1, & если\ \sum_{i=1}^{I}(p_j - 0{,}5) > 0, \\ 0, & иначе \end{cases}. \qquad (4)$$

Таким образом, решение о том, что $p_i \to 1$ принимается, когда p_j чаще оказывается больше 0.5 (а также дальше от 0.5). Пример усреднения для произвольно выбранных компонент вектора вероятностей в алгоритме прогноза сходимости ВГА показан на рисунке 9. Сплошная линия – с использованием критерия (3.3) и пунктирная – с интегральным критерием (4).

Из графиков видно, что взвешенное усреднение с интегральным критерием позволяет выявить тенденцию изменения компонент вектора вероятностей за меньшее число итераций, т.к. уменьшает вклад «неудачных» прогонов.

Алгоритм прогноза сходимости ВГА позволяет выявить тенденцию изменения компонент вектора вероятностей за меньшее число итераций, чем алгоритм прогноза для обычного ГА, т.к. ВГА более активно использует статистику о пространстве поиска и превосходит ГА по быстродействию. Алгоритм прогноза с взвешенным усреднением всегда выигрывает по надежности, т.к. уменьшает вклад неудачных прогонов и увеличивает

вклад удачных. С увеличением числа итераций в прогоне I, надежность прогноза растет. При I близких к среднему числу итераций до нахождения оптимума, алгоритм прогноза имеет надежность 100%.

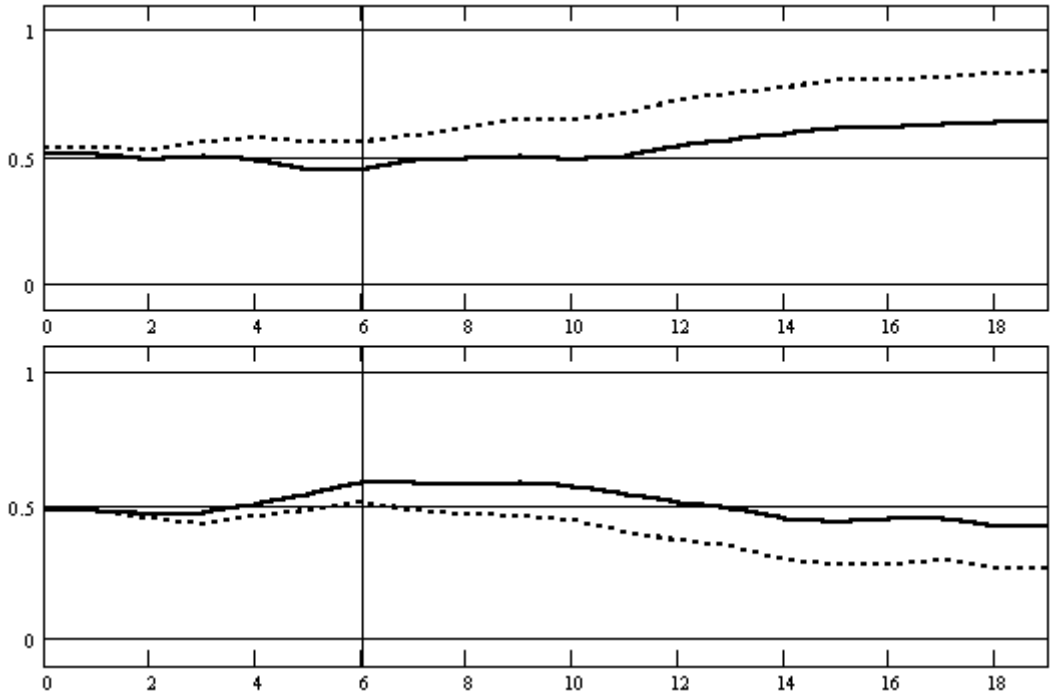


Рис. 9. Пример усреднения компонент вектора вероятностей
в алгоритме прогноза

В общем случае, алгоритм прогноза может быть использован и для любых других стохастических алгоритмов псевдобулевой оптимизации.

Тщательное тестирование вероятностного генетического алгоритма показало, что его эффективность практически не отличается от эффективности обычного ГА не только для задач безусловной однокритериальной оптимизации, но и в задачах условной и многокритериальной оптимизации. Это позволяет сделать вывод о перспективности данного направления, т.к. ВГА столь же эффективен, как и ГА, но обладает меньшим числом настраиваемых параметров (отсутствует скрещивание и может быть исключена мутация), что само по себе существенно для практики, к тому же он работает значительно быстрее, т.к. отсутствующие генетические операторы на должны выполняться в ходе оптимизации.

МЕТОД АЛЬТЕРНАТИВНОЙ АДАПТАЦИИ ДЛЯ УСКОРЕННОГО ВЫБОРА ЭФФЕКТИВНОГО ЭВОЛЮЦИОННОГО АЛГОРИТМА

Как уже было отмечено ранее, генетический алгоритм (ГА) это метод оптимизации, основанный на имитации процессов естественной эволюции. Генетические алгоритмы находят применение повсюду: в экономике, в промышленности, медицине и т.д. Это означает, что применять их должны не только математики-оптимизаторы, но и специалисты из других областей, не являющиеся экспертами в области эволюционной оптимизации.

Одним из недостатков, существенно ограничивающим широкое применение генетических алгоритмов является то, что для их успешной работы необходима их настройка. Эффективность генетического алгоритма напрямую и существенно зависит от его настроек, и может варьироваться от 0 до 100 процентов. Для пользователей, не знакомых с эволюционной оптимизацией, выбор эффективных настроек для ГА может стать дополнительной сложной задачей. Кроме того, если алгоритм удачно справляется с одной, конкретной задачей, то не факт, что он будет так же хорошо решать другие задачи, при неизменных настройках.

Поэтому, необходимо определить способ, благодаря которому, можно быстро и эффективно выбирать настройку алгоритма, для решения поставленной задачи. При этом надо учитывать, что конечный пользователь должен обладать минимальным опытом в эволюционной оптимизации.

Методы альтернативной адаптации

Одним из стандартных инженерных подходов к определению эффективности эволюционного алгоритма является исчерпывающий анализ на конкретной задаче при всех сочетаниях настроек с многократными прогонами для усреднения («инженерный» подход, позволяющий получить решение в каждом конкретном случае). Такой подход малоприемлем на практике, так как он требует много времени и других ресурсов.

В ходе экспериментов с инженерным подходом, было замечено, что обычно приходится выбирать один алгоритм из нескольких, уже являющихся более или менее подходящими, а не из всех возможных. В простейшем случае осуществляется выбор из двух альтернатив. Одним из наилучших способов выбора наилучшего варианта из двух является метод альтернативной адаптации, предложенный Л. А. Растригиным в 1968 г.

Идея метода альтернативной адаптации с линейной тактикой состоит в следующем: повторять удачное (+) действие (т.е. выбирать ту альтернативу, которая проявила себя хорошо) и переходить на другую при неудаче (-). Схематично это показано на рисунке 1.

Идея метода альтернативной адаптации с обучением состоит в следующем: при неудаче (-) альтернатива сменяется не всегда, а с некоторой вероятностью, которая вычисляется по результатам наблюдения за работой алгоритма.

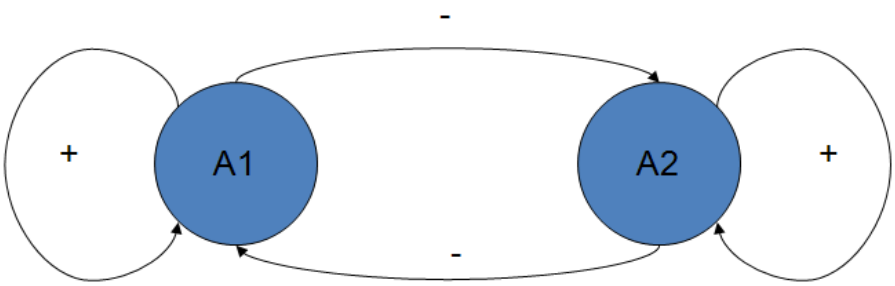


Рисунок 1 – Альтернативная адаптация с линейной тактикой

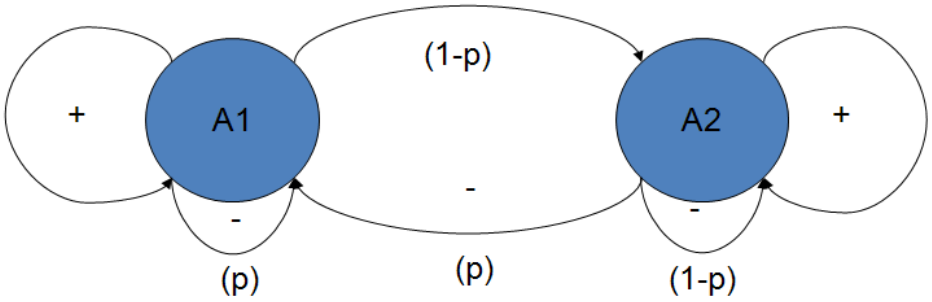


Рисунок 2 – Альтернативная адаптация с обучением

В результате проведенных исследований было предложено применить метод альтернативной адаптации с линейной тактикой и метод альтернативной адаптации с обучением к выбору эффективных настроек для ГА.

Правило применения метода альтернативной адаптации с линейной тактикой для ГА формулируется следующим образом:

Если в текущем прогоне ГА сработал хуже, чем в предыдущем, и результат оказался хуже, чем наилучший результат альтернативного алгоритма, то в следующем прогоне мы решаем задачу альтернативным алгоритмом. При этом неважно, какой из альтернативных алгоритмов работал в текущем прогоне.

Правило применения метода альтернативной адаптации с обучением для ГА будет сформулировано следующим образом:

Если в текущем прогоне ГА результат решения задачи оказался хуже, чем наилучший результат альтернативного алгоритма, то право решения задачи отдаем альтернативному алгоритму с вероятностью $p_1 = \frac{N_1}{N}$. Где N_1 – число раз, которое решал задачу альтернативный алгоритм, N – общее число решений задачи.

Экспериментальное исследование

Для проверки работоспособности подхода была разработана программная система, реализующая генетические алгоритмы и сформулированные выше методы альтернативной адаптации. Одно из окон программы показано на рисунке 3.

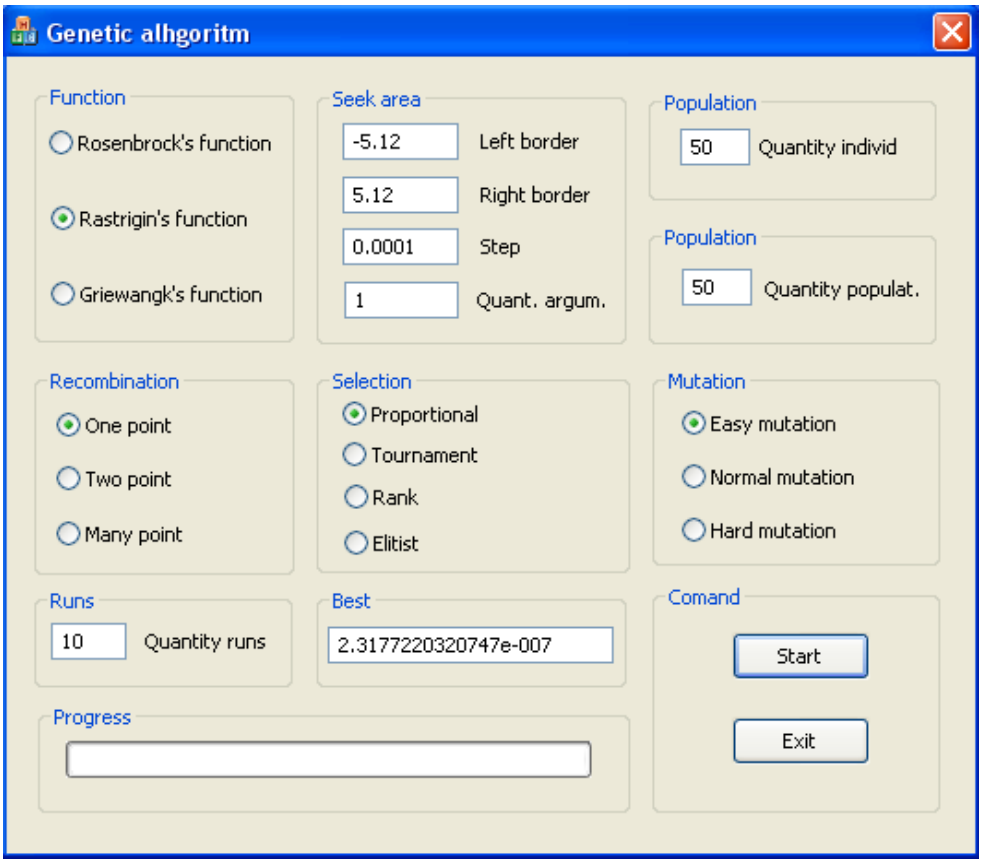


Рисунок 3 – Окно настройки генетического алгоритма

Экспериментальное исследование проводилось для трех тестовых функций: Розенброка, Растригина и Гриванка. Все три функции являются сложными для оптимизации классическими методами, так как они имеют узкий изогнутый овраг (Розенброк) или большое количество локальных экстремумов и всего один глобальный, который незначительно отличается от локальных (Растригин, Гриванк).

Сначала был применен стандартный инженерный подход, для объективного определения наилучших и наихудших настроек для ГА. После этого применялись методы альтернативной адаптации. Были рассмотрены следующие исходные ситуации.

1. *Алгоритмы, выбранные для методов альтернативной адаптации, значительно отличаются по эффективности.*

В таблице 1 приведены результаты работы метода альтернативной адаптации с линейной тактикой для функции Гриванка. Видно, что уже на первых прогонах, лучший алгоритм (1) ведет себя более уверенно, чем алгоритм (2). Последующие прогоны также подтверждают этот факт. При линейной тактике лучший алгоритм можно определить уже после 12-15 прогонов.

Таблица 1

Результаты работы метода альтернативной адаптации с линейной тактикой

№ прогона	№ альтернативы	Результат
1	1	5,6152e-7
2	2	1,8103e-3

3	1	5,6152e-7
4	1	5,6152e-7
5	1	5,6152e-7
…		
181	1	5,6152e-7
182	1	5,6152e-7
183	1	5,6152e-7
184	1	5,6152e-7

В таблице 2 приведены результаты метода альтернативной адаптации с обучением. Видно, что даже при неудачном решении задачи наилучшим алгоритмом, право решения задачи, по-прежнему остается у него (прогон №5). При применении алгоритма с обучением наилучший ГА опознан уже после 8 -10 прогонов.

Таблица 2

Результаты работы метода альтернативной адаптации с обучением

№ прогона	№ альтернативы	Результат
1	1	5,6152e-7
2	2	2,1837e-3
3	1	5,6152e-7
4	1	5,6152e-7
5	1	3,2975e-3
6	1	5,6152e-7
…		
161	1	5,6152e-7
162	1	5,6152e-7
163	1	5,6152e-7
164	1	5,6152e-7

Из результатов видно, что оба метода альтернативной адаптации достаточно быстро справляются с нахождением наилучшего генетического алгоритма, в случае, если эффективность одного из них существенно превосходит эффективность другого.

2. *Алгоритмы, выбранные для методов альтернативной адаптации одинаково плохи.*

В таблице 4 приведены результаты работы метода альтернативной адаптации с линейной тактикой. В таблице 5 приведены результаты работы метода альтернативной адаптации с обучением. В обоих случаях генетические алгоритмы сменяют друг друга часто и ненадолго. Этот характерный факт можно использовать для быстрого исключения выбранных альтернатив из дальнейшего рассмотрения, без проведения полного анализа их (не)эффективности.

Таблица 4

Результаты работы метода альтернативной адаптации с линейной тактикой

№ прогона	№ альтернативы	Результат
1	1	5,2837e-6
2	2	1,7028e-4
3	1	2,0016e-5
4	1	5,2837e-6
5	1	1,1126e-3
6	2	9,7250e-5
7	2	1,2862e-4
8	1	1,1126e-3
9	2	2,0324e-3

Таблица 5

Результаты работы метода альтернативной адаптации с обучением

№ прогона	№ альтернативы	Результат
1	1	1,1219e-5
2	2	1,2027e-4
3	1	9,8745e-6
4	1	8,5877e-6
5	1	1,2622e-5
6	2	1,5603e-5
…		
186	2	2,4075e-5
187	1	4,299e-5
188	2	3,5755e-4
189	1	1,4084e-5
190	1	7,5545e-5
191	2	3,4001e-5
192	2	2,7871e-5

3. *Алгоритмы, выбранные для методов альтернативной адаптации одинаково хороши.*

В таблице 6 приведены результаты работы метода альтернативной адаптации с линейной тактикой. Аналогичным образом ведет себя и метод альтернативной адаптации с обучением. В обоих случаях генетические алгоритмы сменяют друг друга редко и надолго, что не только является характерным поведением, но и не позволяет предпочесть их друг другу. Этот факт может использовать для выделения наиболее эффективных алгоритмов с целью их более тщательного исследования для данной задачи.

Таблица 6

Результаты работы метода альтернативной адаптации с линейной тактикой

№ прогона	№ альтернативы	Результат
1	1	5,6152e-7
2	2	1,8103e-3
3	2	5,6152e-7
4	2	5,6152e-7
5	2	5,6152e-7
…		
15	1	5,6152e-7
16	1	5,6152e-7
17	1	5,6152e-7
18	1	5,6152e-7

Выводы

При использовании методов альтернативной адаптации требуется значительно меньшее количество прогонов алгоритмов, для определения наиболее эффективного из них. Используя методы альтернативной адаптации и наблюдения полученные в ходе выполненной работы, можно эффективно и быстро выбирать наилучший алгоритм из числа предложенных. Для этого достаточно сравнивать эволюционные алгоритмы, значительно отличающиеся по своему характеру. Например, одноточечное скрещивание, пропорциональная селекция и низкая мутация придают алгоритму квазилокальный характер, а равномерное скрещивание, турнирная селекция и высокая мутация приводят к глобальному поведению алгоритма и слабой локальной сходимости. Маловероятно, что на какой-либо задаче они будут одинаково пригодны, следовательно альтернативная адаптация покажет лучший из них после нескольких прогонов (вместо нескольких десятков) или будет установлено, что оба они малоэффективны.

Приведенные идеи и результаты их экспериментальной проверки могут лечь в основу методики ускоренного тестирования эволюционных алгоритмов, что имеет существенное значение для их практического применения.

АДАПТИВНЫЕ СТОХАСТИЧЕСКИЕ АЛГОРИТМЫ ОПТИМИЗАЦИИ

Алгоритмы локального поиска на дискретных и комбинаторных структурах

Общие идеи локального поиска

Локальный поиск базируется на старейшем в истории человечества методе оптимизации - методе проб и ошибок. Эта идея столь проста и естественна, что просто удивительно насколько успешным оказывается локальный поиск при решении многих задач оптимизации.

Главной идеей локального поиска является концепция соседства, выражаемая понятием "окрестность". Прежде всего надо дать математическую формализацию концепции окрестности.

Пусть мы имеем допустимую точку X Î D для некоторой конкретной задачи оптимизации. Было бы весьма полезно определить некоторое множество точек "близких", в некотором смысле, к этой точке X.

Определение 1. Пусть задана задача оптимизации (D, f), где D - допустимая область, f - целевая функция. Тогда системой окрестностей или окрестностной функцией называется отображение $N : D \rightarrow 2D$, определенное для каждой индивидуальной задачи (2D есть множество всех подмножеств множества D).

Если D - это обычное евклидово пространство, то множество точек, находящихся ближе определенного расстояния в евклидовой метрике от заданной точки, являются "естественной" окрестностью. Например, в алгоритме Нелдера-Мида окрестностью текущей точки являются вершины деформируемого многогранника данного этапа, включая полученную отражением (растяжением или сжатием).

Однако во многих задачах, особенно в дискретной и комбинаторной оптимизации, выбор системы окрестностей совсем не очевиден и существенно зависит от структуры допустимого множества D. Например, для задачи коммивояжера система окрестностей определяется как 2-замена:

N2(X) = {Y: Y∈D и Y может быть получен из X путем удаления двух дуг и их заменой двумя другими}

Например - первый маршрут - {1 2 3 4 5 6 7}, а второй - {1 2 4 3 5 6 7}.

Пример 3-замены: {1 2 3 4 5 6 7} → {1 2 3 5 4 7 6}.

Очевидным обобщением этих систем окрестностей является так называемая k-замена N*k*, в которой заменяются k дуг. Такие системы окрестностей могут давать очень эффективные результаты для задачи коммивояжера.

Для задачи оптимизации на дискретной решетке могут быть рассмотрены следующие системы окрестностей:

$$N1(X) = \left\{ Y \in D : \sum_{i=1}^{n} |x_i - y_i| \le 1 \right\},$$

$$N2(X) = \left\{ Y \in D : \sum_{i=1}^{n} |x_i - y_i| \le 2 \right\},$$

$$N3(X) = \left\{ Y \in D : |x_i - y_i| \le 1 \; \forall i = \overline{1,n} \right\},$$

Эти системы окрестностей порождены обычной евклидовой метрикой.

Теперь мы должны дать определение точек локального и глобального минимума для дискретных и комбинаторных задач оптимизации.

Определение 2. Пусть дана индивидуальная задача оптимизации (D,f) и система окрестностей N. Тогда некоторое допустимое решение X* называется локально оптимальным или оптимальным относительно системы окрестностей N, если $f(X^*) < f(Y) \; \forall \; Y \in N(X^*)$.

Определение 3. Решение X задачи (D, f) называется глобально оптимальным, если $f(X) < f(Y) \; \forall Y \in D$.

Определение 4. Если любая точка $X \in D$, локально оптимальная относительно системы окрестностей N, является глобально оптимальной, то система окрестностей N называется точной.

В частности для задачи коммивояжера точной будет система окрестностей, порожденная n-заменой, а для задачи оптимизации с булевыми переменными точной будет система окрестностей N3(X).

Опишем теперь обобщенный алгоритм локального поиска.

Задана индивидуальная задача оптимизации (D, f), где D - это допустимая область, а f - целевая функция.

Выберем систему окрестностей N, по которой будем пытаться улучшить решение в точке X с помощью процедуры "УЛУЧШЕНИЕ":

$$\text{УЛУЧШЕНИЕ}(X) = \begin{cases} \text{любому } Y \in N(X), \text{где } f(Y) < f(X), \text{если } Y \text{ существует,} \\ \text{"NO", если такой } Y \text{ не существует.} \end{cases}$$

Итерационная процедура локального поиска может быть описана следующим образом.

1. Выбрать некоторую начальную допустимую точку X.

2. Выполнить процедуру УЛУЧШЕНИЕ(X).

3. Если УЛУЧШЕНИЕ(X) = "NO", то остановиться. Точка X - точка минимума.

4. Положить X = УЛУЧШЕНИЕ(X) и идти к 2.

Этот алгоритм начинает работу из произвольной начальной допустимой точки и использует процедуру УЛУЧШЕНИЕ(X) для поиска лучшего решения в окрестности точки X. Если удалось найти лучшее решение, то оно запоминается и поиск продолжается по окрестности этого лучшего решения. Если лучшего решения найти не удалось, то текущая точка и есть локальный минимум.

Нетрудно видеть, что при применении описанного алгоритма приходится несколько раз делать выбор.

Первой проблемой является выбор начальной допустимой точки. На практике бывает полезно выполнить локальный поиск из нескольких начальных точек (мультистарт) и выбрать лучший результат. В таком случае прийдется решать сколько начальных точек выбирать и как распределить их в пространстве.

Второй проблемой является выбор "хорошей" системы окрестностей и метода поиска в этой системе. В настоящее время этот выбор осуществляется на основе человеческой интуиции, а теоретических рекомендаций не имеется. Однако, существует конкуренция между малыми и большими (по количеству содержащихся в них точек) окрестностями. Окрестности больших размеров могут дать лучший локальный минимум, но это займет больше времени, а значит будет найдено меньшее количество локальных минимумов в течение фиксированного временного интервала. Стоит вопрос: что мы должны делать? Сгенерировать больше малых окрестностей или меньше больших? Такого рода вопросы обычно имеют эмпирический ответ, а значит создание эффективных алгоритмов локального поиска было и остается более похожим на искусство, а не на науку.

Третье, что нужно выбрать - это порядок просмотра окрестности. Для этого необходимо в первую очередь упорядочить окрестность. Иногда это легко сделать, например в случае оптимизации на дискретной решетке, где порядок естественным образом порождается нумерацией координат, а иногда упорядочение не совсем очевидно как, например, в задаче

коммивояжера. Для оптимизации на перестановках и некоторых других комбинаторных структурах бывает полезным лексикографическое упорядочение. В сложных случаях может быть применено случайное упорядочение элементов окрестности. После упорядочения точек окрестности возникает задача выбора тактики просмотра окрестности. В основном различают тактику перехода по первому улучшению и наискорейший спуск, который состоит в том, что каждая текущая окрестность просматривается полностью и переход осуществляется в самую лучшую точку этой окрестности. Опыт показал, что лишние затраты на наискорейший спуск редко бывают оправданы и для применения этой тактики нужны серьезные основания. В случае перехода по первому улучшению появляется еще одна проблема - это выбор способа продолжения просмотра окрестности после перехода в новую точку. Возможны такие варианты как продолжение просмотра с начала (т.е. в новой окрестности первой просматривается точка с номером один, затем - два, и т.д.), продолжение просмотра с точки со следующим номером и продолжение просмотра с точки с тем же номером. В последнем случае возможно вырождение алгоритма в покоординатный спуск или зацикливание.

Так как локальный поиск может найти глобальный минимум только случайно (если стартовая точка лежит в зоне притяжения глобального минимума или целевая функция унимодальна), то разрабатываются всевозможные способы усиления глобальной сходимости локальных схем. Наиболее часто применяется так называемый мультистарт, когда по некоторому правилу генерируются стартовые точки, из которых затем выполняется локальный спуск, после чего лучший локальный минимум объявляется глобальным. Для уменьшения количества спусков в одну и ту же зону притяжения может быть использован запрет на приближение к уже найденным локальным минимумам (типа туннельных функций или списка Табу), который вводится дополнительным ограничением в постановку задачи или присутствует в виде динамического списка запрещенных направлений в процессе оптимизации. Существуют также статистические методы усиления глобальной сходимости - кластерный метод Торна, когда после нескольких шагов спуска из каждой стартовой точки выполняется кластер-анализ и в обнаруженных сгустках точек (предполагая, что они соответствуют траекториям, ведущим в один и тот же локальный минимум) отбрасываются все, кроме наилучшей, а также метод обнаружения закономерностей, когда уже полученные локальные

минимумы анализируются на выявление закономерности, после чего эта закономерность вводится в условия задачи в качестве дополнительного ограничения. Отметим, что статистические методы влекут существенные "накладные" расходы на оптимизацию и должны быть использованы только в случаях, когда это оправдывается резким снижением затрат на вычисление целевой функции.

Локальный поиск для оптимизации псевдобулевых функций

Задачами псевдобулевой оптимизации называются задачи оптимизации вещественных функций булевых переменных, т.е. такие задачи, в которых аргументы целевой функции могут быть только нулем или единицей, а значение функции - вещественное число.

Введем структуру на множестве B_n булевых переменных следующим образом.

Определение 5. Точку X булева гиперкуба назовем k-соседней к точке Y, если она отличается значениями k своих координат.

Определение 6. Множество $O_k(X)$ всех k-соседних к X точек назовем k-м уровнем точки X.

Лемма 1. Пространство B_n булевых переменных может быть представлено как объединение не пересекающихся подмножеств, являющихся уровнями выбранной точки начала структуры, а именно

$$B_n = \bigcup_{k=0}^{n} O_k(X^0), \ \forall\ X^0 \in B_n.$$

Доказательство следует с очевидностью из того, что уровней каждой точки действительно может быть ровно n+1, если считать саму точку начала структуры за ее нулевой уровень, и из того, что уровни с различными номерами не могут пересекаться.

Определение 7. Множество $\{X^0, X^1, \dots, X^k\}$ назовем путем длины k в булевом гиперкубе, если $X^i \in O_1(X^{i+1})$ $\forall i = 0, 1, \dots, k$.

Определение 8. Путь $\{X^0, X^1, \dots, X^k\}$ назовем путем убывания, если $f(X^{i+1}) < f(X^i)$ $\forall i = 0, 1, \dots, k$.

На основе введенной таким образом на B_n структуры были построены довольно сложные алгоритмы локального спуска, в том числе неулучшаемые алгоритмы для монотонных и локально-монотонных (полимодальных) псевдобулевых функций, а также

универсальный алгоритм локального спуска для произвольной псевдобулевой функции.

Обсудим введенную структуру с точки зрения общей теории локального поиска и выясним степень "универсальности" алгоритмов, работающих на основе данного подхода.

Прежде всего отметим, что упомянутые алгоритмы являются алгоритмами прямого поиска, реализующими идею локального спуска по некоторой частной системе окрестностей. Действительно, алгоритмы принимают решение о переносе поиска в новую точку после просмотра первого уровня начальной точки, что в соответствии с общей теорией локального поиска равнозначно поиску по системе окрестностей, представляющей собой набор первых уровней текущих точек поиска. Очевидно, что эта система окрестностей является самой слабой в том смысле, что трудно предложить какую-либо другую систему, содержащую меньшее количество точек и все еще согласующуюся со здравым смыслом. Например, формально можно ввести "окрестность" типа "все точки отличающиеся седьмой и десятой компонентой", содержащую только две "соседние" точки, но это выглядит не очень убедительно. Поэтому среди "разумных" определений окрестностей, порождаемых метрикой Хэмминга, т.е. индуцированных естественной Евклидовой метрикой, указанная система окрестностей является самой слабой и содержит самое малое количество соседних точек. В связи с этим возникает стандартная проблема не убедительной обоснованности выбора системы окрестностей. Достаточно указать, что найденный по данной системе окрестностей локальный минимум может лежать буквально "рядом" с точкой, дающей намного лучшее значение функции, если только эта точка отличается от полученного решения двумя компонентами и не была случайно просмотрена при продвижении алгоритма к указанному "минимуму".

Описанные алгоритмы получили название "неулучшаемых" по быстродействию алгоритмов в том смысле, что алгоритмы реализуют информационную сложность класса. Эти алгоритмы указывают локальный минимум из любой начальной точки после (n+1)-го вычисления целевой функции, то есть затраты адекватны просмотру только одной окрестности, как если бы мы всегда стартовали из самой точки локального минимума, и это действительно неулучшаемый результат. Правда эти алгоритмы работают только на монотонных псевдобулевых функциях и их ответ для произвольной функции будет совершенной бессмысленным. Для произвольных псевдобулевых функций был

предложен обычный алгоритм локального спуска по этой же системе окрестностей, реализующий идею наискорейшего спуска, т.е. полного просмотра окрестности и перехода в лучшую из найденных точек. Показано существование (специальным образом построенной) унимодальной псевдобулевой функции, при оптимизации которой данный алгоритм выполнит полный перебор. Т.е. даже выбор самой слабой системы окрестностей не исключает полного перебора при оптимизации даже унимодальной (по данной системе окрестностей!) функции.

В этой связи необходимо обсудить выбор других тактик оптимизации и других систем окрестностей, с тем, чтобы обеспечить более "качественный" локальный минимум и меньшие затраты на произвольной функции. Кроме того неполохо построить алгоритм "почти столь же" эффективный на монотонных функциях, как неулучшаемый, и, в то же время, работоспособный на произвольных псевдобулевых функциях.

Очевидной альтернативой наискорейшему спуску является переход по первому улучшению, который и должен быть применен в условиях, когда отсутствует специфическая информация, дающая повод надеяться, что "наискорейший" спуск будет действительно наискорейшим, а заодно позволит исключить полный перебор в худшем случая для любой псевдобулевой функции, так как не будет всегда просматривать окрестность полностью, а сделает это лишь однажды в конце поиска. Кроме того, для исключения полного перебора может быть использован "раздельный старт" локального спуска, что близко по реализации к мультистарту в глобальной оптимизации.

Относительно построения других систем окрестностей можно предложить очевидные обобщения типа просмотра 2-соседних точек вместо 1-соседних как в указанных выше алгоритмах, что приводит с одной стороны к увеличению затрат до $O(n^2)$ при полном просмотре окрестности, а с другой полностью исключает полный перебор для любой унимодальной функции. Аналогичными свойствами обладают все системы окрестностей с четными k, хотя их мощности резко возрастают без видимых преимуществ в качестве получаемого решения (4-соседняя точка может быть получена как 2-соседняя к 2-соседней при увеличении количества вычислений целевой функции в два раза, но при уменьшении мощности окрестности в $O(n^2)$ раз).

При построении алгоритмов учитывались следующие факты.

Определение 9. Точка $X^* \in B_n$ называется точкой локального минимума псевдобулевой функции f, если $f(X^*) < f(X)$ $\forall X \in O_1(X^*)$.

Определение 10. Псевдобулева функция f называется унимодальной, если для любой точки X булева гиперкуба существует путь убывания $\{X, X^*\}$, где X^* - точка минимума.

Определение 11. Унимодальная псевдобулева функция f называется монотонной, если в любой точке $X \in O_k(X^*)$ выполняется $f(Y) \geq f(X)$ $\forall Y \in O_1(X) \cap O_{k+1}(X^*)$.

Лемма 2. Пусть f - унимодальная строго монотонная псевдобулева функция, X^* - точка минимума, $X \in O_k(X^*)$ - произвольная точка булева гиперкуба и $X^i \in O_1(X)$, $i = 1,\ldots, n$. Тогда

$$x_i^* = \begin{cases} x_i, & f(X) < f(X^i), \\ 1 - x_i, & f(X) > f(X^i), \end{cases} \quad i = 1,2,\ldots,n.$$

Из приведенных определений и леммы легко получить следующий

Алгоритм 1.

1. Выбираем $X^0 \in B_n$ произвольно.

2. Последовательной заменой значений координат точки X^0 на противоположные определяем все ее 1-соседние точки X^i, $i = 1, 2, \ldots, n$.

3. Вычисляем значения $f(X^0)$ и $f(X^i)$, $i = 1, 2, \ldots, n$.

4. Координаты точки X^* определяем по правилу:

$$x_i^* = \begin{cases} x_i^0, & f(X^0) < f(X^i), \\ 1 - x_i^0, & f(X^0) > f(X^i), \end{cases} \quad i = 1,2,\ldots,n.$$

Основная идея алгоритма состоит в том, что из леммы 2 легко узнать уровень точки, если мы знаем какие из ее 1-соседних точек лежат ближе к локальному минимуму, а какие - дальше, что не трудно сделать в случае монотонной функции по ее значениями в соответствующих точках. Кроме номера уровня легко определяются и те компоненты, которыми текущая точка отличается от точки минимума - в этих точках значения функции меньше.

Теорема 1. Оптимизация произвольной монотонной псевдобулевой функции алгоритмом 1 требует вычисления значений функции в (n+1) точках булева гиперкуба.

Нетрудно видеть, что при практической реализации алгоритма его реальная трудоемкость будет большей. Действительно, при решении практической задачи трудно ожидать, что будет заранее известно, является ли функция монотонной, а в случае немонотонности целевой функции точка, выдаваемая алгоритмом 1, может быть сколь угодно далека от точки минимума. Поэтому для нужд практики алгоритм 1 должен быть модифицирован добавлением шага верификации, который состоит в том, что вычисляются значения функции в предполагаемой точке минимума и во всех ее 1-соседних точках и, если при этом будет обнаружено, что найденная точка дает значение функции меньшее, чем все точки в ее окрестности, то можно утверждать, что функция монотонная, а найденная точка действительно есть точка минимума. При отрицательном ответе на шаге верификации можно лишь утверждать, что оптимизируемая функция не является монотонной, т.е. алгоритм 1 был применен зря, а процесс оптимизации должен быть выполнен снова. Трудоемкость модифицированного алгоритма будет равна (2n+2) вычисления целевой функции, что все равно является очень хорошей оценкой. Вопрос о том, как часто на практике будут встречаться монотонные функции остается открытым. Теоретически же доля монотонных псевдобулевых функций среди всех возможных ничтожно мала.

Рассмотрим алгоритм локального спуска для оптимизации произвольных псевдобулевых функций.

Алгоритм 2.

1. Положить r = 0, выбрать точку $X^r \in B_n$ произвольно и вычислить $f(X^r)$.

2. Последовательной заменой компонент вектора X^r на противоположные определить все точки $X^i \in O_1(X^r)$, $i = 1, 2, \ldots, n$.

3. Для всех $X^i \in O_1(X^r) \backslash O_1(X^{r-2})$ (при $r<0$ предполагается $O_1(X^r)=\varnothing$) вычислить $f(X^i)$. Если $f(X^r) < f(X^i)\ \forall i = 1, 2, \ldots, n$, то $X^* = X^r$, иначе - идти к 4.

4. Положить r = r + 1, определить точку X^r из условия

$$f(X^r) = \min_{X^i \in O_1(X^{r-1})} f(X^i)$$

и идти к 2.

Как нетрудно видеть этот алгоритм реализует наискорейший спуск по системе окрестностей $O_1(X^r)$, т.е. полный просмотр всех 1-соседних точек каждой текущей точки с переходом в ту из них, которая дает наилучшее значение целевой функции.

Теорема 2. При оптимизации монотонных и слабо немонотонных унимодальных псевдобулевых функций алгоритм 2 требует в среднем $(n^2+4)/2 - 1/2^n$ вычислений целевой (усреднение по выбору начальной точки).

Теорема 3. При оптимизации произвольных унимодальных псевдобулевых функций алгоритм 2 требует в среднем не более, чем

$$T = 2^n - (n^2+3n-2)/2 + (2n^3+n^2-4n+3)/2^n$$

вычислений целевой функции при $n>4$.

Как видно из теорем, трудоемкость алгоритма 2 велика для монотонных функций (по сравнению с алгоритмом 1) и сравнима с полным перебором для произвольных функций в худшем случае. Поэтому и возникает задача модернизации приведенного подхода с целью разработки более эффективных алгоритмов.

Для решения практических задач, сводимых к безусловной оптимизации унимодальных псевдобулевых функций необходимы алгоритмы с более приемлемой оценкой, чем даваемая в теореме 3.

Прежде всего необходимо выбрать такую систему окрестностей, чтобы исключить полный перебор в самом худшем случае. Слишком высокая трудоемкость алгоритма 2 на произвольной унимодальной функции объясняется тем, что в сложных случаях приходится двигаться по замысловатому пути, "наматывая" при этом на траекторию поиска почти все точки гиперкуба именно за счет того, что в окрестность входят 1-соседние точки, т.е. потенциально все точки могут быть просмотрены как соседние к соседним. Этот недостаток системы окрестностей усугубляется тактикой наискорейшего спуска, применение которой ведет к тому, что в каждой точке поиска будут просмотрены все соседние точки, в том числе дающие худшее значение целевой функции. Сочетание этих недостатков ведет к тому, что при недостаточно удачном выборе начальной точки алгоритм 2 выполняет полный перебор. Если же траектория поиска была достаточно "удачной" и затраты оказались небольшими, то всегда остается сомнение, а не лежит ли где-нибудь совсем рядом с найденной точкой локального минимума точка, дающая еще лучшее значение, так как данная система окрестностей - слабейшая из возможных.

Отсюда следует, что для модифицированного алгоритма необходимо подобрать более сильную систему окрестностей, в то же время исключающую полный перебор в худшем случае. Этому довольно-таки противоречивому требованию полностью удовлетворяет система окрестностей типа $O_1(X^0) \subset O_2(X^0)$, правда при "правильной" тактике просмотра текущей окрестности. Действительно, если начинать проверку с точек $O_1(X^0)$ и выбрать наискорейший спуск, то изменение будет несущественным - после нахождения локального минимума по системе $O_1(X^0)$ будет просмотрена еще и $O_2(X^0)$, т.е. затраты просто возрастут, хотя и качество решения усилится.

Поэтому лучше использовать переход по первому улучшению и начинать просмотр окрестности с точек второго уровня. Сокращение перебора будет обеспечено за счет того, что все точки нечетных по отношению к стартовой точке уровней не будут просматриваться до тех пор, пока не будет "накрыта" точка минимума, после чего будет произведен просмотр первого уровня (единственный раз!). Точкой минимума является либо текущий центр окрестности, либо его 1-соседняя точка, что и будет установлено. Кроме того, существенное сокращение перебора обеспечит и переход по первому улучшению, который исключает полный просмотр текущей окрестности на тех стадиях, когда поиск еще далек от точки локального минимума. Конечно же возможны и "фатальные" случаи, когда из текущей окрестности есть только "узкий проход" к точке локального минимума, состоящий только из одной 2-соседней точки, и в этих случаях даже при переходе по первому улучшению будут просматриваться все 2-соседние точки. Но первое условие, обеспечивающее сокращение перебора, остается в силе и для этого "фатального" случая.

Сказанное позволяет записать следующий алгоритм оптимизации произвольных унимодальных псевдобулевых функций, реализующий локальный спуск по системе окрестностей $O_2(X^0) \cup O_1(X^0)$ с переходом по первому улучшению.

Алгоритм 3.

1. Выбрать начальную точку X^0 произвольно.

2. Определять точки $X^j \in O_2(X^0)$, $j = 1, 2, \ldots, C_n^2$, и вычислять $f(X^j)$ до тех пор, пока не будет найдена такая точка X^j, что $f(X^j) < f(X^0)$.

3. Если такая точка найдена, то положить $X^0 = Xj$ и идти к шагу 2.

4. Среди точек $X^k \in O_1(X^0)$ выбрать точку X^*, дающую наименьшее значение функции.

5. Если $f(X^*)<f(X^0)$, то X^*- точка минимума, иначе это - X^0.

6. Вывести точку минимума и значение функции. Остановиться.

Теорема 4. Если стартовая точка лежит на k-м уровне точки минимума, то при оптимизации монотонной унимодальной псевдобулевой функции алгоритм 3 сделает не более, чем $(n^2-n-k^2+k+1)/2$ вычислений значений функции.

Доказательство. В худшем случае алгоритм 3 должен сначала перебрать все точки (k+2)-го и k-го уровня точки минимума, являющиеся 2-соседними к стартовой точки. Таких точек $„^2_{n-k}$ и

$k\cdot (n-k)$ соответственно. Далее остаются только комбинации пар координат, которые будут давать улучшение, т.е. точки лежащие на (k-2)-м уровне. В первую же из них будет перенесен центр окрестности. Так как очевидно, что нет никакого смысла повторять все комбинации пар сначала, то будет выбрана следующая пара координат, которая опять-таки даст улучшение в силу монотонности функции. Таким образом алгоритм будет продвигаться к точке минимума шагами длины 2, т.е. сделает k/2 (для четного k) шагов до "накрытия" точки минимума, после чего будет сделано еще $Ñ^2_n+n$ (просмотр всей окрестности) вычислений с тем, чтобы убедиться, что найденная точка и есть точка минимума. Итак, имеем $„^2_{n-k}+ k\cdot (n-k)+k/2+ Ñ^2_n+n$ вычислений. После преобразований получим утверждение теоремы.

Теорема 5. При оптимизации произвольной унимодальной монотонной псевдобулевой функции алгоритм 3 требует в среднем не более, чем $3(n^2-n)/8+1/2$ вычислений целевой функции (усреднение по выбору стартовой точки).

Доказательство. Вероятность попасть на k-й уровень при случайном выборе стартовой точки равна $Ñ^k_n/2^n$. В этом случае среднее количество вычислений определяется из выражения

$$T = \frac{1}{2}\sum_{k=0}^{n}\left[n^2-n-k^2+k+1\right]\cdot\frac{C_n^k}{2^n} =$$

$$= \frac{1}{2^{n+1}}\left[n\cdot(n-1)\cdot\sum_{k=0}^{n}C_n^k-\sum_{k=0}^{n}k^2\cdot C_n^k+\sum_{k=0}^{n}k\cdot C_n^k+\sum_{k=0}^{n}C_n^k\right] =$$

$= \frac{1}{2^{n+1}}[n\cdot(n-1)\cdot 2^n - n\cdot(n+1)\cdot 2^{n-2} + n\cdot 2^{n-1} + 2^n] = 3(n^2-n)/8+1/2.$ ²

Нетрудно видеть, что в случае монотонных функций трудоемкость алгоритма 3 ниже трудоемкости алгоритма 2, но все-таки остается квадратичной в то время как трудоемкость неулучшаемого алгоритма 1 линейна.

Оценкой сверху для алгоритма 2 является полный перебор, причем эта оценка - достижимая. Верхней оценкой алгоритма 3 является половина полного перебора и эта оценка также теоретически достижима, если стартовая точка будет выбрана столь неудачно, что путь убывания, ведущий к точке минимума будет единственным и его траектория "намотает" все точки четных уровней. Т.е. половина полного перебора для алгоритма 2 - случай весьма экзотический, который может быть специально организован, но практически не встречается. В то же время для алгоритма 2 трудоемкость, близкая к полному перебору, является рядовым событием (см. оценку в среднем из теоремы 3). В принципе можно было бы сделать вывод о преимуществе алгоритма 3 перед ранее предложенными алгоритмами локального спуска для произвольных унимодальных функций. Однако алгоритм 3 можно усилить, если применить гибридную схему с переключением тактик поведения в зависимости от установленных свойств целевой функции. Кроме того, можно гарантировать исключение наихудшего случая, когда алгоритм выполняет половину полного перебора, применив "раздельный" старт, т.е. запуская процесс оптимизации из нескольких точек и выбирая наилучшую из них. Даже если среди стартовых точек окажется та самая "наихудшая", заставляющая сделать половину перебора, то, выбирая наилучшую из всех стартовых точек, мы исключим не только наихудшую точку, но и все точки траектории, соединяющей наихудшую точку с наилучшей, а заодно и все точки их окрестностей, которые были бы просмотрены при старте из наихудшей точки по наихудшей траектории. Получаемый алгоритм напоминает мультистарт при глобальной оптимизации с той разницей, что из выбранных точек не будет выполняться локальный спуск (за исключением, конечно, наилучшей из них). Пошаговая процедура, реализующая описанную схему, приведена ниже как

Алгоритм 4.

1. Выбрать произвольным образом стартовую точку X^1 и определить ее (n-1)-соседние точки X^2, X^3, ... , X^n, вычислить значения функции в этих точках.

2. Определить точки $Y^j = (x_1^j, x_2^j, ..., x_{j-1}^j, 1-x_j^j, x_{j+1}^j, ..., x_n^j)$, j = 1, 2, …, n, и вычислить значения функции в этих точках.

3. Определить точку X^* по правилу

$$x_j^* = \begin{cases} x_j^j, & f(X^j) < f(Y^j), \\ 1 - x_j^j, & f(X^j) > f(Y^j), \end{cases} \quad j = 1,2,...,n,$$

и вычислить значение функции в ней.

4. Если $f(X^*)<f(X^j)$ $\forall$ j = 1, 2, …, n и $f(X^*)<f(Y^j)$ $\forall$j = 1, 2, …, n, то определить все точки $Z^i \in O_1(X^*)$ и вычислить значения функции в них. Иначе - идти к шагу 6.

5. Если $f(X^*)<f(Z^i)$ $\forall i$ = 1, 2, …, n, то вывести X^*, $f(X^*)$ и остановиться (X^* - точка минимума, функция f - унимодальная и монотонная).

6. Выбрать точку X^0 из условия

$$f(X^0) = \min \{f(X^*), f(X^j), f(Y^j), f(Z^j), \forall j = 1, 2, ..., n\}$$

и выполнить локальный спуск в соответствии с шагами 2-6 алгоритма 3.

Теорема 6. При оптимизации произвольной монотонной унимодальной псевдобулевой функции алгоритмом 4 потребуется ровно 3·n+1 вычислений значений этой функции.

Доказательство. Если функция монотонна, то шаг 6 алгоритма не понадобится и вычисления ограничатся точками X^j, Y^j, Z^j, j = 1, 2, …, n, и X^*, которых в точности 3·n+1.

Оценку сверху на произвольной унимодальной функции довольно трудно рассчитать точно, но для сравнения с ранее предложенными алгоритмами достаточно указать, что точки X^j находятся на (n-1)-м уровне стартовой точки и если бы эта точка была выбрана столь неудачно, чтобы реализовался самый худший случай, то в результате "раздельного" старта удается исключить просмотр всех точек, входящих в окрестности тех точек, которые лежали бы на траектории, соединяющей стартовую точку и X^j. При нечетном n таких точек будет $\tilde{N}_n^2 \cdot$ (n-1)/2 т.е. порядка $n^3/4$. Это и есть минимальный выигрыш от раздельного старта. Повторим, что "наихудший" случай практически никогда не реализуется, а в остальных случаях трудоемкость алгоритма 4 будет существенно ниже, вплоть до линейной.

Таким образом в данном параграфе предложен алгоритм 4, реализующий локальный спуск с раздельным стартом в системе окрестностей $O_1(X^0) \cup O_2(X^0)$ и переход по первому улучшению с

просмотром окрестности начиная с точек из $O_2(X^0)$ и имеющий трудоемкость почти такую же как неулучшаемый алгоритм на монотонных функциях и существенно (в 2 раза минимум) лучшую трудоемкость, чем обычный локальный спуск, на произвольных унимодальных функциях. Этот алгоритм, сочетающий различные системы окрестностей и тактики просмотра и перехода, и называется обобщенным алгоритмом локального поиска для псевдобулевой оптимизации.

Однако на практике редко встречаются задачи без ограничений на переменные. Отсутствует обычно и информация о количестве локальных минимумов. Это означает, что необходимо предложить процедуру оптимизации псевдобулевых функций, учитывающую наличие ограничений и возможность появления нескольких локальных минимумов. Характерной особенностью задач дискретной (в частности псевдобулевой) оптимизации является тот факт, что даже унимодальная сама по себе функция становится многоэкстремальной при введении даже простых (например, линейных) ограничений. Именно поэтому проблемы глобальной и условной оптимизации лучше рассматривать вместе.

Для решения задач условной и глобальной оптимизации псевдобулевых функций использовались два подхода. Первый из них предполагал наличие "хороших" свойств у функций ограничений для задач условной оптимизации, либо правильной структуры расположения локальных минимумов. Второй подход сводился к использованию методов случайного поиска для организации поиска в допустимой области. Несомненным достоинством этих подходов является то, что они позволяют предложить рациональный результат в тех случаях, когда другие методы не работают. Однако задачи условной и глобальной оптимизации в общем случае ими не решаются. Ниже предлагаются алгоритмы, которые работоспособны в общем случае.

В качестве общей схемы для условной и безусловной глобальной оптимизации предлагается использовать мультистарт с последующим локальным спуском из каждой стартовой точки с соответствующими процедурами, сокращающими перебор и обеспечивающими получение допустимого решения. Преимуществом перед случайным поиском является главным образом рациональный характер алгоритмов и гарантия того, что при сопоставимом количестве просмотренных точек будет получен истинный локальный минимум, причем лучший из известных, в то время как случайный поиск выдает практически произвольную

точку, которая может не быть даже локальным минимумом. Случайный поиск не дает оценок значений функции в других локальных минимумах, и кроме того, при больших размерностях вырождается в случайное блуждание из-за невозможности адаптировать вероятности выбора компонент. Для учета ограничений предлагается ввести штрафную функцию для предотвращения выхода за границу допустимой области, так как доказано. что при достаточно больших коэффициентах штрафа получаемая точка будет обязательно допустимой.

Пусть задана целевая псевдобулева функция f и функции-ограничения $g_j(X) \le 0$, $j = 1, 2, \ldots, m_1$, $h_i(X) = 0$, $i = 1, 2, \ldots, m_2$.

Для решения задачи условной оптимизации составляем обобщенную функцию со штрафом

$$F(X) = f(X) + r \cdot \sum_{j=1}^{m_1} g_j(X) + \rho \cdot \sum_{i=1}^{m_2} h_i^2(X)$$

и минимизируем ее алгоритмом локального спуска с мультистартом, обеспечивая допустимый результат. Лучший из найденных локальных (условных) минимумов принимаем за ответ. Параметрами алгоритма являются коэффициенты штрафа r и ❒, а также количество точек мультистарта. Данные параметры должны выбираться в зависимости от задачи с тем, чтобы обеспечить допустимое решение в рамках имеющихся вычислительных ресурсов.

Кроме штрафных функций возможен также алгоритм с прямой проверкой точек на допустимость по системе ограничений. Такая модификация алгоритмов безусловной оптимизации непрерывных функций (метод конфигураций, метод деформируемого многогранника, метод сопряженных направлений) является неэффективной из-за невозможности двигаться вдоль границы. Однако в случае дискретной или псевдобулевой оптимизации локальный спуск с мультистартом может быть эффективным и при таком подходе.

Мультистарт локального поиска существенно зависит от выбора стартовых точек. Обычно генерирование точек, из которых затем запускается локальный спуск, осуществляется методом Монте-Карло, т.е. практически случайным набросом какого-то количества точек на допустимую область. Хотя для поисковых алгоритмов чрезмерно большое количество вычислений целевой функции является привычным делом, тем не менее обычная схема мультистарта использует эти вычисления слишком нерационально.

Подавляющее количество вычислений приходится на спуск в уже известные или худшие, чем уже известные, локальные минимумы.

В связи с вышеизложенным возникает идея использовать для генерации стартовых точек локального спуска алгоритмы адаптивного поиска (см. раздел 7). Такие гибридные схемы обладают тем очевидным преимуществом, что их теоретические гарантии сходимости во всяком случае не хуже, чем у соответствующего базового алгоритма (имитация отжига или эволюционные подходы) и, кроме того, все вычислительные затраты будут направлены на сам процесс оптимизации.

Локальный поиск для оптимизации на дискретной решетке

Прежде всего сформулируем постановку задачи оптимизации на конечной дискретной решетке. Для данных задач характерны два типа ограничений на переменные - так называемые "естественные" и существенные. Естественные ограничения задают пределы изменения переменных, а существенные задаются функциями-ограничениями или другими процедурами для проверки допустимости. Математически задача может быть записана следующим образом.

$$f(X) \rightarrow \min, \quad (1)$$

$$l_i \le x_i \le u_i,\ i = 1, \ldots, n, \quad (2)$$

$$g_j(X) \le 0,\ j = 1, \ldots, m_1, \quad (3)$$

$$h_k(X) = 0,\ k = 1, \ldots, m_2 \quad (4)$$

В задаче (1)-(4) ограничения (2) являются естественными, а ограничения (3)-(4) - существенные. Под задачей безусловной оптимизации на конечной дискретной решетке будем понимать задачу

$$f(X) \rightarrow \min, \quad (5)$$

$$l_i \le x_i \le u_i,\ i = 1, \ldots, n, \quad (6)$$

т.е. задачу без существенных ограничений. Частным случаем данной задачи, когда $l_i = 0 \wedge u_i = k\text{-}1 \ \forall\ i = 1, \ldots, n$, является так называемая задача оптимизации для k-значных логик.

Утверждение 1. Пространство оптимизации D_n задачи (5)-(6) содержит $N = \prod_{i=1}^{n} |u_i - l_i|$ точек, в частности для k-значных логик - k^n, а для псевдобулевой оптимизации - 2^n.

Утверждение 2. Множество $O_1(X)$ всех точек, отличающихся от X значениями только одной координаты (единичная окрестность, первый уровень), содержит $2 \cdot n - l(X) - u(X)$ точек, где

$l(X)$ и $u(X)$ - это количество координат точки X, равное l_i и u_i, соответственно.

Следствие 1. $n \le \text{card } O_1(X) \le 2 \cdot n$.

Лемма 3. Пространство D_n может быть представлено как объединение не пересекающихся подмножеств, являющихся уровнями выбранной точки начала структуры, а именно

$$D_n = \bigcup_{k=0}^{N(X)} O_k(X), \quad \forall X \in D_n,$$

где $N(X) = \sum_{i=1}^{n} \max\{u_i - x_i, x_i - l_i\}$ - максимально возможный номер уровня для точки X.

Лемма 4. $max\ N(X) = \sum_{i=1}^{n} (u_i - l_i)$,

$$min\ N(X) = \sum_{i=1}^{n} \left\lfloor \frac{1}{2} \cdot (u_i - l_i) \right\rfloor,$$

где $\lfloor \alpha \rfloor = \begin{cases} a/2, \text{если } \alpha \text{ делится на 2} \\ (a+1)/2, \text{ если не делится.} \end{cases}$

Определение 12. Множество $\{X^0, X^1, \ldots, X^k\}$ назовем путем длины k в дискретной решетке, если $X^i \in O_1(X^{i+1})$ $\forall\ i = 0, 1, \ldots, k$.

Определение 13. Путь $\{X^0, X^1, \ldots, X^k\}$ назовем путем убывания, если $f(X^{i+1}) < f(X^i)$ $\forall\ i = 0, 1, \ldots, k$.

Определение 14. Точка $X^* \in D_n$ называется точкой локального минимума функции f, если в D_n не существует пути убывания, ведущего из X^* (т.е. $f(X^*) < f(X)$ $\forall X \in O_1(X^*)$).

Определение 15. Функцию f на дискретной решетке D_n будем называть унимодальной, если для любой точки X существует путь убывания $\{X, X^1, \ldots, X^k, X^*\}$, где X^* - точка минимума.

Определение 16. Унимодальная функция f называется монотонной, если в любой точке $X \in O_k(X^*)$ выполняется $f(Y) \ge f(X)$ $\forall Y \in O_1(X) \cup O_{k+1}(X^*)$.

Лемма 5. Пусть f - унимодальная строго монотонная функция, X^* - ее точка минимума, $X \in O_k(X^*)$ - произвольная точка и $X^i \in O_1(X)$, $i = 1, \ldots, n$, такие, что $|x_i - x_i^i| = 1$, $x_j = x_j^i$ $\forall\ j \ne i$. Тогда точка X^0, полученная по правилу

$$x_i^0 = \begin{cases} x_i, \text{если } f(X) < f(X^i), \\ x_i^i, \text{если } f(X) > f(X^i), i = 1,2,...,n, \end{cases}$$

принадлежит $O_{k-s}(X^*)$, где s - количество соседних точек, имеющих значения функций меньше, чем f(X).

Из приведенных определений и леммы легко получить алгоритмы оптимизации функций, заданных на конечных дискретных решетках, аналогичные алгоритмам, построенным в предыдущем параграфе для псевдобулевых функций.

Алгоритм 5.

1. Выбираем $X^0 \in D_n$ произвольно.

2. Поочередно увеличиваем или уменьшаем координаты X^0 на единицу и вычисляем значения функции в получаемых точках. Запоминаем изменение, приведшее к улучшению.

3. Если ни одно улучшение не найдено, то X^0 - точка минимума, остановиться и вывести ответ.

4. Заменяем все координаты X^0 на те, которые приводили к улучшению и вычисляем значение функции в полученной точке. Обозначаем полученную точку через X^0.

5. Поочередно изменяем координаты точки X^0 таким образом, который приводил к улучшению на предыдущем шаге и вычисляем значения функции в получаемых точках.

6. Если ни одно улучшение не найдено, то проверяем остальные точки окрестности.

7. Перейти к шагу 3.

Данный алгоритм аналогичен по поведению неулучшаемому алгоритму 1, хотя и изложен несколько по другому.

Теорема 7. Пусть f(X) - унимодальная монотонная функция, X^* - ее точка минимума, а X^0 - произвольная стартовая точка оптимизации. Если $\max_i |x_i^0 - x_i^*| = k$, то для определения точки минимума алгоритму 5 понадобится не более $2 \cdot k \cdot n$ вычислений целевой функции.

Нетрудно видеть, что в большинстве случаев алгоритм 5 будет делать заметно меньше, чем $2 \cdot k \cdot n$ вычислений целевой функции, так как полный просмотр окрестности может понадобиться только на последних шагах, а $2 \cdot n$-соседние точки имеются не у всех точек решетки. Таким образом теорема 7 дает (достижимую) оценку сверху.

Приведенный алгоритм работоспособен и на немонотонных функциях, хотя и не обязательно с таким же быстродействием.

Снижение быстродействия (дополнительные затраты на спуск) в случае немонотонной функции появляется за счет того, что не все пути убывания будут сразу же приближать траекторию поиска к точке минимума, что является обязательным при монотонной функции. Более того, траектория спуска может быть столь изощренной, что "намотает" все точки пространства оптимизации. Т.е. оценкой сверху для алгоритма 5 на немонотонных функциях является полный перебор. Поэтому для произвольных функций необходимо предложить другой алгоритм. Как и в случае псевдобулевых функций можно перейти к другой системе окрестностей и отказаться от наискорейшего спуска, т.е. выбрать переход по первому улучшению. Кроме того, различные тактики просмотра окрестности тоже могут генерировать дополнительные схемы алгоритмов.

Рассматривая различные системы окрестностей для дискретной решетки мы указывали три возможых типа. Нетрудно видеть, что первый тип окрестности был выбран для алгоритма 5. Третий тип окрестности, придется отклонить из-за очень большой размерности. Остается только система окрестностей второго типа:

$$N_2(X) = \left\{ Y \in D : \sum_{i=1}^{n} |x_i - y_i| \le 2 \right\},$$

которая, хотя и содержит намного больше точек, чем рассмотренная ранее система первого типа, но зато дает более надежный результат в том смысле, что любая точка минимума по первой системе окрестностей будет таковой и во второй, но обратное утверждение неверно.

Относительно тактики просмотра окрестности необходимо отметить следующее. Во-первых, просмотр необходимо начинать с точек, имеющих суммарную разность координат, равную двум, а не единице, т.к. это позволяет исключить полный перебор в худшем случае за счет того, что вплоть до "накрытия" точки минимума будут просматриваться только точки, имеющие координаты, отличающиеся на четное число от координат стартовой точки. Все точки, отличающиеся на нечетное число, просмотрены не будут при любой функции и при любой траектории поиска, что гарантирует просмотр максимум половины всех точек пространства, т.е. обеспечивает оценку сверху в худшем случае в два раза лучше, чем у алгоритма 5. Кроме того, необходимо начинать с точек, отличающихся одной координатой на две единицы, а не с точек, отличающихся двумя координатами, на единицу каждая. Этим будет обеспечиваться более быстрое продвижение на начальных этапах, когда точка минимума еще

достаточно далеко от стартовой. Последнее, что должно быть оговорено - это способ продолжения просмотра окрестности после перехода в лучшую точку. Возможны три варианта - начинать сначала, продолжать с той же комбинации, что дала улучшение, и - продолжать со следующей комбинации. Первый вариант не очень убедителен, т.к. комбинация, которая только что не давала улучшения, вряд ли будет удачной и в новой окрестности. Второй вариант приводит во многих случаях к вырождению в покоординатный спуск со всеми вытекающими отсюда последствиями (особенно тяжелыми для задачи с ограничениями). Остается только третий вариант, который выглядит наиболее осмысленным. Упорядочение окрестности естественно выбрать следующее. Первыми точками будут точки, отличающиеся одной координатой на два в обе стороны, а затем - точки, отличающиеся парами координат на единицу в обе стороны, упорядоченные лексикографически, и, наконец, точки, отличающиеся только на единицу в порядке номеров координат.

Лемма 6.6 $\max\limits_{X \in D} |N_2(X)| = 2 \cdot n^2$

$$\min_{X \in D} |N_2(X)| = \frac{1}{2} \cdot (n^2 + n)$$

Итак, для решения задач оптимизации на дискретной решетке без существенных ограничений, предлагается алгоритм локального спуска по системе окрестностей $N_2(X)$, реализующий тактику перехода по первому улучшению с продолжением просмотра после перехода в новую точку.

Алгоритм 6.

1. Стартовую точку выбрать произвольно.

2. Увеличивать каждую координату по очереди на два и вычислять значения целевой функции. Если улучшения не произошло, то уменьшать ту же координату на два и вычислять значение функции. В случае улучшения значения функции переходить в точку, дающую такое улучшение и продолжать со следующей координаты.

3. Если изменение всех координат на два не дает улучшения функции, то начинать просмотр всех пар координат в порядке 1-2, 1-3, … , 2-3, 2-4, …, увеличивая и уменьшая каждую координату пары на единицу. Если получено улучшение целевой функции, то переходить в точку, дающую улучшение и продолжать с шага 2.

4. Если на втором и третьем шаге не получено улучшения, то увеличивать и уменьшать каждую координату на единицу,

вычисляя значение функции и запоминая лучшее из них вместе с дающей его точкой.

5. Вывести наилучшее значение целевой функции и точку, в которой это значение получено.

Так как количество вычислений целевой функции алгоритмом 6 достаточно велико, то можно предложить различные эвристические методы ускорения сходимости. Наиболее перспективным выглядит алгоритм с переменной длиной шага, который не намного усложняет процедуру и не существенно увеличивает количество вычислений (при неудачных попытках увеличить длину шага), зато может намного ускорить спуск. Соответствующая пошаговая процедура выглядит также, как и у алгоритма 6, но к ней добавляется еще один шаг, который состоит в том, что на шаге 2, после того, как были поочереди изменены все координаты и в некоторых случаях было получено улучшение, определяются еще две точки, которые отличаются от текущей теми координатами и в ту сторону, которые давали улучшение, на единицу ("осторожный оптимизм") и на двойку ("уверенный оптимизм"), а затем вычисляются значения функции в этих точках. Наилучшая из трех точек (текущая и две "оптимистичных") выбирается для продолжения спуска. В данном алгоритме делается попытка увеличить длину шага спуска с двух до k или 2k, где k - количество координат, давших улучшение на шаге 2. Конечно же возможны и "чрезвычайно оптимистические" варианты, когда шаг будет увеличиваться еще больше вплоть до максимально возможного, но оснований для такого "оптимизма" становится уже совсем мало.

Предложенные алгоритмы обладают определенными достоинствами в одних ситуациях и существенными недостатками в других. Один из путей усилить алгоритмы - обобщение используемых ими моделей поведения. Первый алгоритм использует слабую систему окрестностей и тактику наискорейшего спуска, что приводит к высокой эффективности на монотонных функциях, т.е. дает возможность находить минимум при малом количестве вычислений целевой функции, которые к тому же используются рационально. Но поведение данного алгоритма не рационально в случае немонотонной целевой функции и поиск может даже вырождаться в полный перебор, несмотря на то, что окрестности содержат мало точек. Второй алгоритм не вырождается в полный перебор, хотя и использует намного более мощную систему окрестностей, и ведет себя рационально на немонотонных функциях и вдали от локального минимума, но его

поведение на монотонных функциях хуже, чем у первого алгоритма, а при приближении к минимуму, количество производимых им вычислений целевой функции становится большим и относительные затраты на продвижение к цели резко возрастают, так как путей убывания, ведущих к точке минимума становится все меньше и все больше времени уходит на отыскание хотя бы одного такого пути. Поэтому есть смысл попытаться объединить подходы предложенных алгоритмов с тем, чтобы получаемые алгоритмы сочетали их достоинства.

Первый из обобщенных алгоритмов использует "раздельный" старт и аналогичен предложенному для псевдобулевых функций. Так как свойство монотонности предполагается в каждой точке целевой функции (если она монотонна), то направление спуска по каждой координате может быть определено из отдельных точек с тем же успехом, как и из одной. При этом за счет существенного разброса стартовых точек по поисковому пространству будут исключены худшие траектории спуска в случае немонотонной функции, которые "наматывают" все точки решетки. Если же в случае немонотонности целевой функции использовать спуск по сильной системе окрестностей, то быстродействие алгоритма еще более усилится за счет того, что будут просматриваться точки только четных уровней, а значит полного перебора в принципе не может быть.

Алгоритм 7.

1. Выбрать произвольным образом стартовую точку X^1.

2. Определить (n-1)-соседние к X^1 точки X^2, X^3, ..., X^n, такие что $x_j^j = x_j^1$, j = 2, 3, ... , n, и $|x_i^j - x_j^1| = 1 \ \forall$ i ≠ j, и вычислить значения функции в этих точках.

3. Определить точки $Y^j = (x_1^j, x_2^j, ..., x_{j-1}^j, x_j^j \pm 1, x_{j+1}^j, ..., x_n^j)$, j=1, …, n, и вычислить значения функции в этих точках.

4. Определить точку X^* по правилу

$$x_j^* = \begin{cases} x_j^j, & f(X^j) < f(Y^j), \\ y_j^j, & f(X^j) > f(Y^j), \end{cases} j = \overline{1,n},$$

и вычислить значение функции в ней.

5. Если $f(X^*) < f(X^j)$ $\forall$j=1, …,n, и $f(X^*) < f(Y^j)$ $\forall$j=1, …, n, то определить все точки $Z^i \in O_1(X^*)$ и вычислить значения функции в них. Иначе - идти к шагу 7.

6. Если $f(X^*)<f(Z^i)$ $\forall$ i = 1, …, n, то вывести X^*, $f(X^*)$ и остановиться (X^* - точка минимума, функция f - унимодальная и монотонная).

7. Выбрать точку X^0 из условия

$$f(X^0) = \min \{f(X^*), f(X^j), f(Y^j), f(Z^j), \forall\ j = 1, 2, \ldots, n\}$$

и выполнить локальный спуск в соответствии с шагами 2-5 алгоритма 6.

Второй обобщенный алгоритм также использует переменную структуру в зависимости от состояния процесса оптимизации. Преимущество сильной системы окрестностей в том, что она исключает полный перебор в худшем случае и позволяет быстрее приближаться к точке минимума, если стартовая точка находилась в удалении от нее. Однако вблизи локального минимума сильная система окрестностей существенно замедляет приближение, так как при малом количестве путей убывания, ведущих в точку минимума, слишком много усилий будет затрачиваться на обнаружение этого пути. В худшем случае придется просматривать всю окрестность текущей точки, чтобы установить единственную соседнюю точку, дающую улучшение целевой фунции (количество точек в сильной окрестности порядка n^2), а затем всю окрестность следующей точки и т.д. Это происходит за счет того, что сначала просматриваются 2-соседние точки, а лишь затем - 1-соседние. Такой порядок, дававший существенные преимущества поиску в начале, оборачивается чрезмерными затратами в конце. Переключение системы окрестностей позволит в какой-то мере сочетать преимущества двух типов систем, сводя к минимуму их недостатки.

Алгоритм 8.

1. Стартовую точку выбрать произвольно.

2. Увеличивать каждую координату по очереди на два и вычислять значения целевой функции. Если улучшения не произошло, то уменьшать ту же координату на два и вычислять значение функции. В случае улучшения значения функции переходить в точку, дающую такое улучшение и продолжать со следующей координаты.

3. Если изменение всех координат на два не дает улучшения функции, то начинать просмотр всех пар координат в порядке 1-2, 1-3, … , 2-3, 2-4, …, увеличивая и уменьшая каждую координату пары на единицу. Если получено улучшение целевой функции, то переходить в точку, дающую улучшение и продолжать с шага 2.

4. Если дважды подряд получение лучшей точки потребовало проверки более, чем $4 \cdot n$ 2-соседних точек, то увеличивать и уменьшать каждую координату на единицу, вычисляя значение функции и переходя в точку, дающую лучшее значение.

5. Если после просмотра всей окрестности улучшения не найдено, то вывести наилучшее значение целевой функции и точку, в которой это значение получено.

Предложенные в данном параграфе эвристические алгоритмы реализуют локальный поиск с переключением стратегий в зависимости от результатов поиска на каждом шаге и обладают преимуществом перед предложенными прежде алгоритмами при оптимизации произвольной функции. В более простых и известных заранее ситуациях (например при монотонности целевой функции) данные алгоритмы делают лишние вычисления функции. Нерациональный характер поведения алгоритмов в простой ситуации являестя платой за более высокую эффективность в сложных ситуациях. Во всяком случае предложенные алгоритмы исключают полный перебор в наихудшем случае и существенно сокращают количество проверяемых точек в среднем.

Сравнение алгоритмов друг с другом возможно только на основе численного эксперимента при реальной оптимизации неявных функций, заданных на конечной целочисленной решетке. Проведенное сравнение алгоритмов на тестовых примерах не выявило преимуществ алгоритмов друг перед другом, а также не позволило сделать вывод об областях эффективного применения этих алгоритмов, так как результаты расчетов существенно зависят от выбора стартовой точки, а в остальном собранная статистика каких-либо закономерностей не содержит. Алгоритмы работают примерно одинаково на всех тестовых функциях.

Предложенные алгоритмы являются базовыми процедурами для построения алгоритмов оптимизации, ориентированных на задачи условной и глобальной оптимизации.

Пусть задана целевая функция $f(X)$ и функции-ограничения

$$g_j(X) \le 0,\ j = 1, \ldots, m_1+2 \cdot n, \qquad h_i(X) = 0,\ i = 1, \ldots, m_2,$$

где естественные ограничения на переменные полагаем включенными в список ограничений-неравенств. Возможны два подхода для модификации алгоритмов безусловной оптимизации на случай задачи с ограничениями. Алгоритм с прямой проверкой точек на допустимость по системе ограничений вполне работоспособен на таких задачах, но гарантирует рациональный результат только в сочетании с мультистартом. Другой подход

использует обобщенную функцию со штрафом, например, в следующем виде:

$$\mathrm{F(X)} = f(X) + r \cdot \sum_{j=1}^{m_1+2n} g_j(X) + r \cdot \sum_{i=1}^{m_2} h_i^2(X).$$

Возможны и другие виды штрафных функций.

В этой связи появляется идея генерирования стартовых точек с помощью адаптивных поисковых алгоритмов также, как и для алгоритмов псевдобулевой оптимизации. Аналогично этим алгоритмам алгоритмы дискретной оптимизации. имеют те же гарантии сходимости, что и базовые адаптивные алгоритмы, но требуют большего времени для расчетов, хотя и позволяет преодолеть такие недостатки как остановка вдали от глобального минимума, чрезмерные затраты на получение допустимой стартовой точки и неспособность работать на несвязных допустимых областях и вне допустимого множества. Также очевидно, что в случае простых целевых функций и связных допустимых областей должен применяться только базовый алгоритм.

В случае же сложных целевых функций и ограничений можно рассмотреть различные варианты гибридных алгоритмов, которые могут по-разному генерировать стартовые точки, по-разному обрабатывать ограничения (например, по методу штрафных функций, методом скользящего допуска, с помощью коэволюционирующих популяций и т.п.), запускать различные локальные алгоритмы и т.д.

Метод изменяющихся вероятностей (МИВЕР)
Формальная постановка задачи

Классическая постановка задачи псевдобулевой оптимизации имеет вид:

$$\chi(X) \to extr, \text{ где}$$

$$\chi : B_{2^n} \to R^1; \qquad (1)$$

$$B_{2^n} = \{X : x_j \in B_2, j = \overline{1,n}\}; B_2 = \{0,1\};$$

R^1 - поле действительных чисел. То есть требуется определить точку, дающую требуемое (минимум или максимум) экстремальное значение функции χ, отображающей множество n-мерных векторов с булевыми компонентами на вещественную прямую.

Используя аппарат погружения комбинаторных множеств в арифметическое евклидово пространство, из задачи псевдобулевой оптимизации получаем эквивалентную ей задачу оптимизации функционалов с булевыми переменными:

$$\chi(X) \to \underset{X \in D}{extr},$$

$$D = \{X \in R^n : x_j = 0 \vee 1\}, \qquad (2)$$

где $\chi(X)$ принимает значения из R^1 [сняв это условие, т.е. допуская, что $\chi(X)$ может принимать значения из произвольного множества R^n, дискретного B_2 и т.д. можем получить более общую постановку задачи, уже не эквивалентную указанной постановке (1).

Постановка (2) соответствует задаче без ограничений. На практике чаще имеет место задача

$$\chi(X) \to \underset{X \in S}{extr},$$

где $S \subset D$ - некоторое подмножество D, определяемое заданными системами ограничений.

Как следует из (2), элементам множества D будут соответствовать вершины единичного n-мерного гиперкуба, одна из вершин которого совпадает с началом координат (элементам множества S, очевидно соответствует часть вершин гиперкуба). Т.е. геометрическую поставленную задачу можно интерпретировать как задачу оптимизации функционалов, заданных в вершинах единичного гиперкуба с «нулевой вершиной».

Общая схема МИВЕР

Метод изменяющихся вероятностей (МИВЕР) представляет собой семейство стохастических алгоритмов псевдобулевой оптимизации, имеющих общую схему, в основе которой лежит алгоритм случайного

поиска с адаптацией. В самом общем виде эта схема (обозначим МИВЕР1) применительно к задаче (2) выглядит следующим образом:

1. Задаются начальные значения компонент вектора вероятностей $P^0 = \{p_1^0, ..., p_n^0\}$, где $p_j = P\{x_j = 1\} = 1/2$ - вероятность присвоения j-й $(j = \overline{1,n})$ компоненте вектора $X \in D$.
2. Случайным образом (в соответствии с распределением вероятностей P^k, $k = \overline{1,R-1}$), независимо выбираются r векторов $X^i \in D$, $i = \overline{1,r}$.
3. Вычисляются соответствующие значения функционала $\chi(X^i), i = \overline{1,r}$.
4. Определяются $\chi_{\min}^k = \min\limits_{i=1,r}\{\chi(X^i)\}$ и $\chi_{\max}^k = \max\limits_{i=1,r}\{\chi(X^i)\}$. Значения $\chi_{\min}^k$ и $\chi_{\max}^k$ и соответствующие векторы $X_{\min}^k$ и $X_{\max}^k$ запоминаются.
5. По результатам п. 4 изменяются (в соответствии с правилом конкретного алгоритма) компоненты вектора вероятностей P^k.
6. Пункты 2-5 повторяются R раз.
7. За решение задачи принимается вектор X^*, определяемый из условия $\chi(X^*) = \min\limits_{k=1,R}\{\chi_{\min}^k\}$ ($\chi_{\min}^k = \min\limits_{i=1,r}\{\chi(X^i)\}$ при $k-м$ повторении п. 4).

Начальное распределение вероятностей единичных значений компонент вектора X - P^0 определяется на основе априорной информации о решаемой задаче. Если такая информация отсутствует, используется равновероятностное распределение, т.е. $p_j^0 = 1/n, j = \overline{1,n}$.

Пункт 5 соответствует получению апостериорного распределения вероятностей по результатам испытаний п. 2 и п. 3.

Были предложены следующие алгоритмы пересчета вектора вероятностей единичных компонент: Случайный поиск с адаптацией (СПА) и модификации случайного поиска с адаптацией (МСПА1, МСПА2, МСПА3):

СПА $j = \overline{1,n}$:

На пятом шаге алгоритма МИВЕР компоненты вектора P^k выбираются по следующему правилу:

$$\overline{p_j^k} = \begin{cases} p_j^k, \ если \ x_j = 1 \notin X_{\max}^k, \\ p_j^k - h^-, \ если \ x_j = 1 \in X_{\max}^k; \end{cases}$$

$$p_j^{k+1} = \begin{cases} \overline{p_j^k}, \ если \ x_j = 1 \notin X_{\min}^k, \\ \overline{p_j^k} + h^+, \ если \ x_j = 1 \in X_{\min}^k; \end{cases} . \qquad (3)$$

Адаптация сводится к изменению компонент вероятностей на последующих этапах поиска в зависимости от предыдущих.

Существенным недостатком метода СПА является то, что на каждом этапе поиска часть единичных компонент X^* может совпадать с единичными компонентами как $\chi_{\min}^k$, так и $\chi_{\max}^k$, в результате чего вероятности

присвоения единичных значений для ряда единичных компонент X^* не будут увеличиваться (а могут и уменьшаться), что заметно ухудшает сходимость алгоритма и при решении отдельных задач вообще не позволяет получить удовлетворительного решения. Поэтому была предложена модификация СПА (МСПА1), которая заключалась в том, что при реализации п. 4 МИВЕР определялись лишь $\chi^k_{\min}$, $k=\overline{1,n}$ и в п. 5 компоненты вектора вероятностей P^k определялись по правилу $j=\overline{1,n}$:

$$p_j^{k+1}=\begin{cases} p_j^k+h^+, \ если \ x_j=1\in X^k_{\min}, \\ p_j^k-h^-, \ если \ x_j=1\notin X^k_{\min}; \end{cases}, \qquad (4)$$

где $h^-=h^+=1/R\cdot n$.

Как видно из (3) и (4), СПА реализует идею «наказания» единичных компонент «наихудшего» (по критерию) и «поощрения» единичных компонент «наилучшего» векторов из рассмотренных на этапе. В МСПА1 в соответствии с (4) реализуется идея «поощрения» единичных компонент «наилучшего» вектора за счет его нулевых компонент, поэтому алгоритм МСПА1 в основном избавлен от вышеуказанного недостатка.

В большей степени учесть поступающую апостериорную информацию позволяют алгоритмы МСПА2 и МСПА3, единственным отличием которых от алгоритма СПА является то, что в п. 5 схемы МИВЕР компоненты вектора P^k для МСПА2 определяются по правилу ($j=\overline{1,n}$):

$$p_j^{k+1}=\begin{cases} p_j^k+h_1^+, если \ x_j=1\in X^k_{\min}, \\ p_j^k-h_1^-, если \ x_j=1\in X^k_{\max}, \\ p_j^k-h_2^-, если \ x_j=1\notin X^k_{\min}\cup X^k_{\max}, \\ p_j^k, если \ x_j=1\in X^k_{\min}\cap X^k_{\max}; \end{cases}, \qquad (5)$$

а для МСПА3 по правилу ($j=\overline{1,n}$):

$$p_j^{k+1}=\begin{cases} p_j^k+h_1^+, если \ x_j=1\in X^k_{\min}, \\ p_j^k-h_1^-, если \ x_j=1\in X^k_{\max}, \\ p_j^k+h_2^+, если \ x_j=1\notin X^k_{\min}\cup X^k_{\max}, \\ p_j^k, если \ x_j=1\in X^k_{\min}\cap X^k_{\max}; \end{cases} \qquad (6)$$

Очевидно, что для того что бы не получить недостатка алгоритма СПА, шаги изменения вероятностей в (5) и (6) должны задаваться из условий $h_1^+>h_1^-$ $>h_2^-$ для МСПА2 и $h_1^+>h_1^-$ $>h_2^+$ для МСПА3. Кроме того, $h_1^-\le 1/R\cdot n$ (чтобы не получить отрицательных вероятностей, если какие-то компоненты вектора X будут «наказываться» на каждом шаге поиска), наконец, величины

шагов в алгоритмах должны быть таковы, что бы на каждом шаге поиска выполнялось условие нормировки для распределения P^k.

Алгоритмы СПА и МСПА при безусловной оптимизации

Первоначально метод был реализован для задачи условной псевдовулевой оптимизации, поэтому для безусловной оптимизации метод модифицирован так, что решение ищется во множестве S. Множество $S = \left\{ Y \in R^{2n} : y_i = 0 \vee 1, \sum_{j=1}^{2n} y_j = n \right\}$. Общая схема (обозначим МИВЕР2) имеет вид:

1. Задаются начальные значения компонент вектора вероятностей $P^0 = \{p_1^0, ..., p_{2n}^0\}$, $p_j = P\{y_j = 1\} = 1/2n$ $(j = \overline{1, 2n}), Y \in S$.
2. Случайным образом (в соответствии с распределением вероятностей P^k, $k = \overline{1, R-1}$), независимо выбираются r векторов $Y^i \in S, i = \overline{1,r}$ и соответствующие значения $X^i \in D, i = \overline{1,r}$ по правилу $x_i = y_i$, $i = \overline{1,n}$.
3. Вычисляются соответствующие значения функционала $\chi(X^i), i = \overline{1,r}$.
4. Из условий $\chi(\overline{X}^k) = \min\limits_{i=\overline{1,r}}\{\chi(X^i)\}$ и $\chi(\overline{\overline{X}}^k) = \max\limits_{i=\overline{1,r}}\{\chi(X^i)\}$ находятся векторы $\overline{X}^k$ и $\overline{\overline{X}}^k$. Значения $\chi^k_{\min} = \chi(\overline{X}^k)$ и $\chi^k_{\max} = \chi(\overline{\overline{X}}^k)$ запоминаются. По правилу: $\overline{y}^k_j = \overline{x}^k_j$, $\overline{\overline{y}}^k_j = \overline{\overline{x}}^k_j$ для $j = \overline{1,n}$;

   $$\overline{y}^k_j = \begin{cases} 1, & для\ j = \overline{n+1, 2n-m'} \\ 0, & для\ j = \overline{2n-m'+1, 2n} \end{cases};$$

   $$\overline{\overline{y}}^k_j = \begin{cases} 1, & для\ j = \overline{n+1, 2n-m''} \\ 0, & для\ j = \overline{2n-m''+1, 2n} \end{cases},$$

   где m' и m'' - число единичных компонент векторов $\overline{X}^k$ и $\overline{\overline{X}}^k$ соответственно, определяются векторы $\overline{Y}^k$ и $\overline{\overline{Y}}^k$, которые тоже запоминаются.
5. По результатам п. 4 изменяются (в соответствии с правилом конкретного алгоритма (СПА, МСПА1, МСПА2)) компоненты вектора вероятностей P^k.
6. Пункты 2-5 повторяются R раз.
7. За решение задачи принимается вектор X^*, определяемый из условия $\chi(X^*) = \min\limits_{k=\overline{1,R}}\{\chi^k_{\min}\}$ ($\chi^k_{\min} = \min\limits_{i=\overline{1,r}}\{\chi(X^i)\}$ при $k-м$ повторении п. 4).

СПА $j = \overline{1,2n}$:

$$\overline{p_j^k} = \begin{cases} p_j^k, \ если \ y_j = 1 \notin \overline{\overline{Y}}^k, \\ p_j^k - h^-, если \ y_j = 1 \in \overline{\overline{Y}}^k; \end{cases}$$

$$p_j^{k+1} = \begin{cases} \overline{p_j^k}, \ если \ y_j = 1 \notin \overline{Y}^k, \\ \overline{p_j^k} + h^+, если \ y_j = 1 \in \overline{Y}^k; \end{cases},$$

где $h^- = h^+ = \frac{1}{2 \cdot R \cdot n}$.

МСПА1, МСПА2 и МСПА3 меняются аналогично.

Исследование работоспособности на тестовых задачах

Методы МИВЕР1 и МИВЕР2 был программно реализованы. Исследование алгоритмов СПА, МСПА1 и МСПА2 проводилось на тестовых функциях: функция Растригина (первоначально бинаризуется) и многоэкстремальная псевдобулевая функция.

Задача 1. Дискретная функция Растригина:

$F(x,y) = 8 + (0.1x)^2 + (0.1y)^2 - 4\cos(0.8x) - 4\cos(0.8y)$, $x, y = 0, \pm1, \pm2, ..., \pm64$.

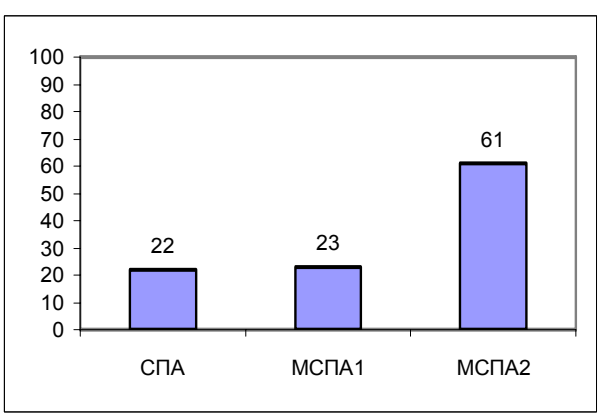
$\min F(x,y) = 0, \ (x^*, y^*) = (0,0)_{10} = (1000000, 1000000)_2$.

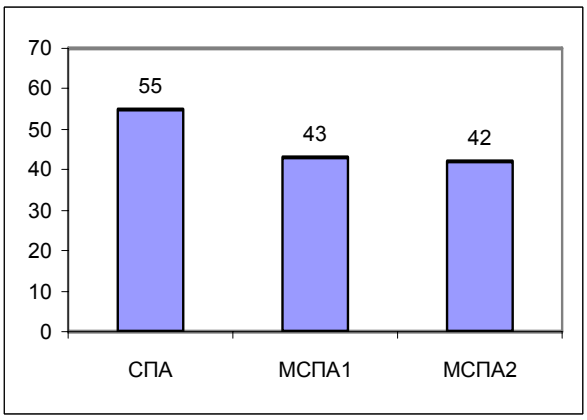
Задача 2. Задача псевдобулевой оптимизации:

$F(\bar{x}) = \sin(\sum_{i=1}^{30} i \cdot x_i)$, $x_i = 0 \vee 1$. $\min F(\bar{x}) = \sin(344)$.

Для исследования эффективности использовались показатели надежности и среднее число итераций. Показатели вычислялись по статистике набранной из 100 запусков алгоритмов. Для всех алгоритмов $r = 50, \ R = 100$. Результаты исследования эффективности представлены ниже.

Задача 1.





а) б)

Рис.1 а) Надежность алгоритма. б) Среднее число итераций до определения минимума.

Зональные алгоритмы схемы МИВЕР

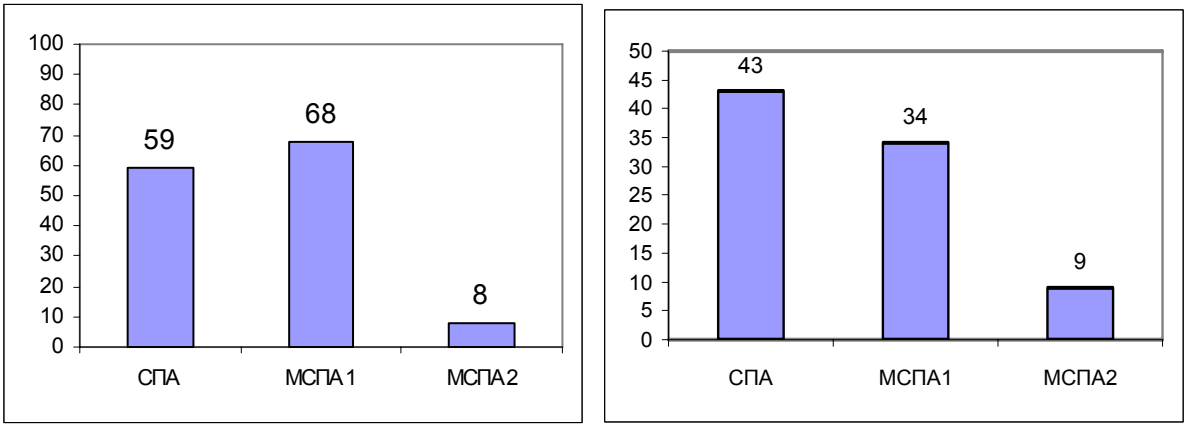
Предлагаемые алгоритмы позволяют учесть больший объем апостериорной информации, полученной на каждом этапе поиска.

Зональный алгоритмы СПА (СПАЗ)

Отличаются друг от друга только п. 5 (пересчет апостериорного распределения вероятностей) общей схемы.

Пункты 1-3 те же, что и у алгоритма СПА.

Задача 2.



а) б)

Рис.2 а) Надежность алгоритма. б) Среднее число итераций до определения минимума.

Пункт 4. Определяются $\chi^k_{\min} = \min\limits_{i=\overline{1,r}}\{\chi(X^i)\}$ и $\chi^k_{\max} = \max\limits_{i=\overline{1,r}}\{\chi(X^i)\}$. $\chi^k_{\min}$ и соответствующий вектор $X_{\min}{}^k$ запоминаются. Выбираются векторы $\overline{X}^k{}_1,...,\overline{X}^k{}_{Q_k}$, для которых выполняются $\chi(\overline{X}^k_q) \in [\chi^k_{\min}, \alpha\chi^k_{\min}], q = \overline{1,Q}_k$, и векторы $\overline{X}^k{}_1,...,\overline{X}^k{}_{G_k}$, для которых $\chi(\overline{X}^k_g) \in [\beta\chi^k_{\max}, \chi^k_{\max}], g = \overline{1,G}_k$. Здесь α и β – *const*, α>1, β<1, которые задают размеры зон сбора дополнительной апостериорной информации. Очевидно, что всегда должно выполняться $\alpha\chi^k_{\min} \le \beta\chi^k_{\max}$ и $Q_k + G_k \le r, k = \overline{1,R}$.

Пункт 5. Алгоритм СПАЗ:

$$\overline{p}_j^{k+1} = \begin{cases} p_j^k, \text{если } x_j = 1 \notin \{\overline{\overline{X}}^k_1,...,\overline{\overline{X}}^k_{G_k}\}; \\ p_j^k - h_1^-, \text{если } x_j = 1 \in \{\overline{\overline{X}}^k_1,...,\overline{\overline{X}}^k_{G_k}\} t_1 \text{ и более раз}; \\ p_j^k - h_2^-, \text{если } x_j = 1 \in X^k_{\max}, \end{cases}$$

где $t_1 = E[G_k / 2]$ (здесь $E[\gamma]$ означает целую часть числа γ), $h_2^- = 1/R \cdot n, h_1^- < h_2^-$.

$$p_j^{k+1} = \begin{cases} \overline{p}_j^{k+1}, если\ x_j = 1 \notin \{\overline{X}_1^k, ..., \overline{X}_{Q_k}^k\}; \\ \overline{p}_j^{k+1} - h_1^+, если\ x_j = 1 \in \{\overline{X}_1^k, ..., \overline{X}_{Q_k}^k\} t_2 \quad и \quad более \quad раз; \\ \overline{p}_j^{k+1} - h_2^+, если\ x_j = 1 \in X_{\min}^k, \end{cases}$$

где $t_2 = E[Q_k / 2]$, $h_1^+ > h_2^+$.

Один из возможных способов задания величины шагов изменения вероятностей:

$$h_2^- = 1/R \cdot n,$$

$$h_1^- = \begin{cases} 1/R \cdot n\gamma_1, если \quad \gamma_1 \neq 0; \\ 0, если \quad \gamma_1 = 0, \end{cases}$$

$$h_2^+ = \begin{cases} n - m/m \cdot R \cdot n, если \quad \gamma_1 \neq 0 \vee \gamma_2 \neq 0 \wedge \gamma_1 = 0 \vee \gamma_2 = 0; \\ n - m/2m \cdot R \cdot n, если \quad \gamma_1 \neq 0 \vee \gamma_2 = 0; \\ n - m + 1/m \cdot R \cdot n, если \quad \gamma_1 = 0 \vee \gamma_2 \neq 0, \end{cases}$$

$$h_2^+ = \begin{cases} n - m/m \cdot R \cdot n, если \quad \gamma_1 \neq 0 \vee \gamma_2 \neq 0 \wedge \gamma_1 = 0 \vee \gamma_2 = 0; \\ n - m/2 \cdot \gamma_2 \cdot m \cdot R \cdot n, если \quad \gamma_2 \neq 0 \vee \gamma_1 = 0; \\ 0, если \quad \gamma_2 = 0, \end{cases}$$

где $\gamma_1 = \sum_{j=1}^{n} \prod_{q=1}^{Q_k} \overline{x}_{jq}^{-k}$; $\gamma_2 = \sum_{j=1}^{n} \prod_{g=1}^{G_k} \overline{x}_{jg}^{=k}$;

m определяет множество *S*.

Зональный алгоритм МСПА

Отличие от исходного алгоритма в п. 4 и п. 5.

Пункт 4. Выбираются векторы $\overline{X}^k{}_1, ..., \overline{X}^k{}_{Q_k}$, значения функционала в которых принадлежат $[\chi_{\min}^k, \alpha\chi_{\min}^k]$ (α>1 – *const*).

Пункт 5.

$$p_j^{k+1} = \begin{cases} p_j^k + h_1^+, если\ x_j = 1 \in \{\overline{X}_1^k, ..., \overline{X}_{Q_k}^k\} t_1 \quad и \quad более \quad раз, \\ p_j^k + h_2^+, если\ x_j = 1 \in X_{\min}^k, \\ p_j^k - h_1^-, во \quad всех \quad остальных \quad случаях. \end{cases}$$

Возможно усиление алгоритма:

$$p_j^{k+1} = \begin{cases} p_j^{k+1} + h_1^+, \textit{если } x_j = 1 \in \{\overline{X}_1^k, ..., \overline{X}_{Q_k}^k\} t_1 \quad \textit{и} \quad \textit{более} \quad \textit{раз}; \\ p_j^k + h_2^+, \textit{если } x_j = 1 \in X_{\min}^k; \\ p_j^{k-1} - h_1^+, \textit{если } x_j = 1 \notin \{\overline{X}_1^k, ..., \overline{X}_{Q_k}^k\} t_4 \quad \textit{и} \quad \textit{более} \quad \textit{раз}; \\ p_j^k - h_2^+, \textit{если } x_j = 1 \notin X_{\min}^k. \end{cases}$$

Здесь $t_4 = E[(r - Q_k)/2]$, $h_1^+ < h_2^+$, $h_1^- < h_2^-$. В Усиленном алгоритме, так же как и в алгоритме СПАЗ трудно задать величину шагов h_1^- и h_2^-, обеспечивающую неотрицательность вероятностей распределений.

Алгоритмы интеллектуального поиска (поиск табу)

Методы интеллектуального поиска, обычно называемые поиском табу, получили широкое распространение и признание в решении практических оптимизационных задач. Приложения этих методов стремительно расширяются в таких областях, как управление ресурсами, проектирование процессов, логистика, планирование и общая комбинаторная оптимизация. Сочетание с другими методами, как эвристическими, так и алгоритмическими, также дает продуктивные результаты. Все это послужило основанием организовать специальное издание известного журнала "Анналы исследования операций (Annuals of Operations research)", целиком посвященное поиску табу и его приложениям. Базовые идеи поиска табу были нами задействованы при построении алгоритмов локального поиска в четвертой главе.

В данном параграфе рассматриваются основные характеристики поиска табу, которые в наибольшей степени влияют на успешность его применения, а также приводятся основные сведения, позволяющие построить улучшенные методы интеллектуального поиска для целей поддержки принятия решения.

Метод поиска табу (TS) является *метаэвристикой*, которая управляет процедурой *локального* эвристического поиска с целью *глобального* исследования пространства решений. В последние годы стремительно расширяется область приложений метода поиска табу в оптимизации. Процедуры TS, включающие основные концепции и гибридные схемы, сочетающие эти концепции с другими эвристическими и алгоритмическими методами, успешно применялись в различных практических задачах.

Поиск табу основан на предположении, что процесс решения задач, претендующий на звание “интеллектуального”, должен включать *адаптивную память* и *тщательное исследование ситуации*.

Использование адаптивной памяти контрастирует с подходами без памяти, пришедшими из биологии и физики, и с методами с негибкой памятью, примером которой может служить метод ветвей и границ, а также его модификации. Основной идеей в поиске табу как при детерминированном, так и при стохастическом подходе, является предположение, что плохой выбор, *основанный на* продуманной *стратегии*, может принести больше информации, чем хороший *случайный* выбор. В системах, использующих память, плохой выбор, основанный на стратегии, может обеспечить пригодной к использованию информацией о том, как стратегия может быть выгодно изменена. Даже в пространстве со значительным влиянием случайности, которая тем не менее недостаточна для того, чтобы уничтожить все следы порядка в большинстве реальных задач, целенаправленное проектирование может быть более успешным в раскрытии внутренней структуры задачи и тем самым дать шанс использовать условия в

тех ситуациях, когда случайность не является всеподавляющей. Основные характеристики поиска табу приведены в табл. 1.

Таблица 1.

Основные черты поиска табу
Адаптивная память
Избирательность (включая стратегическое забывание)
Абстракция и декомпозиция (с явной и атрибутивной памятью)
Фактор времени:
новизна событий
частота событий
различие между короткими и длинными интервалами времени
Качество и воздействие:
относительная привлекательность альтернатив
масштабы изменений в структуре или ограничивающих отношениях
Контекст:
региональная взаимозависимость
структурная взаимозависимость
последовательная взаимозависимость
Тщательное исследование ситуации
Стратегически накладываемые ограничения и предпочтения (*условия табу* и *уровни аспирации*)
Концентрация внимания на хороших регионах и свойствах хороших решений (*интенсификация процессов*)
Установление и исследование новых обещающих областей (*диверсификация процессов*)
Немонотонные стратегии поиска (*стратегическая осцилляция*)
Интеграция и расширение решений (*пересоединение путей*)

Поиск табу предназначен для нахождения новых и более эффективных путей получения улучшенных решений с помощью концепций, приведенных в табл. 1, и для определения соответствующих принципов, которые могут расширить основы интеллектуального поиска.

Как это обычно случается, появляются новые сочетания базовых идей, что приводит к улучшенным решениям и лучшим практическим результатам. Это делает TS привлекательной областью для исследований и использования в практических задачах.В последнее время появились гибридные схемы, сочетающие поиск табу не только с локальным поиском, но и с эволюционными алгоритмами и имитацией отжига, которые и сами по себе являются глобальными поисковыми процедурами. Применение в практических задачах показывает, что эти подходы имеют большое будущее.

Основные положения метода могут быть описаны следующим образом. Задана функция f(x), которая должна быть оптимизирована в множестве X. TS начинает как и обычный локальный поиск, итеративно обрабатывая одну

точку (решение) за другой до тех пор, пока не будет удовлетворен выбранный критерий остановки. Каждое $x \in X$ имеет *окрестность* $N(x) \subset X$, и каждое решение $x' \in N(x)$ достижимо из x при помощи операции, называемой *перемещением*.

TS выходит за рамки локального поиска путем применения стратегии модификации окрестности N(x) по ходу поиска, эффективно заменяя ее на другую окрестность N*(x). Как видно из вышесказанного, ключевой аспект TS заключается в использовании специальных структур памяти, которые служат для определения N*(x), и в организации траектории, по которой исследуется пространство оптимизации.

Решения, включаемые в N*(x) этими структурами памяти, определяются несколькими способами. Один из них, давший поиску табу его название, определяет решения, с которыми сталкивается на определенном уровне (и, неявно, дополнительные связанные с ними решения), и запрещает им принадлежать N*(x), классифицируя их как *табу*. (Термин "табу" предназначен для выражения типа ограничения, которое имеет больше "культурное" значение, то есть такое значение, которое находится под влиянием истории и конкретных условий и может быть изменено, если условия позволят).

Процесс, посредством которого решения приобретают статус табу, имеет несколько составляющих, разработанных для того, чтобы обеспечить обоснованно строгую проверку новых точек. Полезным способом организации этого процесса является замена обычного оценивания решений *оценками табу*, которые вводят штрафы, чтобы значительно сократить выбор решений, то есть определить те из них, которые лучше исключить из N*(x) в соответствии с их зависимостью от элементов, составляющих статус табу. Кроме того, оценки табу также периодически включают поощрения, чтобы увеличить выбор решений других типов в результате каких-то дополнительных соображений о предпочтениях и влияния долговременных факторов.

Следует подчеркнуть, что понятие окрестности, применяемое в поиске табу, отличается от применяемого в локальном поиске, охватывая типы перемещений, применяемых в конструктивных и деструктивных процессах (основания для таких перемещений называются, соответственно, *конструктивными* и *деструктивными* окрестностями). Такое использование расширенного понятия окрестности укрепляет фундаментальную перспективу TS, которая должна определять окрестности динамическими способами, чтобы иметь возможность включать последовательные или параллельные операции со многими типами перемещений.

Ввиду того, что TS имеет несколько важнейших компонент, и задача объединения их может показаться на первый взгляд чрезмерно трудоемкой, многие разработки были основаны только на первых идеях, рассматриваемых

к тому же в самой общей постановке. Однако надо подчеркнуть, что число важнейших компонент не так уж и велико (каждая с несколькими основными вариантами), и, один раз систематизированные, они составляют взаимосвязанную схему, которая оказывается значительно более эффективной, чем одна-две компоненты по отдельности.

Уровни аспирации. Расширяя определение условий табу различного ограничивающего уровня, введем еще один важный элемент, придающий поиску табу дополнительную гибкость, определив его как уровень аспирации (предпочтительности, привлекательности). Статус табу для решения не является абсолютным, а может быть отменен, если появились определенные условия, выражаемые в форме уровней аспирации. Такие уровни аспирации задают определенные пороги привлекательности, которые указывают можно ли рассматривать решения как допустимые, несмотря на то, что они классифицированы как табу. Ясно, например, что решение лучшее, чем ранее проверенные, заслуживает быть рассмотренным как допустимое. Подобный критерий аспирации может быть определен на подмножествах решений, которые принадлежат одному региону или все обладают специфическими свойствами (как некоторое частное значение функции или уровень недопустимости). Например, один из таких критериев аспирации базируется на определении “условно наилучших” значений целевой функции, которые могут быть достигнуты посредством перемещений, начинающихся из некоторых интервалов значений f(x). Затем перемещение считается приемлемым, если оно может достигнуть нового лучшего значения для интервала, из которого оно начинается.

Упомянутый подход естественно обобщается заменой интервалов значений целевой функции другими типами интервалов. В этом случае часто предпочтительнее задавать условие таким образом, чтобы перемещение достигало лучшего относительно интервала значения целевой функции в конце перемещения, а не в начале. Это более точно соответствует стандартному табу-ограничению, исключая то, что оно используется для преодоления других ограничений. (Неявно оно соответствует специальному типу конъюнкции.)

Стратегии списка кандидатов. Активность поиска табу подкрепляется поиском наилучшего доступного перемещения, которое может быть определено с приемлемыми дополнительными затратами. Следует помнить, что понимание наилучшего не ограничивается значением целевой функции. (Как уже отмечалось, оценки табу находятся под воздействием штрафов и стимулов, определяемых историей поиска, а также под воздействием некоторых соображений о *влиянии*, которые будут охарактеризованы позже). Для ситуаций, когда N*(x) велика или оценивание ее элементов дорого, стратегии списка кандидатов используются для ограничения числа решений, проверяемых на данной итерации.

Так как предписание поиска табу отбирать элементы обоснованно является исключительно важным, решающими для процесса поиска являются эффективные правила для генерирования и оценивания хороших кандидатов. Даже там, где стратегии списка кандидатов не используются явно, структуры памяти для эффективного обновления оценок перемещений от одной итерации к другой и для уменьшения затрат на нахождение наилучших или близких к ним перемещений, часто интегрированы в разработку TS. Разумное обновление может значительно уменьшить время получения решения, а использование стратегий списка кандидатов явным образом для больших задач может значительно повысить результаты.

Представление штрафов как “большого” или “очень маленького” выражает пороговый эффект. Ранее мы задавали статус табу как условие типа “все-или-ничего”, но ясно, что возможна дифференциация, например, с учетом различного числа табу-активных атрибутов или с различием уровней не окончившихся сроков табу. Статус табу в общем случае соответствует такому использованию штрафов, когда в список табу отбираются решения с очень плохими оценками (большими штрафами), и наоборот, служит для сохранения связей между очень хорошими решениями, когда нужно блокировать изменения, нарушающие наборы атрибутов, стабильно дающие высокие оценки.

Если все доступные в данный момент перемещения приводят к решениям, которые являются табу и должны бы быть исключенными из дальнейшего рассмотрения, использование штрафа приводит к выбору решения, соответствующего “наименьшему табу”. Этот подход дает возможность избежать крайних вариантов, когда или все перемещения могут быть выбранными (обычный поиск) или нежелательные перемещения вообще не могут быть выбраны (поиск табу типа “все или ничего”).

Оценка табу может быть модифицирована также путем создания стимулов, основанных на уровнях аспирации, также как эта оценка модифицируется на основе штрафов, основанных на статусе табу. В этом смысле условия табу и условия аспирации можно рассматривать как “зеркальные отражения” друг друга.

Долговременная память. В некоторых приложениях компоненты кратковременной памяти в TS способны давать решения очень высокого качества. Однако, в общем случае, TS становится значительно сильнее при включении в него долговременной памяти и связанных с ней стратегий. (Большое число реализаций TS, содержащих только кратковременную память, было в последствии значительно улучшено введением компонентов долговременной памяти).

При рассмотрении более длительных интервалов времени главным аспектом являются специальные типы памяти, *основанной на частоте*. Их работа обеспечивается введением штрафов и стимулов, определяемых

относительным интервалом времени, в течение которого атрибуты принадлежали решениям, проверенным при поиске. Разрешается также делать различия для различных областей поиска. *Частоты переходов* сохраняют информацию о том, как часто атрибуты изменялись, в то время как *резидентные частоты* дают информацию об относительных промежутках времени, когда атрибуты входили в генерируемые решения. Эти виды памяти иногда сопровождаются расширенными формами памяти, основанной на недавнем прошлом.

Возможно будет неожиданным тот факт, что использование долговременной памяти не требует длительной работы процедуры поиска для того, чтобы выгода от нее стала очевидной. Часто ее достоинства начинают проявляться через относительно короткий промежуток времени, что позволяет предотвратить излишние усилия по решению задачи благодаря тому, что может быть найдено решение очень высокого качества за очень короткое время. Например, самые быстрые методы решения задач составления расписаний различного типа основаны на включении долговременной памяти (как явной так и атрибутивной). С другой стороны, также справедливо, что шансы найти более хорошее решение - в случае, когда оптимальное решение еще не найдено - с ростом времени повышаются при использовании долговременной памяти в дополнение к кратковременной.

Интенсификация и диверсификация. Два в высшей степени важных компонента TS - это *стратегии интенсификации* и *стратегии диверсификации*. Стратегии интенсификации основаны на модификации правил выбора для поощрения тех комбинаций перемещений и свойств решений, которые были признаны удачными в течение предыдущего времени. Они могут быть применены с конструктивными и деструктивными окрестностями, а также с окрестностями переходов, путем перезапуска поисковых процедур, которые включают хорошие атрибуты в текущую структуру алгоритма с учетом уже включенных атрибутов. Такие подходы хорошо работали при выработке правил выбора для вероятностного поиска табу. Стратегии интенсификации могут также инициировать возврат к многообещающим регионам для более подробного их исследования.

Стратегия выбора элитных решений исключительно важна. Доказано, что три варианта являются исключительно удачными. В некоторых работах представлена мера диверсификации для обеспечения того, что записанные решения отличаются друг от друга в нужной степени, и последующего очищения кратковременной памяти перед записью лучших решений. Второй вариант хранит последовательный список ограниченной длины, в конец которого добавляет новое решение только в случае, если оно лучше всех, найденных раньше. Текущий последний член списка всегда выбирается (и удаляется) в качестве базиса для возобновления поиска. Однако

кратковременная память TS, соответствующая этому решению, также сохраняется, что запрещает использовать в качестве первого перемещения ранее использованное перемещение из этого решения, так что будет начат новый путь получения решения.

Третий вариант работает с упорядоченным списком из k лучших решений. Через определенное число итераций начинается перебор с худшего члена списка по направлению к лучшему. Текущий отобранный член порождает новый процесс, используя вероятностный поиск табу как альтернативу простому восстановлению прежней памяти. Выполняется только фиксированное число итераций перед запуском нового процесса, однако список обновляется непрерывно. Таким образом, когда найдено более хорошее решение чем имеющееся наихудшее, это новое решение вставляется в список на соответствующее место, а наихудшее удаляется из списка. Доказана высокая эффективность этого метода для задач проектирования сетей телекоммуникации.

Эти подходы - пример того, что иногда называется подходом с *реструктурированым перемещением*, отражая тот факт, что обычное множество перемещений периодически модифицируется с тем, чтобы позволить прямой скачок к решению, лежащему за пределами текущей окрестности. При таком подходе хранится информация о наилучших непроверенных еще соседних решениях (включенных в список кандидатов) с ограничением на специфические типы решений, типа соседей локальных оптимумов или соседей решений, проверенных непосредственно перед достижением таких локальных оптимумов. Несмотря на то, что эта стратегия “непроверенных соседей” еще недостаточно исследована, было замечено, что похожие стратегии, обеспечивали решения хорошего качества.

Другой тип интенсификации - *интенсификация декомпозицией*, когда накладываются ограничения на некоторые части задачи или структуры решения для того, чтобы получить такой способ декомпозиции, который бы дал возможность больше сконцентрироваться на других частях структуры. Классический пример - задача о коммивояжере, когда ребра, которые принадлежат пересечению элитных туров, могут быть “навсегда” включены в решение, для того, чтобы сосредоточить внимание на манипулировании другими частями маршрута. Использование сечений может быть рассмотрено как крайний пример более общей стратегии, которая позволяет идентифицировать и ограничивать значения *строго определенных* и *последовательных переменных*. В этом подходе информация о частотах хранит сведения о переменных, которые принимают некоторые конкретные значения (или значения из некоторых интервалов) на подмножествах элитных решений. Качество решений, в которых реализуются эти значения, и разрушающий эффект изменений таких значений и дают степень их предпочтительности. Ограничение значений соответствующих переменных с

использованием подобной информации может привести к идентификации дополнительных переменных, которые также следует ограничить по тем же причинам, что привносит рекурсивность в этот подход. Общий эффект может быть сравнен с созданием *комбинаторного сокращения* числа возможностей (по обратной аналогии с образным выражением “комбинаторный взрыв”), так как ограничение дискретных переменных, например путем временной их фиксации или удаления, приводит в точности к противоположному эффекту по сравнению с добавлением новых дискретных переменных. Этот тип интенсификации был очень эффективно применен к задаче составления маршрутов для транспортных средств.

Интенсификация декомпозицией приводит к рассмотрению и других типов стратегий, основанных не только на показателях силы и последовательности, но и на возможностях продуктивного взаимодействия некоторых элементов. Для задач на перестановках, которые могут быть удобно описаны в терминах графов (например, составление расписаний, маршрутизация транспортных средств, задача коммивояжера), декомпозиция может основываться на идентификации некоторых подцепочек элитных решений, где две или больше подцепочек могут быть назначены в общее множество, если они содержат узлы, которые “очень привлекательны”, чтобы соединить их с узлами других подцепочек множества. С ребром, расчленяющим набор цепочек, может работать процесс интенсификации, который оперирует параллельно с каждым множеством при ограничении, что начальные и конечные вершины подцепочек остаются неизменными. В результате такой декомпозиции могут быть найдены новые лучшие множества подцепочек, которые создадут новое решение. Такой процесс может быть применен для увеличения альтернативных видов декомпозиций в более широких формах интенсификации декомпозицией.

Стратегии диверсификации. Стратегии диверсификации TS, как следует из названия, разрабатываются для того, чтобы направлять поиск в новые области. Часто они основаны на модификации правил выбора, которые вводят некоторые атрибуты в редко используемые решения. Другой путь - они могут вводить такие атрибуты путем частичного или полного перезапуска процесса поиска решения. В качестве основы для таких процедур можно использовать те же виды памяти, что были описаны ранее, хотя они должны работать с другими (вообще говоря более крупными) подмножествами решений, нежели обрабатываемые в стратегиях интенсификации.

Однако необходимо подчеркнуть, что при введении диверсификации такого типа очень важным является правильный выбор момента времени. Диверсификация не может применяться произвольным образом, а должна использоваться только в зоне локальных оптимумов. Кроме того, лучшие перемещения все еще используются для управления процессом поиска, хотя

и штрафуются, причем штрафы имеют ограниченный период действия. Локальный оптимум, достигаемый поиском табу при использовании данного подхода и используемый для начала последовательности шагов диверсификации, естественно, может отличаться от истинных локальных оптимумов ввиду того, что правила выбора поиска табу могут просто исключить некоторые перемещения, которые могли бы привести к улучшению.

Успех этого подхода - заслуга включения варианта TS, который всегда продолжает двигаться к локальному оптимуму, как только улучшающее перемещение становится альтернативой, которую можно принять, основываясь на критерии аспирации (привлекательности), который активируется только после выполнения улучшающего перемещения. В этом подходе так долго, пока существуют улучшающие перемещения, критерий привлекательности позволяет одному из них быть выбранным путем оценивания табу, которое штрафует выборы, основываясь на их состоянии табу (заостряя внимание на улучшающем множестве). Как только достигается истинный локальный оптимум, специальный критерий привлекательности прерывается пока стандартными правилами TS не будет выбрано новое улучшающее перемещение. Этот подход воплощает понятие *привлекательности направления поиска*.

Стратегическая осцилляция тесно связана с основами TS и обеспечивает положения для получения эффективного взаимодействия между ускорением и замедлением в течение длительного промежутка времени. Подход работает путем ориентации перемещений относительно *критического уровня*, который определяется как состояние конструкции или выбранный интервал значений функционала.

Такой критический уровень часто представляется точкой, где метод должен нормально останавливаться. Однако вместо того, чтобы останавливаться тогда, когда этот уровень достигается, правила для выбора перемещений изменяются для того, чтобы разрешать пересечение области, определенной критическим уровнем. Метод тогда продолжает работу до определенной глубины вне пределов критического уровня, и его работа возобновляется. Затем метод снова приближается к критическому уровню и пересекает его с противоположного направления, и метод переходит к новой отметке разворота.

Процесс неоднократного приближения и пересечения критического уровня с различных направлений порождает “колебание” стратегии метода, которое и дало методу имя. Управление этим поведением устанавливается путем генерации изменяемых оценок и правил движения в зависимости от управляемой области и направления поиска. Стандартные механизмы TS избегают возможности повторного рассмотрения предшествующей траектории.

Такие изменения правил, основанные на направлении и фазе поиска, - типичные особенности стратегической осцилляции и обеспечивают расширенную эвристику, называемую “жизнеспособностью”. Приложение различных правил может сопровождаться пересечением границы до различной глубины на различных сторонах. Эта возможность предназначена, чтобы приближаться к границе и отступать от нее без пересечения, то есть выбирая пересечение "нулевой глубины". Говорят о конструктивном и деструктивном типе стратегической осцилляции, когда конструктивные шаги "добавляют" элементы, а деструктивные шаги эти элементы "удаляют". Как только конструктивная фаза, состоящая из ряда операций "добавить перемещения", завершается, выполняется наиболее привлекательное действие - "удалить перемещение". Однако TS структуры памяти также необходимы, чтобы гарантировать чередующиеся фазы от действительной отмены друг друга.

При условной оптимизации осцилляция может также функционировать, увеличивая и уменьшая пределы нарушенности для ограничений. Такой подход стал основой для большого числа эффективных приложений, где пределы представляется такими понятиями, как значения целевой функции и допустимость решения для того, чтобы вести поиск и исследовать на различную глубину связанные области.

Полезная интеграция стратегий ускорения и замедления происходит в подходе, называемом *повторным компонованием маршрута*. Этот подход генерирует новые решения, исследуя траектории, которые "соединяют" элитные решения - начиная с одного из этих решений, называемого *решением инициализации*, и генерируя маршрут в пространстве окрестностей, который ведет к другим решениям, вызываемым *руководящими решениями*. Это выполняется путем выбора перемещений, которые представляют атрибуты, содержащиеся в руководящих решениях.

Подход может рассматриваться как крайний пример стратегии, которая пытается включить атрибуты решений высокого качества, создавая побуждения для предпочтительности этих атрибутов в выбранных перемещениях. Однако вместо того, чтобы использовать побуждение, которое просто поощряет включение таких атрибутов, подход повторной компоновки маршрута подчиняет все другие рассмотрения цели выбора перемещений, которые представляют атрибуты руководящих решений для того, чтобы создать "хорошую композицию атрибутов" в текущем решении. Композиция при каждом шаге определяется путем выбора лучшего перемещения, используя обычные критерии выбора из ограниченного набора перемещений, которое включает максимальный количество (или максимальное взвешенное значение) атрибутов руководящих решений. Как и в других приложениях TS, критерии привлекательности могут отменять это

ограничение, чтобы разрешать рассмотрение других перемещений особенно высокого качества.

Метод имитации отжига

Еще один подход, появившийся недавно и интенсивно развивающийся в настоящее время - имитация отжига, выглядит многообещающим и позволяет эффективно решать многие задачи на дискретных и комбинаторных структурах. Рассмотрим коротко основные идеи этого подхода.

Будем рассматривать задачу комбинаторной оптимизации в виде пары (D, f), где D - некоторая область допустимых решений (конфигураций), а f - критерий качества (функция стоимости), которая и должна быть оптимизирована выбором конфигурации из допустимого множества. Решить задачу (D, f) - значит найти решение, для которого $f = f^*$:

$$f^* = \min_{X \in D} f(X).$$

С точки зрения структуры алгоритм имитации отжига основан на двух областях науки - статистической физике и локальном поиске.

Основными понятиями локального поиска являются конфигурация X (одно решение), целевая функция f и механизм генерации новой структуры (переход от одной структуры к другой). Механизм генерации предполагает наличие системы окрестностей N(X), которая и определяет все конфигурации, достижимые из текущей конфигурации X за один переход. Процедуры локального спуска, его достоинства и недостатки, а также пути преодоления недостатков здесь мы обсуждать не будем. Мультистарт, усиление системы окрестностей за счет учета информации, полученной в процессе спуска, поиск с переменной длиной шага - примеры подходов, направленных на обеспечение глобальной сходимости локального спуска.

Еще одним путем может быть обеспечение возможности перехода в "худшие" точки с тем, чтобы затем спуститься в зону притяжения другого (лучшего) локального минимума. Этот подход обсуждался ранее для глобальной оптимизации в случае непрерывных функций (метод подъема к перевалу и метод тяжелого шарика), однако был обоснованно забыт из-за полной его неэффективности. В таком виде этот подход и не мог быть эффективным из-за полной непредсказуемости топологии оптимизируемой функции. Однако в начале 80-х годов С. Киркпатрик и В. Черны независимо друг от друга предложили использовать для этого так называемый критерий Метрополиса. Полученный таким образом алгоритм стал называться алгоритмом имитации отжига (simulated annealing) по аналогии с термодинамическим процессом закаливания металлов. Решения, получаемые этим алгоритмом, имеют хорошее приближение к глобальному минимуму и не зависят от выбора начальной точки поиска, что является одним из главных недостатков локального спуска.

Рассмотрим термодинамический процесс. При расплавлении металла все его частицы движутся свободно и случайным образом. Это состояние

вещества близко к полному хаосу и характеризуется высокой температурой и энтропией. При медленном охлаждении металла его структурные частицы выстраиваются в строгую кристаллическую решетку. Это состояние соответствует некоторому минимуму внутренней энергии тела, причем, чем выше начальная температура и медленнее охлаждение, тем меньше этот минимум. При каждой температуре T вещество достигает термического равновесия, которое характеризуется вероятностью пребывания системы в состоянии с внутренней энергией E, определяемой распределением Больцмана:

$$\Pr\{e=E\} = \frac{1}{Z(t)} \cdot \exp\left(-\frac{E}{K_B \cdot T} \right),$$

где Z(t) - нормирующий фактор, зависящий от T, второй сомножитель - фактор Больцмана, K_B - постоянная Больцмана.

При быстром охлаждении происходит закаливание - аналог локального поиска - "жадный" процесс, сваливающийся к ближайшему локальному минимуму с относительно высоким значением внутренней энергии.

При определенной температуре тела T и определенном состоянии у некоторой произвольной структурной частицы появляется возможность переместиться, что привело бы систему в другое состояние. При этом, если разность энергий этих состояний $\Delta E<0$, то частица перемещается и процесс продолжается уже из нового состояния, а если $\Delta E>0$, то вероятность перехода в новое состояние равна:

$$P = \exp\left(\frac{-\Delta E}{K_B \cdot T} \right).$$

При большом количестве переходов распределение этой вероятности стремится к распределению вероятности Больцмана, указанной выше. Идея алгоритма основывается на том, что в реальной системе действительно время от времени происходят переходы в состояние с более высокой энергией.

При решении задач оптимизации на комбинаторных и дискретных структурах допустимые решения (конфигурации) являются аналогами состояний физической системы, целевая функция - аналогом внутренней энергии, специальный управляющий параметр c - аналогом температуры T. Неформально алгоритм имитации отжига может быть описан как последовательность шагов:

1. Для текущего допустимого решения X_i случайным образом выбирается некоторое решение $X_j \in N(X_i)$, где $N(X_i)$ – окрестность точки X_i.

2. Вычисляется $\Delta f_{ij} = f(X_i) - f(X_j)$.

3. Переход из допустимого решения X_i в допустимое решение X_j осуществляется с вероятностью p_{ij}, где

$$p_{ij} = \begin{cases} 1, \text{ если } \Delta f_{ij} \le 0, \\ \exp\left(-\dfrac{\Delta f_{ij}}{\mathbf{c}}\right), \text{ если } \Delta f_{ij} > 0, \end{cases}$$

затем процесс продолжается с шага 1 до достижения квазиравновесия.

4. После достижения квазиравновесия параметр c уменьшается и шаги 1-3 повторяются до достижения квазиравновесия уже при другой "температуре" системы с.

5. Шаги 1-4 повторяются до тех пор, пока не будет достигнуто некоторое "замороженное" состояние, при котором вероятность перехода в любое соседнее состояние близка к нулю. Такая точка и принимается в качестве глобального минимума.

Более формально обобщенный алгоритм имитации отжига может быть описан на псевдо-Паскале следующим образом:

```
begin
   инициализация;
   M := 0;
   repeat
  repeat
    переход(Xi → Xj, Δfij)
         if Δfij ≤ 0 then перейти else
           if exp(-Δfij/c) > random(0,1) then  перейти;
         if перейти then принять(Xj);
  until состояние системы близко к равновесию;
   until критерий остановки ("замерзание" системы);
end.
```

При c = 0 алгоритм имитации отжига работает также, как локальный спуск со случайным порядком просмотра окрестностей.

Относительно области эффективного применения алгоритма можно сказать, что его работа в случае существенно полимодальной функции ("лисьи норы" Шекеля) и при несвязной допустимой области оставляет желать лучшего. Кроме того, "замерзание" поиска в точке глобального минимума гарантируется при бесконечном времени, а реально алгоритм приходится останавливать после истечения определенного срока, то есть последняя точка может оказаться не только отличной от искомой, но и худшей, чем уже найденные в процессе поиска. Поэтому необходимо отслеживать эти возможности и уточнять (при необходимости) положение

глобального минимума после остановки алгоритма имитации отжига с помощью какого-либо локального спуска.

Алгоритм муравьиной колонии (автокаталитический алгоритм комбинаторной оптимизации)

Алгоритм муравьиной колонии (Ant Colony System) – универсальный аналитический алгоритм, который может быть использован для решения многих комбинаторных задач. Автокаталитический означает сам себя усиливающий.

Преимущества:

- Многосторонность – может быть использован при решении различных задач и при различных постановках одной и той же задачи.
- Робастность – может применяться к произвольным комбинаторным задачам оптимизации при минимальных изменениях структуры.
- Использует популяцию решений – происходит использование положительных обратных связей для поиска решения, может использоваться на параллельных компьютерах.

Муравьи живут и наблюдают мир в одной плоскости. Они практически слепые и глухие. Однако в поисках еды они способны находить кратчайший путь к источнику пищи. Каким образом это у них получается?! Муравьи выделяют специальное вещество – феромон, которое имеет сильный, устойчивый запах. Обычно муравьи ориентируются и перемещаются по запаху оставленному другими муравьями. Т.о. получается, что по пути, по которому прошло большее количество муравьев, останется больше следа (запаха), и новые муравьи пойдут по этому пути.

Пример (рис. 1). Пусть первоначально муравьи находятся в точке A. Еда находится в точке E. Расстояния BD и DE равны 1, а BC и CE равны 0.5. В точке B муравьи разделятся поровну. В то время когда одна часть муравьев (15 муравьев) пройдут расстояние BD, другая часть успеет прийти BC и CE и пойдет обратно. Когда муравьи, выбравшие путь BCE вернуться назад, другие муравьи только дойдут до точки E, а когда они вернутся, другие 15 муравьев пройдут путь BCE туда и обратно еще раз. Т.о., на пути BDE оставят след (феромон) 15 муравьев, а на пути BCE – 30 муравьев (15 муравьев пройдут два раза). Теперь в точке B большая часть муравьев выберет путь BC.

Идея автокаталитического алгоритма состоит не в имитации поведения одного муравья, а использовании идеи поведения колонии в целом. Искусственные муравьи живут в искусственной среде с дискретным временем. Они наделены памятью – не являются полностью слепыми.

Рассмотрим идею алгоритма ant-cycle на примере задачи коммивояжера (далее ЗК).

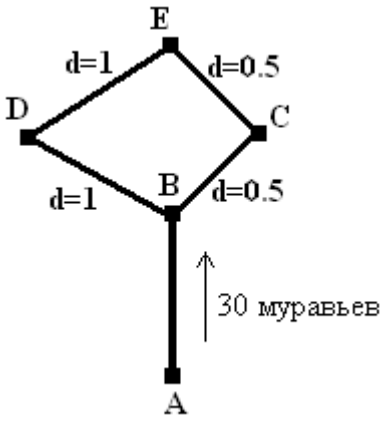


Рис. 1. Пример поведения колонии из 30 муравьев.

Пусть даны *n* городов. d_{ij} - расстояния между городами (длины ребер в графе), $d_{ij} \neq d_{ji}$. Необходимо найти замкнутый обход всех городов, имеющий минимальную длину. Каждый город должен быть посещен только один раз.
Например, (N, E), где N - количество городов, E - множество ребер, соединяющих города.
Обозначим общее количество муравьев m, $b_i(t)$ - количество муравьев в городе i в момент времени t, т.е. $\sum_{i=1}^{m} b_i(t) = m$.

Каждый муравей является простым агентом со следующими свойствами:

- Он выбирает город, в который собирается перейти с вероятностью, которая является функцией расстояния до города и количества следов на соединяющем ребре.
- Переходы в уже посещенные города запрещены, пока обход не завершен. Это контролируется с помощью списка табу.
- Когда получен полный обход, муравей оставляет след на каждом пройденном ребре ij.

$\tau_{ij}(t)$ - интенсивность следа на ребре ij в момент времени t.
M ходов, выполненные m муравьями за время $(t, t+1)$ называется итерацией. За n итераций все муравьи совершат один обход – цикл алгоритма.

После завершения цикла:
$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$, где $0 < \rho < 1$ - коэффициент испарения следа.

$\Delta\tau_{ij} = \sum_{i=1}^{m} \Delta\tau_{ij}^{k}$, где $\Delta\tau_{ij}^{k}$ - количество следа, оставленного не единицу длины ребра ij k-м муравьем.

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, \text{если } k-\text{ый муравей использует ребро } ij \\ 0, \text{ в противном случае} \end{cases}.$$

Q - константа, параметр алгоритма.

L_k - длина обхода k-ого муравья.

Для того, чтобы исключить не ограниченное накопление следа, каждому муравью ставится список ТАБУ. После обхода, список обнуляется.

Обозначим:

$tabu_k$ - список табу у k-ого муравья.

$\eta_{ij} = \dfrac{1}{d_{ij}}$ - видимость для ребра ij.

p_{ij}^{k} - вероятность перехода из точки i в точку j для k-ого муравья.

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}^{k}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \notin tabu_k} [\tau_{ij}^{k}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}, j \notin tabu_k \\ 0, j \in tabu_k \end{cases} \quad (1)$$

α, β - параметры алгоритма, управляющие относительной важностью видимости и следа.

Алгоритм (Ant-cycle):

1. Инициализация. Муравьи размещаются в разных городах случайным образом. На ребрах задаются начальные значения следа $\tau_{ij}(0)$. В список табу ставится номер исходного города, обозначим c_i.
2. Построение решений: для каждого муравья k=1,...,m.

2.1. Обозначим текущее частичное решение муравья k на итерации t через y^k. $y^k = \varnothing$.

2.2. Муравей переходит в город j из города i с вероятностью (7.1).

2.3. Если город j выбран, то $y^k = y^k \cup c_j$. Выбранный город заносится в список табу.

2.4. Если решение y^k не завершено, перейти к шагу 2.2.

3. После n итераций все муравьи совершат обход, и все списки табу будут полными. Для всех муравьев вычисляются длины обходов L_k. Кратчайший запоминается ($\min_k\{L_k\}$).

4. Вычисляются $\Delta\tau_{ij}^k$ и $\tau_{ij}(t+1)$. Списки табу обнуляются.

5. Если не конец, то повторить 2-5.

Вероятность перехода в (7.1) является функцией привлекательности, состоящей из двух мер. Первая указывает, сколько муравьев прошло по дуге ij в прошлом. Другая говорит о том, что чем ближе город, тем он желателен.

Если параметр $\alpha = 0$, то получается «жадный» алгоритм со многими стартовыми точками, не учитывающий поведение других муравьев.

В качестве критерия остановки обычно выбирают:

- максимальное количество циклов;
- пока все муравьи не получат один и тот же обход (стагнация).

Существуют следующие модификации алгоритма Ant-cycle:

- Ant-density
- Ant-quantity

Алгоритмы различаются способом обновления следов. В Ant-density, если муравей за время $(t,t+1)$ перешел в j из i, то он оставит след Q (в алгоритме ant-cycle все следы обновляются в конце цикла). В алгоритме ant-density при переходе из i в j, муравей оставит след Q/d_{ij}.

Рекомендации:

- Лучший алгоритм – ant-cycle с $\rho = 0.5$.
- Оптимальное число муравьев в колонии $m \approx n$.

Комбинации параметров α, β дают результаты, показанные на рис. 2. Где **x** – плохое решение без стагнации, * - плохое решение и стагнация, + - хорошее решение без стагнации.

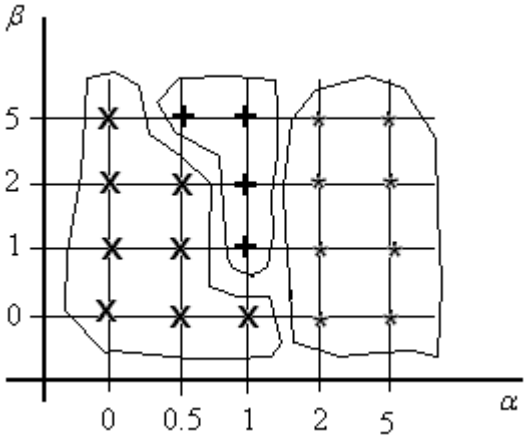


Рис. 2. Эффективность комбинации параметров

При установки оптимальных параметров алгоритм находит оптимальное решение и не попадает в стагнацию. Алгоритм слабо чувствителен к размерности задачи.

«Стайный» алгоритм (Particle Swarm Optimization)

Идею алгоритма Particle Swarm Optimization (PSO) впервые сформулировали Дж. Кеннеди и Д. К. Эберхартом в 1995 году. Идея алгоритма была почерпнута из социального поведения некоторых животных – стаи птиц, стада копытных или косяка рыб. PSO аналогичен генетическому алгоритму – он начинает с создания популяции случайным образом. Но, в отличие от ГА, у PSO нет таких эволюционных операторов, как скрещивание и мутация. Строки в PSO называются частицами, тогда как у ГА это хромосомы. Строки-частицы представляют собой вектор координат точки в пространстве оптимизации (вещественных чисел), в отличие от ГА, хромосомы которого являются бинарным кодом. Каждая частица передвигается по поверхности графика функции с какой-то скоростью. Частицы изменяют свою скорость и координаты, основываясь на собственном опыте и опыте других частиц, что выражается формулами:

$$V_{m,n}^{new} = V_{m,n}^{old} + \Gamma_1 * r_1 * (p_{m,n}^{local\_best} - p_{m,n}^{old}) + \Gamma_2 * r_2 * (p_{m,n}^{global\_best} - p_{m,n}^{old}),$$

$$p_{m,n}^{new} = p_{m,n}^{old} + V_{m,n}^{new},$$

где $V_{m,n}$ - скорость частицы, $p_{m,n}$ - координаты частицы, r_1, r_2 - независимые случайные числа, распределенные по равномерному закону, Γ_1, Γ_2 - коэффициенты обучения, $p_{m,n}^{local\_best}$ - лучшие координаты, когда-либо найденные данной частицей, $p_{m,n}^{global\_best}$ - лучшие координаты, когда-либо найденные всей стаей.

Алгоритм PSO сначала обновляет вектор скорости каждой частицы, а затем прибавляет этот вектор к вектору координат соответствующей частицы. Обновление вектора скорости осуществляется с учетом наилучшего глобального решения, соответствующего наименьшему значению минимизируемой функции, когда-либо найденному всей стаей, и с учетом наилучшего локального решения, соответствующего наименьшему значению функции, когда-либо найденному данной частицей популяции. Если наилучшее локальное решение имеет значение функции, меньшее, чем значение функции в глобальном наилучшем решении, то оно становятся лучшим, когда-либо найденным решением для всей стаи. Использование скорости частицы в каком-то смысле напоминает об алгоритмах локальной оптимизации, использующих информацию о производных, т.к. скорость - это производная от координат. Константа Γ_1 называется когнитивным (т.е. познавательным) параметром и позволяет учитывать «собственный опыт» (историю) частицы. Константа Γ_2 называется социальным параметром и позволяет частице «учитывать опыт» всей стаи. Преимущество PSO заключается в простоте реализации и в наличии малого количества

параметров, требующих настройки. PSO способен работать со сложными целевыми функциями, имеющими множество локальных минимумов.

Обычно оптимизация алгоритмом PSO выглядит следующим образом. Частицы, которые сначала равномерно распределены по поверхности функции, с течением времени (от поколения к поколению) начинают группироваться («сбиваться в стаи») около локальных минимумов, причем наибольшая стая собирается около глобального минимума. При этом почти всегда имеются частицы, находящиеся в стороне от таких стай, а также частицы, выскакивающие за границы допустимой области.

Исследование эффективности стайного алгоритма проводилось на достаточно сложных тестовых функциях двух независимых переменных – функции Розенброка и функции Гриванка.

Первая функция, хотя и одноэкстремальная, имеет длинных изогнутый овраг, что значительно затрудняет отыскание точки минимума алгоритмам математического программирования, как использующим информацию о производных, так и задействующим детерминированный прямой поиск. Вторая функция – многоэкстремальная с узкими зонами притяжения локальных минимумов, которые незначительно отличаются друг от друга по глубине, что делает такую функцию исключительно сложной для оптимизации. Обе функции входят в стандартный набор тестовых функций для поисковых алгоритмов.

Алгоритм PSO практически не имеет настраиваемых параметров, поэтому основной интерес вызывал вопрос о размере и распределении ресурса, приводящем к наилучшим результатам. Ресурс – это общее количество вычислений целевой функции, которое разрешено сделать алгоритму для отыскания оптимума. Данный ресурс изменялся от 100 до 100 тысяч. Распределение ресурса изменялось от ситуации, когда незначительное число частиц эволюционирует длительное время (например, 10-1000), до ситуации, когда большое число частиц эволюционирует незначительное по длительности время (100-10). Промежуточные варианты, разумеется, тоже рассматривались.

Таблица 1

Исследование эффективности стайного алгоритма на функции Гриванка

	10	25	50	75	100	200	300	400	500	600	700	800	900	1000
10	0	0	0,001	0,007	0,017	0,133	0,356	0,524	0,661	0,8	0,916	0,968	0,995	0,999
20	0	0	0,002	0,017	0,065	0,393	0,604	0,773	0,918	0,981	0,994	1	1	1
30	0,001	0,004	0,01	0,032	0,124	0,562	0,753	0,914	0,988	0,998	1	1	1	1
50	0,004	0,004	0,019	0,124	0,278	0,777	0,927	0,975	0,996	1	1	1	1	1
70	0,002	0,003	0,030	0,195	0,418	0,887	0,963	0,996	1	1	1	1	1	1
100	0,003	0,006	0,065	0,341	0,628	0,942	0,993	1	1	1	1	1	1	1

По таблице 1 можно определить необходимый размер и распределение ресурса (100000 вычислений функции достаточно в любом случае, распределение лучше то, когда небольшое число частиц эволюционируют достаточно долго). В ячейках таблицы 1 приведена надежность отыскания экстремума тестовых функций стайным алгоритмом.

Были проведены также эксперименты с альтернативной адаптацией стайных алгоритмов, показавшие, что выбор более эффективного из них (по распределению ресурсов) может быть установлен с намного меньшими вычислительными затратами, чем приведенные в таблице 1. Использовалась альтернативная адаптация с линейной тактикой. Представляется очевидным, что затраты на установление лучшего алгоритма могут быть еще больше сокращены при использовании нелинейной тактики.

Сравнение эффективности стайного алгоритма и стандартного генетического алгоритма проводилось на тех же функциях, что и исследование эффективности алгоритма PSO. Для стандартного ГА заранее были выбраны оптимальные настройки (турнирная селекция, равномерное скрещивание, средняя мутация). Так как стайный алгоритм такой настройки не требует, сравнение алгоритмов проводилось по различным распределениям выделенного ресурса (10 тыс. вычислений для функции Гриванка и 2 тыс. вычислений для функции Розенброка). Результаты сравнения эффективности приведены в таблицах 2 и 3. Здесь в первой строке таблицы приведено распределение ресурса (например, 50 индивидов и 200 поколений), а во второй и в третьей – надежность алгоритмов.

Анализ таблиц показывает, что при правильном распределении ресурса (минимум частиц – максимум поколений) стайный алгоритм значительно превосходит в обоих случаях даже оптимально настроенный стандартный ГА, независимо от того, как он распределяет ресурсы.

Таблица 2. Сравнение на функции Гриванка

100-100	50-200	20-500	10-1000
Particle Swarm Optimization			
0, 782	0, 907	0, 989	1
Генетический Алгоритм			
0, 310	0, 291	0, 416	0, 444

Таблица 3. Сравнение на функции Розенброка

100-20	50-40	40-50	20-100	10-200
Particle Swarm Optimization				
0,003	0,009	0,393	0,409	0,946
Генетический Алгоритм				
0,2855	0,474	0,393	0,350	0,394

Бинарный алгоритм PSO

Первоначально PSO был создан для задач с вещественными переменными. В настоящий момент использование алгоритмов расширилось вплоть до дискретных задач и задач с бинарными переменными. При расширении версии PSO, работающей с вещественными переменными, в бинарное/дискретное пространство наиболее важным является определить смысл таких понятий, как траектория и скорость в бинарном/дискретном пространстве.

Кеннеди и Эберхарт (Kennedy, Eberhart 1997) используют скорость как вероятность для определения, будет ли x_{id} (некоторый бит) находиться в том или ином состоянии (1 или 0). Они преобразовывали v_{id} с помощью логической функции $s(v)=1/(1+exp(-v))$, вычисляя скорость по тому же самому уравнению, что и в обычном алгоритме PSO:

$$v_{id} = v_{id} + c_1 \cdot rand() \cdot (p_{id} - x_{id}) + c_2 \cdot Rand() \cdot (p_{gd} - x_{id}) . \quad (1а)$$

Если случайно сгенерированное число из интервала [0;1] меньше, чем $s(v_{id})$, то x_{id} устанавливается равным 1, в противном случае - 0. Версия бинарного PSO работает лучше, чем некоторые версии ГА на всех тестовых задачах кроме одной. (Кеннеди и Спирс (Kennedy, Spears) 1998).

Аграфиотис и Цедено (Agrafiotis, Cedeno 2002) адаптировали PSO, работающий с вещественными переменными, к бинарному пространству и применили его в задаче выбора признаков (feature selection problem), в которой x_{ij} может принимать только значения 0 или 1 и означает выбирается ли j-й признак i-й частицы или нет. Вещественное значение, вычисляемое с использованием уравнения

$$x_{id} = x_{id} + v_{id} , \quad (1б)$$

трактуется как вероятность, с которой колесо рулетки используется, чтобы определить выбирается соответствующий признак в следующее поколение или нет.

Мохан и Ал-каземи (Mohan, Al-kazemi 2001) предложили 5 бинарных вариантов алгоритмов PSO. Методы, которые они использовали для этих пяти бинарных PSO, названы прямым методом, квантовым методом, методом с регуляризацией, методом вектора смещения и методом смешанного поиска, соответственно. Прямой метод бинарного PSO – это прямой перевод глобальной версии PSO из непрерывного пространства в бинарное пространство. Позиция частицы является решением-кандидатом, используются уравнения 1а и 1б. Результаты в непрерывном пространстве превращаются в бинарные строки с использованием некоторого метода исправления, например процесса декодирования для жестких решений (да-нет). Квантовый метод бинарного PSO подобен прямому методу PSO, за исключением того, что используется другой метод исправления, при котором полученное (вещественное) значение преобразуется в 0 или 1 с

вероятностью, определяемой самим этим значением. Метод с регуляризацией создан в расчете на то, что он объединит PSO с другими эволюционными алгоритмами при использовании абстрактно выраженного PSO в качестве примера регуляризации (ссылка 6 в работе Мохана и Ал-каземи (Mohan, Al-kazemi) 2001). В методе вектора смещения, частицы в новом поколении случайно выбирают из трех частей в правой части уравнения (1a) с вероятностями, зависящими от пригодности значений соответствующих позиций. Метод смешанного поиска схож с прямым методом за исключением того, что частицы разделяются на большее число групп, каждая из которых динамически использует выбирать локальную и глобальную версию PSO.

Топологии

Обычно используется одно из двух – глобальная или локальная версия PSO. В глобальной версии PSO каждая частица «летает» по поисковому пространству со скоростью, которая динамически настраивается в соответствии с наилучшим, достигнутым этой частицей, на данный момент значением, и наилучшим, достигнутым всеми частицами, значением на данный момент. В локальной версии PSO скорости каждой из частиц регулируются согласно их собственному наилучшему на данный момент и наилучшему, достигнутому на данный момент частицами из ее окрестности. Окрестность каждой из частиц в общем случае определяется как множество частиц, топологически ближайших к данной частице с каждой стороны. Глобальная версия PSO также может рассматриваться как локальная версия PSO, где окрестностью каждой частицы является вся популяция. Предполагалось, что локальная версия PSO сходится быстро, но скорее всего к локальному минимуму, в то время как локальная версия PSO может иметь больше шансов найти лучшее решение медленно (Кеннеди (Kennedy) 1999, Кеннеди, Эберхарт и Ши (Kennedy, Eberhart, Shi) 2001). С тех пор множество исследователей работали над улучшением его исполнения дизайном или выполнением различных типов соседских структур в PSO.

Кеннеди (Kennedy 1999) заявил, что PSO с маленьким соседством может работать лучше на комплексных проблемах, в то время как PSO с большим соседством будет работать лучше на простых проблемах.

Кеннеди и Мендес (Kennedy, Mendes 2002) протестировали PSO с возможностью регулирования определения соседей, таких как глобальная версия, локальная версия, структура в виде пирамиды, структура в виде звезды, «маленькая» структура, и фон Неуман, и PSO со случайной генерацией соседей. Размер популяции этих PSO фиксирован и равняется 20. Они проводили наблюдения над этими PSO со случайной генерацией соседей, эти PSO со средним количеством соседей -5 имеют лучшее исполнение, основанное на их системе мер. Они поддерживают рекомендации PSO со структурой соседей фон Неумана, которое может

работать лучше, чем PSO с другими регулировками определения соседей, включая глобальную версию и локальною версию.

Сугантан (Suganthan 1999) занимался комбинацией версий PSO, где локальная версия PSO впервые работает, следуя за глобальной версией PSO в конце его работы. К тому же, в локальной версии PSO, соседство каждой частицы динамично регулируется. Дистанция между частицами считается и затем используется как эталон параметров формы новых соседей с предопределенным критерием.

Ху и Эберхарт (Hu, Eberhart 2002 a) ввели понятие динамичного соседства для их проблем многоцелевой оптимизации используя PSO. Соседство каждой частицы динамично регулируется. m ближайших частиц в исполнительном пространстве отбираются чтобы быть этим новым соседством в каждом поколении. Исполнительное пространство в пространстве, где каждая координата – переменная, означающая исполнительное значение каждой целевой функции проблемы многоцелевой оптимизации.

Мендес и Кеннеди (Mendes, Kennedy 2004) предложили полностью осведомленный алгоритм PSO, основанный на анализе коэффициента φ и их вера в то, что там нет присвоения, что лучший сосед на самом деле находит лучший регион, затем двух или трех лучших соседей. В этом новом алгоритме все соседи частицы сложны в нахождении следующего перемещения вместо использования лучшей предыдущей позиции в оригинальном PSO. Влияние каждой частицы на их соседей является авторитетом, основанном на ее значении пригодности и размере соседства.

Каждая структура соседства имеет свое преимущество и свой недостаток. Это работает на одном виде проблем, но хуже на остальных видах проблем. Когда PSO используется для разрешимых проблем, проблемы не нуждаются в абсолютно точном определении, но структура соседства PSO также должна использоваться чтобы быть очевидно точно определенной.

Параметры

Изменения скорости PSO состоят из трех частей: «социальная» часть, «когнитивная» часть и инерционная часть. Соотношение между этими частями определяет соотношение между глобальными и локальными свойствами поиска, а, следовательно, и эффективность (работоспособность) PSO.

Первый новый параметр, добавленный в исходный алгоритм PSO – это инерционный вес (весовой коэффициент инерции) (Ши и Эберхарт (Shi, Eberhart) 1998 a, 1998 b). Динамическое уравнение PSO с инерционным весом имеет вид:

$$v_{id} = w \cdot v_{id} + c_1 \cdot rand() \cdot (p_{id} - x_{id}) + c_2 \cdot Rand() \cdot (p_{gd} - x_{id}) \qquad (2a)$$

$$x_{id} = x_{id} + v_{id} \quad (2b)$$

Уравнения (2) – это то же самое, что и уравнения (1) кроме нового параметра – инерционного веса *w*. Инерционный вес вводится для балансирования между глобальным и локальным характером поиска. Большой инерционный вес способствует глобальному поиску в то время, как маленький инерционный вес способствует локальному поиску. Введение инерционного веса также исключает требование точного определения максимальной скорости v_{max} каждый раз при использовании алгоритма PSO. V_{max} может быть просто установлено равным величине динамического размаха каждой переменной, при этом алгоритм PSO работает достаточно хорошо, если не лучше.

Другой параметр, называющийся коэффициент сжатия, применяется в надежде, что это может гарантировать сходимость PSO (Клерк 1999 (Clerc), Клерк и Кеннеди (Clerc, Kennedy) 2002). Упрощенный метод введения коэффициента сжатия показан в уравнении (3), где *k* – это функция от c_1 и c_2, как видно из уравнения (4).

$$v_{id} = k[v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})] \quad (3a)$$

$$x_{id} = x_{id} + v_{id}, \quad (3b)$$

при

$$k = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4 * \varphi}|}, \quad (4)$$

где $\varphi=c_1+c_2$, $\varphi>4$.

Математически, уравнения (2) и (3) эквивалентны установлению инерционного веса *w* равного *k*, при этом c_1 и c_2 удовлетворяют условию $\varphi=c_1+c_2$, $\varphi>4$. Алгоритм PSO со сжимающим множителем может рассматриваться как специальный случай алгоритма PSO с инерционным весом в то время, как три параметра связаны уравнением (4). Более лучшим подходом, используемым как эвристическое правило, является применение PSO со сжимающим множителем, ограничивая V_{max} величиной X_{max}, размахом каждой переменной по каждому измерению, или применение PSO с инерционным весом, выбирая *w*, c_1 и c_2 в соответствии с уравнением (4) (Эберхарт и Ши (Eberhart, Shi) 2000).

Когда стягивающий метод Клерка используется, φ обычно становится 4,1 и постоянный коэффициент *k* приблизительно равен 0,729. Это эквивалентно PSO с инерционным весом $w\approx0{,}729$ и $c_1=c_2=1{,}49445$. Так как процесс поиска алгоритма PSO нелинейный и сложный, PSO с хорошо выбранными параметрами может хорошо работать, но еще более эффективная работа может быть достигнута, если хорошо спланировать динамический изменение параметров. Интуитивно, PSO должен

предпочитать глобальные свойства поиска в начале работы и локальные свойства в конце.

Ши и Эберхарт (1998 a, 1999) первыми ввели в PSO линейной убывающий по ходу работы инерционный вес, затем они спроектировали нечеткие системы нелинейного изменения инерционного веса (Ши и Эберхарт 2001 a, 2001 b). Нечеткие системы имеют на входе несколько измерений меры эффективности работы PSO, а на выходе – новый инерционный вес. В более поздних работах использовался инерционный вес со случайным сомножителем [0,5 + (rand()/2.0)], а не убывающий со временем. Это дает случайное число между 0,5 и 1,0, со средним значением 0,75, которое похоже на сжимающий множитель Кларка, описанный выше.

Ратнавира, Халгамуг и Ватсон (Ratnaweera, Halgamuge, Watson 2004) ввели в PSO изменяющийся по времени коэффициент ускорения в дополнение к изменяющемуся во времени инерционному весу. В дальнейшем предлагался PSO, названый «Самоорганизующийся иерархический PSO, в котором только остаются только «социальная» и «когнитивная» составляющие, как в (Хи и др. 1998), а инерционная составляющая используется только для повторной ре-инициализации частиц, когда частицы застаиваются в поисковом пространстве (впадают в стагнацию), т.е. этот инерционный вес равен нулю всегда, кроме момента ре-инициализации. В работе (Фан и Ши (Fan, Shi) 2001), вводится линейно уменьшающееся $V_{\max}$, как уже упоминалось выше.

Задачи условной оптимизации

Условная оптимизация – это одна из наиболее общих областей применения PSO. Одни из главных вопросов при решении задач условной оптимизации является вопрос о том, что делать с ограничениями, как их учитывать. Прямолинейным подходом является преобразование задачи условной оптимизации в задачу безусловной оптимизации путем добавления штрафа за нарушение ограничений (Парсопоулос и Врахатис (Parsopoulos, Vrahatis) 2002b). Другим методом является сохранение допустимых решений и исправление недопустимых (Ху и Эберхарт (Hu, Eberhart) 2002b). Третий метод, называющийся гибридным алгоритмом, обычно задействует некоторую стратегию декодирования информации. Например, в (Рэй и Лью (Ray, Liew) 2001), ограничения управляются некоторой матрицей ограничений. Матрица ограничений используется для генерирования списка более лучших исполнителей (better performer list, BPL), который используется для настройки направления поиска остальных частиц.

Метод обобщенного локального поиска для решения задач глобальной оптимизации

Рассмотрим две гибридные схемы глобальной оптимизации псевдобулевых функций.

Первой из них является комбинация генетического алгоритма с локальным спуском. Выбирается именно генетический алгоритм, а не эволюционные стратегии, в связи с тем, что на множестве булевых переменных между этими подходами нет большой разницы - записью для индивидуума является битовая строка как на уровне фенотипа, так и на уровне генотипа, а мутации в обоих случаях имеют один и тот же характер (замена нуля на единицу или наоборот). Теоретически при достаточно долгой работе генетического алгоритма должно улучшаться качество популяции, то есть среднее значение целевой функции по популяции и значение в лучшей точке должны уменьшаться. Однако время работы алгоритма будет ограничено и поэтому сходимость алгоритма не будет реализовываться в полной мере. В этой связи после остановки генетического алгоритма есть смысл в качестве стартовой точки локального спуска выбирать не только лучшую точку последнего поколения, но и лучшие точки других поколений, которые должны для этих целей запоминаться и храниться в специальном файле. Параметрами гибридного алгоритма (кроме параметров генетического алгоритма и локального поиска) являются количество индивидуумов в поколении, количество поколений до запуска локального поиска (N) и количество стартовых точек для локального спуска (M). Гибрид генетического алгоритма и локального спуска может быть описан следующим образом.

Алгоритм ГАЛС.

1. Сгенерировать первое поколение I_1. Запомнить лучшего индивидуума X_1.

2. Определить множество I^* из (M-N) наилучших индивидуумов множества $I_1 \backslash X_1$ и их значения функции пригодности.

3. (k-й шаг) Выполнить шаги генетического алгоритма, порождающие поколение I_k.

4. Запомнить лучшего индивидуума этого поколения X_k.

5. Из множества $I^* \cup \{I_k \backslash X_k\}$ выбрать (M-N) наилучших индивидуумов, сформировать новое множество I^* и запомнить значения функций пригодности вошедших в него индивидуумов.

6. Если $k < N$, то положить $k = k+1$ и перейти к шагу 3.

7. Запустить локальный спуск из точек множества

$$\{X_1, X_2, ..., X_N\} \cup I^*.$$

8. Среди полученных точек локальных минимумов выбрать наилучшую и выдать ее в качестве глобального минимума.

Вообще говоря, алгоритм ГАЛС представляет собой целый класс алгоритмов, которые могут различаться не только параметрами M и N, но и используемыми базовыми алгоритмами, так как генетические алгоритмы могут быть весьма разнообразными, а алгоритмы локального спуска зависят от выбранной системы окрестностей, способа просмотра окрестности и тактики перехода из окрестности в окрестность. Ответ на вопрос, какая именно гибридная схема является наилучшей для конкретной задачи, не может быть дан априори и должен исследоваться отдельно в ходе численного эксперимента.

Алгоритм имитации отжига сочетает локальный спуск с обеспечением возможности перехода не только в лучшую, чем текущая, точку, но и в худшую (хотя и с малой вероятностью), чем и обеспечивается сходимость к глобальному минимуму. Теоретический результат исследований о сходимости алгоритма гласит, что процесс оптимизации "замерзнет" в точке глобального минимума при бесконечном времени, случайном выборе стартовой точки и правильно выбранном расписании "охлаждения". При решении реальных задач трудно давать рекомендации об эффективном расписании, а бесконечное время просто не доступно. В результате алгоритм может остановиться в точке, далекой от глобального минимума и даже от его зоны притяжения. Случайный выбор стартовой точки тоже затруднен, особенно при большой размерности задачи и сложной форме допустимой области, когда основное время тратится на установление методом Монте-Карло первой допустимой точки. Если указывать стартовую точку произвольно, то теряется гарантия глобальной сходимости даже при бесконечном времени. При несвязной или хотя бы невыпуклой допустимой области алгоритм имитации отжига не имеет возможности преодолеть разрывы или обойти выгнутые участки границы. Все это накладывает ограничения на использование этого подхода, поэтому необходимо модифицировать алгоритм.

Очевидной модификацией алгоритма имитации отжига будет сочетание его с локальным спуском из точки "замерзания" и наилучшей точки, полученной в процессе оптимизации (это может оказаться одна и та же точка). Такая модификация направлена на преодоление проблемы конечного времени оптимизации.

Более интересной может быть модификация, ориентированнная на преодоление несвязности допустимой области.

Алгоритм ИОЛС.

1. Методом Монте-Карло выбрать несколько стартовых точек.

2. Из недопустимых стартовых точек выполнить локальный спуск, минимизируя функцию обобщенного штрафа,

$$P(X) = r \cdot \sum_{j=1}^{m_1} g_j(X) + \rho \cdot \sum_{i=1}^{m_2} h_i^2(X)$$

и получить соответствующие им допустимые точки.

3. Из полученных на шагах 1-2 стартовых точек выполнить фиксированное число шагов алгоритма имитации отжига, запоминая наилучшую точку и значения функции в ней.

4. Если значения функции в различных процессах имитации отжига совпали, то сравнить точки, дающие эти значения, и в случае совпадения точек остановить соответствующий процесс.

5. Из всех точек, полученных после остановки процессов имитации отжига, включая и запоминаемые наилучшие точки (если они отличаются), выполнить локальный спуск.

6. Наилучшую точку, полученную в процессе оптимизации, выдать в качестве глобального минимума.

Предлагаемый алгоритм имеет те же гарантии сходимости, что и базовый алгоритм имитации отжига, но требует большего времени для расчетов, хотя и позволяет преодолеть такие недостатки, как остановка вдали от глобального минимума, чрезмерные затраты на получение допустимой стартовой точки и неспособность работать на несвязных допустимых областях. Дополнительные затраты на оптимизацию заставляют сделать вывод, что в случае простых целевых функций и связных допустимых областях должен применяться только базовый алгоритм.

В заключение отметим, что и для последнего алгоритма нельзя, к сожалению, указать априори, будет ли он эффективен для той или иной практической задачи, если, конечно, не будет заранее известно, что допустимая область несвязна. Очевидно также, что этот алгоритм ориентирован на параллельное выполнение, то есть скорость его работы существенно возрастет при работе на параллельных компьютерах.

На дискретной решетке задана целевая функция f(X) и функции-ограничения $g_j(X) \le 0$, $j = 1, \ldots, m_1+2 \cdot n$, $h_i(X)=0$, $i = 1, \ldots, m_2$, где естественные ограничения на переменные полагаем включенными в список ограничений-неравенств. Как и ранее предполагаем, что целевая функция и/или функции-ограничения могут быть заданы неявно в виде алгоритма, имитационной модели или реального процесса, то есть не имеют явного аналитического вида, позволяющего выполнить математический анализ и выявить их характер. Также как и для псевдобулевой оптимизации возможно генерирование стартовых точек с помощью адаптивных поисковых алгоритмов.

Проще всего выполнить модификацию алгоритма ИОЛС имитации отжига в сочетании с мультистартом и локальным спуском из точек "замерзания" и наилучших точек, полученных в процессе оптимизации. Такая модификация сводится, практически, к построению другой системы

окрестностей и изменению алгоритма генерирования новой точки. Кроме того, штрафная функция тоже должна быть модифицирована очевидным образом.

Алгоритм МИОЛС.

1. Методом Монте-Карло выбрать несколько стартовых точек.

2. Из недопустимых стартовых точек выполнить локальный спуск, минимизируя функцию обобщенного штрафа,

$$P(X) = r \cdot \sum_{j=1}^{m_1+2\cdot n} g_j(X) + \rho \cdot \sum_{i=1}^{m_2} h_i^2(X).$$

и получить соответствующие им допустимые точки.

3. Из полученных на шагах 1-2 стартовых точек выполнить фиксированное число шагов алгоритма имитации отжига, запоминая наилучшую точку и значения функции в ней.

4. Если значения функции в различных процессах имитации отжига совпали, то сравнить точки, дающие эти значения, и в случае совпадения точек остановить соответствующий процесс.

5. Из всех точек, полученных после остановки процессов имитации отжига, включая и запоминаемые наилучшие точки (если они отличаются), выполнить локальный спуск.

6. Наилучшую точку, полученную в процессе оптимизации, выдать в качестве глобального минимума.

Аналогично алгоритму ИОЛС данный алгоритм имеет те же гарантии сходимости, что и базовый алгоритм имитации отжига, но требует большего времени для расчетов, хотя и позволяет преодолеть такие недостатки, как остановка вдали от глобального минимума, чрезмерные затраты на получение допустимой стартовой точки и неспособность работать на несвязных допустимых областях и вне допустимого множества. Также очевидно, что в случае простых целевых функций и связных допустимых областей должен применяться только базовый алгоритм.

Следующими кандидатами на роль генераторов стартовых точек мультистарта являются генетический алгоритм и эволюционные стратегии. В случае оптимизации на дискретной решетке они уже не так похожи, как при псевдобулевой оптимизации. Поэтому есть смысл реализовать две гибридные схемы и экспериментировать с обеими.

Эволюционные стратегии при непрерывной оптимизации обладают недостаточной глобальной сходимостью, так как они изменяют объектные переменные посредством мутации, которая определяется нормальным распределением с нулевым матожиданием, то есть каждый индивид изменяется при мутации незначительно. Однако при дискретной оптимизации объектные переменные изменяются скачкообразно и, если случайно будут изменены несколько из них, новый индивид будет уже существенно отличаться от родителя. Таким образом, преимущество

генетического алгоритма в непрерывном случае - глобальная сходимость - при дискретной оптимизации не так уж и велико, а его недостаток - необходимость бинаризации переменных сохраняется. Бинаризация является недостатком, так как, во-первых, это лишняя операция и не такая простая, во-вторых, резко возрастает размерность решаемой задачи, в-третьих, нарушаются топологические свойства функции и появляются обширные зоны булевых точек, не имеющих физического смысла в исходном пространстве объектных переменных. Все это осложняет работу генетического алгоритма.

У эволюционных стратегий этот недостаток отсутствует. В целом же все рассуждения, приведенные в предыдущем параграфе для генетического алгоритма, сохраняют силу и в случае дискретной оптимизации, причем для обоих подходов. Практически описание обоих гибридных алгоритмов выглядит одинаково, если только помнить, что они работают с различными представлениями индивидов и выполняют преобразования по различным схемам. Так же как и в случае псевдобулевой оптимизации параметрами гибридного алгоритма являются количество индивидуумов в поколении, количество поколений до запуска локального поиска (N) и количество стартовых точек для локального спуска (M).

Алгоритм ЭГАЛС.

1. Сгенерировать первое поколение I_1. Запомнить лучшего индивидуума X_1.

2. Определить множество I^* из (M-N) наилучших индивидуумов множества $I_1 \backslash X_1$ и их значения функции пригодности.

3. (k-й шаг) Выполнить шаги генетического (эволюционного) алгоритма, порождающие поколение I_k.

4. Запомнить лучшего индивидуума этого поколения X_k.

5. Из множества

$$I^* \cup \{I_k \backslash X_k\}$$

выбрать (M-N) наилучших индивидуумов, сформировать новое множество I^* и запомнить значения функций пригодности вошедших в него индивидуумов.

6. Если $k < N$, то положить $k = k+1$ и перейти к шагу 3.

7. Запустить локальный спуск из точек множества

$$\{X_1, X_2, ..., X_N\} \cup I^*.$$

8. Среди полученных точек локальных минимумов выбрать наилучшую и выдать ее в качестве глобального минимума.

Алгоритм ЭГАЛС представляет собой целый класс алгоритмов, которые могут различаться не только параметрами M и N, но и используемыми базовыми алгоритмами, так как генетические алгоритмы и эволюционные

стратегии имеют различные схемы реализации и могут быть весьма разнообразными, а алгоритмы локального спуска зависят от выбранной системы окрестностей, способа просмотра окрестности и тактики перехода из окрестности в окрестность. Ответ на вопрос, какая именно гибридная схема является наилучшей для конкретной задачи, не может быть дан априори и должен исследоваться отдельно в ходе численного эксперимента. Единственным замечанием может быть указание на необходимость использовать слабую систему окрестностей при спуске из точек, порожденных генетическим или эволюционным алгоритмом (а также и алгоритмом имитации отжига), так как главной целью использования сильной системы окрестностей было ускорение сходимости к локальному минимуму, а эта сходимость будет обеспечиваться базовыми алгоритмами. Поэтому при использовании сильной системы окрестностей алгоритм будет иметь все недостатки (большое количество вычислений и отсутствие существенного приближения к цели) без возможности проявить достоинства (быстрое приближение к минимуму на начальных этапах).

В качестве перспективных направлений исследований адаптивных алгоритмов поиска отметим такие возможности, как создание гибридных схем, сочетающих имитацию отжига с генетическим алгоритмом или эволюционными стратегиями, и эволюционно-генетического алгоритма, сочетающего как работу с объектными переменными (фенотипом), так и с бинарным их кодом (генотипом). В первом случае идеи имитации отжига встраиваются в операцию селекции, когда индивид, имеющий худшую функцию пригодности, будет с некоторой вероятностью включаться в следующее поколение на тех же правах, что и самые лучшие индивиды. Уже первые и самые простые эксперименты показывают, что такой гибридный алгоритм приобретает некоторые дополнительные свойства, в том числе и ускорение глобальной сходимости. Второй подход представляет собой моделирование эволюции и может быть схематически описан следующим образом:

1. Генерируется популяция на уровне фенотипа и фиксируется генотип каждого индивида.

2. Выполняется несколько шагов мутации индивидов (локальная адаптация к окружающей среде на уровне фенотипа) и однократная мутация генотипов каждого индивида.

3. В соответствии с правилами эволюционного алгоритма отбираются индивиды для порождения потомков и из них составляются "семьи".

4. Происходит скрещивание (рекомбинация) генотипов отобранных индивидов одной семьи и порождаются потомки в соответствии с правилами генетического алгоритма.

5. По генотипам потомков восстанавливаются их фенотипы и выполняется отбор по правилам эволюционного алгоритма. Идти к 2.

Таким образом, индивиды с лучшим фенотипом будут иметь больше шансов передать по наследству свой генотип, с которым будут закрепляться необходимые свойства. Очевидно, что такой подход более соответствует тому, что происходит в природе при половом размножении, чем эволюционные и генетические алгоритмы по отдельности. Кроме того, на втором шаге вместо мутации могут быть введены шаги локального спуска, что даст возможность моделировать целенаправленное приспособление к среде типа тренировки, обучения или воспитания. Возможно на пути построения именно таких изощренных гибридных схем могут быть получены мощные алгоритмы, позволяющие решать высокосложные задачи оптимизации, возникающие при синтезе сложных систем.

Как для целочисленных, так и для булевых переменных алгоритмы глобальной оптимизации могут быть использованы для идентификации класса оптимизируемой функции (является ли она структурно монотонной или нет), однако в случае немонотонных функций необходима более универсальная оптимизационная процедура. Еще одной причиной для проведения дополнительных исследований является тот факт, что даже унимодальная и монотонная функция на дискретных структурах может оказаться и не унимодальной и не монотонной в случае введения ограничений на объектные переменные. При этом даже использование штрафных или барьерных функций не значительно упростит задачу (штрафная функция, возможно, и исключит полимодальность, но не вернет монотонность).

При наличии хорошо отлаженных локальных методов естественным решением в этом случае является мультистарт, т.е. многократное применение обобщенных методов локального спуска, приведенных в четвертой главе, с различными начальными точками, сгенерированными случайным образом. Альтернативой данному подходу обычно являются алгоритмы случайного поиска, в частности для практических задач, описанных в первой и второй главах многократно применялись так называемые эволюционные алгоритмы, моделирующие процессы естественной эволюции для получения решения оптимизационной задачи.

Стандартными проблемами обычного мультистарта локального поиска являются чрезмерные и нерациональные вычислительные затраты - многократные спуски в уже известные локальные минимумы, которые, к тому же, могут быть намного хуже уже найденного.

Эволюционные алгоритмы требуют довольно точной настройки большого количества параметров, которая должна удовлетворить для каждой оптимизируемой функции весьма противоречивые требования - обеспечить широкий поиск по всему пространству оптимизации для установления перспективных регионов, в которых может находиться глобальный оптимум,

и, в то же время, гарантировать локальную сходимость в найденных перспективных регионах. Один набор параметров этого обеспечить не может, поэтому применяется переключение стратегии, что приводит к удвоению количества настраиваемых параметров и, следовательно, значительному усложнению задачи выбора алгоритма. В этой связи эволюционные алгоритмы зачастую комбинируют с локальным поиском, запускаемым из лучшей найденной точки, или улучшающим каждого индивида в популяции в ходе эволюционирования. Однако это лишь усложняет процедуру выбора правильного алгоритма и настройку его параметров.

Описанный подход, несмотря на его эффективность, на самом деле излишне ориентирован на апологетику эволюционных алгоритмов. Задача доказательства того, что эволюционные алгоритмы являются эффективными для решения сложных задач оптимизации, решена успешно. Однако укажем на то, что среди лучших алгоритмов, отобранных для решения практической задачи, большинство обычно содержат в себе в том или ином виде алгоритмы локального поиска. Закономерен вопрос, что именно определяет эффективность алгоритмов - стратегии эволюционного поиска или локальные спуски? Должен ли эволюционный алгоритм быть основной оптимизационной процедурой, а локальный поиск - лишь вспомогательным процессом или следует организовать все наоборот?

Предложим противоположную процедуру глобальной оптимизации - генерировать начальные точки мультистарта локального поиска эволюционным алгоритмом. Преимущество перед обычным мультистартом очевидно - можно получить значительно меньшее количество начальных точек, которые к тому же не равномерно распределены в пространстве поиска, а сгруппированы в зонах притяжения выявленных локальных минимумов (эволюционный алгоритм сам является оптимизационной процедурой, а значит - будет обеспечивать сходимость популяции к лучшим найденным индивидам). Данный подход позволяет также избежать значительных сложностей с настройкой параметров эволюционных алгоритмов, т.к. нет обычной для них необходимости в удовлетворении противоречивых требований - глобального исследования пространства и, одновременно, хорошей локальной сходимости. От эволюционного алгоритма требуется только обеспечить на первых шагах широкое исследование поискового пространства без сходимости к локальным оптимумам. В этом случае выбор параметров эволюционного алгоритма очевиден - параметры должны выбираться так, чтобы обеспечить наибольшее разнообразие генерируемых хромосом - высокая мутация, равномерное скрещивание, турнирная селекция, бинарное представление решений, запрет на выживание индивидов более, чем в одном, поколении, запрет клонирования.

Дополнительным механизмом обеспечения широкого поиска в пространстве оптимизации, предлагаемым в данной работе, является принудительное удаление индивидов друг от друга путем поощрения "одиноких" решений и наказание "групп". Так как довольно быстро решения начинают группироваться вокруг найденных хороших решений (прототипов локальных минимумов), получающих высокую пригодность, то эволюционный алгоритм перестает исследовать удаленные регионы, представленные отдельными индивидами с низкой пригодностью. В этом случае предлагается ввести понятие кластера, т.е. группы решений, находящихся на небольшом расстоянии друг от друга, и снижать пригодность решений из кластера путем деления их пригодности на количество индивидов, попавших в группу. Тем самым придается высокое значение отдельным индивидам из необследованных регионов и обеспечивается принудительное удаление решений друг от друга. Получаемый кластерный эволюционный алгоритм требует настройки только одного параметра - радиуса кластера, определяющего на каком расстоянии должны находиться решения друг от друга, чтобы считаться группой. Кроме того, совершенно очевидно, что после остановки кластерного эволюционного алгоритма должен выполняться мультистарт обобщенного локального поиска, т.к. именно в такой ситуации важно не ошибиться с выбором. Дело в том, что даже если полимодальная целевая функция не является, вообще говоря, монотонной, то вблизи локального минимума это свойство может, все-таки, иметь место. Значит, при удачном стечении обстоятельств будет автоматически задействован неулучшаемый алгоритм, а в остальных случаях - универсальный. Такую возможность обеспечивает только обобщенный локальный поиск, разработанный в четвертой главе.

Алгоритм КЭАОЛС

1. Выбрать размер популяции N.
2. Выбрать количество поколений M.
3. Установить радиус кластера S.
4. Установить параметры эволюционного поиска, обеспечивающие широкое исследование пространства оптимизации.
5. Инициализировать начальную популяцию из N индивидов.
6. Выполнить M шагов кластерного эволюционного поиска.
7. Взять в качестве стартовых точек N индивидов последней популяции и выполнить оптимизацию алгоритмом обобщенного локального поиска.
8. Вывести все найденные локальные минимумы.
9. Остановиться.

Полученная гибридная процедура, использующая кластерный эволюционный алгоритм на начальном этапе для генерирования стартовых точек и алгоритм обобщенного локального поиска на заключительном этапе для точного определения локальных минимумов, требует значительно

меньших вычислительных затрат, чем простой мультистарт локального поиска, и значительно меньше зависит от сложной процедуры настройки параметров, чем эволюционный поиск.

Дополнительным выигрышем является более легкое преодоление множеств постоянства. Дело в том, что существенное значение для оптимизации имеют только связные множества постоянства, т.е. такие, в которых точки с одинаковыми значениями являются соседними, но именно такие точки и будут составлять кластер, т.е. эволюционный алгоритм будет избегать их и пытаться получить точки, не лежащие рядом, а значит с высокой вероятностью не входящие в одно множество постоянства. Это и дает значительные шансы применяемому позднее обобщенному локальному спуску стартовать из точек, лежащих вне множеств постоянства или, как минимум - в разных множествах.

Таким образом, предложен алгоритм глобальной оптимизации алгоритмически заданных функций булевых, целочисленных и смешанных переменных, автоматически настраивающийся на свойства решаемой задачи.

Метод обобщенного адаптивного поиска для решения сложных задач оптимизации

Методы поиска табу, имитации отжига, эволюционные и генетические алгоритмы принадлежат одной и той же поисковой парадигме, объединяются общим названием адаптивные поисковые методы, и все могут быть рассмотрены как варианты поиска по окрестности. Предположим, что мы имеем некоторое пространство **U** потенциальных решений некоторой задачи. Мы ищем решения, которые отвечают определенным ограничениям, или, по-другому, лежат в некотором подмножестве **S** пространства **U**. Предполагая, что каждой задаче может быть приписано некоторое значение из определенного вполне упорядоченного множества **W**, оптимизационная задача будет состоять в отыскании одного или нескольких элементов множества **S** с наибольшим значением. Таким образом, задав множество $\mathbf{S} \subset \mathbf{U}$, некоторое вполне упорядоченное множество **W** (обычно - множество вещественных чисел) и некоторую функцию f: $\mathbf{U} \rightarrow \mathbf{W}$, мы можем записать задачу оптимизации в виде $f(p) \rightarrow \max_{p \in \mathbf{S} \subset \mathbf{U}}$.

Основная идея таких методов состоит в том, чтобы начинать с некоторого набора **P** потенциальных решений задачи. Этот набор может содержать несколько копий некоторого решения и поэтому называется *мультимножеством*. Если решения в **P** не являются удовлетворительными, то выбирается некоторый поднабор **Q**, который используется для образования нового набора решений **R**. Как бы ни было образовано новое решение - инициализировано ли с помощью оператора *initialise*, или создано заново с помощью оператора *create* - к нему применяется функция *value* для того, чтобы определить значение этого решения. Это значение становится некоторой областью в записи структуры, описывающей решение. Имея в наличии набор **R**, вновь созданный из поднабора **Q**, необходимо затем построить новый набор **P** в зависимости от исходного набора решений, выбранного поднабора и вновь сгенерированного набора. Этот процесс повторяется до тех пор пока набор **P** не будет сочтен удовлетворительным. Более формально данная парадигма может быть описана следующим образом.

```
Обобщенный поиск.
{Цель - максимизировать value(p), где p ∈ U}
P, Q, R : набор решений из U;
initialise(P);
while not finish(P) do
        begin
        Q := select(P);
        R := create(Q);
        P := merge(P, Q, R)
        end
```

end {Обобщенный поиск}

Здесь *select, create* и *merge* - операторы выбора, создания и слияния наборов решений, соответственно.

Соседним решением (соседом) решения $u \in \mathbf{U}$ называется любое решение, лежащее в *окрестности* $N(u)$, где $N(u) : \mathbf{U} \to \mathbf{2}^{U}$ определяет множество "близких" к *u* решений. Вообще говоря, эти решения отличаются от *u* некоторым малым изменением. Каждый поисковый алгоритм, в котором оператор *create* генерирует некоторый поднабор, включающий в себя соседей множества **Q**, может быть отнесен к так называемым алгоритмам *локального поиска* или к алгоритмам *поиска по окрестности*. В простых алгоритмах локального поиска множественные копии не используются и оператор *initialise* выбирает некоторое начальное множество решений $\mathbf{P}_0 \subset \mathbf{U}$, которое и полагается начальным набором **P**. Зачастую множество **P** состоит из единственной точки, хотя известны и более общие подходы. В цикле **while** оператор *select* часто является отображением решения на само себя, но в некоторых случаях **Q** выбирается в соответствии с некоторым критерием, например так, чтобы включить некоторое ограниченное число решений с наибольшими значениями. Затем оператор *create* генерирует подмножество **R** множества $N(\mathbf{Q})=\cup N(q)$ ❁ $q \in Q$❝. Оператор *merge* выбирает подмножество решений в **P**⇳**R**, предпочитая те из них, которые имеют большее значение. В простейшем случае оператор *merge* выбирает подмножество решений в $\mathbf{P} \cup \mathbf{R}$, обладающих максимальным значением *value*, при некоторых ограничениях на размерность. Оператор *finish* может обеспечить остановку процесса оптимизации либо после заданного количества итераций, либо когда значение наилучшего решения в **P** перестало существенно улучшаться. В случае, когда $|\mathbf{P}| = 1$, простейший алгоритм поиска по окрестности переходит из единственного решения со значением $v \in \mathbf{W}$ в соседнее решение со значением $w \in \mathbf{W}$ только когда $w > v$. Алгоритм останавливается вообще, когда не будет найдено соседнее решение с большим значением, чем уже имеется. Алгоритм всегда переходит из одного решения к другому с большим значением.

Поиск табу и имитация отжига являются двумя вариантами поиска по окрестности, специально спроектированными для того, чтобы обойти проблему захвата поиска локальным оптимумом.

В алгоритме *поиска табу* оператор *create* выбирает соседние решения элементов из множества **Q**, но пропускает те из них, которые обозначены как "табу". Список табу является динамическим множеством определяемых пользователем правил, которые устанавливают какие из соседних решений являются табу, т.е. запрещены. В большинстве случаев также включают *уровни аспирации* (*aspiration levels*), которые являются правилами, указывающими, что определенные перемещения предпочитаются другим, и

разрешающими даже нарушать табу. Это позволяет исключить появление чрезмерно ограничивающих правил табу. Поиск табу успешно применялся ко многим прикладным задачам. Это указывает на важность наличия некоторого рода экспертной системы, направляющей поиск.

Идеей, породившей подход *имитации отжига* в оптимизации, является термодинамический закон, который устанавливает, что при температуре t вероятность возрастания величины энергии E задается соотношением, где k есть физическая константа, известная под названием *постоянная Больцмана*. Если поставить в соответствие состояния системы решению некоторой задачи оптимизации, то стоимость решения будет соответствовать понятию энергии, а переход к любому соседнему решению - изменению состояния. Первоначально нет строгого предпочтения для перемещения в стороны решений с большим значением, но, так как температурный параметр уменьшается, то это предпочтение становится все сильнее. Тем не менее, всегда будет некоторый шанс, что решение с худшим значением будет предпочитаться более хорошему решению. Имитация отжига может также быть рассмотрена как поисковая стратегия, которая управляется некоторой экспертной системой. В этом случае правило выбора соседа расширяется таким образом, чтобы включить случайный элемент, зависящий от температуры. Сила имитации отжига в использовании этой случайности.

Генетические алгоритмы являются поисковыми методами, основанным на абстрагированной модели Дарвинской эволюции. Решения представляются строками фиксированной длины над некоторым алфавитом (алфавит “гена”) и каждая такая строка рассматривается как “хромосома”. Значение решения представляется как “*пригодность*” (*fitness*) такой хромосомы. Затем применяется принцип “выживания сильнейшего” и лучшим решениям предоставляется возможность скомбинироваться с тем, чтобы произвести потомство (по возможности более пригодное). Таким образом парадигма обобщенного поиска превращается в генетический алгоритм, если

- **P** является набором содержащим более, чем один элемент,
- оператор *create* использует генетические операторы для порождения решений,
- как оператор *select*, так и оператор *merge* зависят от эволюции решений и вместе благоприятствуют сохранению решений с более высоким значением функции *value*.

Набор **Q**, сформированный с использованием оператора *select*, представляет собой набор “пригодных” решений, которые формируют набор индивидов для дальнейшего “бракосочетания”. Оператор *create* применяет генетические операторы, а именно *скрещивание, воспроизводство* и *мутацию*, к этому набору с тем, чтобы сгенерировать потомство. И наконец, оператор *merge* используется в генетическом алгоритме для того, чтобы

скомбинировать старую популяцию **P**, популяцию решений в воспроизводящем наборе **Q** и новую популяцию **R** решений, произведенных оператором *create*.

В данном разделе предлагается унифицированный подход к различным поисковым стратегиям, которые были рассмотрены.

Генетический алгоритм может быть рассмотрен как иной тип поиска по окрестности. Вместо того, чтобы рассматривать окрестность как функцию $\mathbf{U} \to 2^{\mathbf{U}}$, можно расширить это понятие и определить N_2: $\mathbf{U}\times\mathbf{U} \to 2^{\mathbf{U}}$. Оператор скрещивания в этом случае определяет некоторую окрестность двух решений в **U**, взятых вместе.

Если теперь вернуться к парадигме обобщенного поиска, то задавая множество решений $\mathbf{Q} \subset \mathbf{P}$ можно определить оператор *create* таким образом, чтобы он генерировал некоторое подмножество $\mathbf{N}_1 \cup \mathbf{N}_2$, где

$$\mathbf{N}_1 = \cup \{N_1(u): u \in \mathbf{Q}\},\ \mathbf{N}_2 = \cup \{N_2(u, v) : u, v \in \mathbf{Q}\}.$$

$\mathbf{N}_1$ соответствует мутантам, а $\mathbf{N}_2$ - потомкам. Функция N_2 может быть не только чистым скрещиванием, но и определяться более сложным образом.

Рассматриваемая подобным образом парадигма генетического алгоритма является только простым расширением поиска по окрестностям. В настоящий момент большинство генетических алгоритмов только выбирают случайным образом одно решение из $N_2(u,v)$ для любых $u, v \in \mathbf{Q}$. Существуют и другие возможно более интересные варианты данного подхода, при которых более вероятно сгенерировать более пригодных потомков. Это могло бы быть достигнуто путем либо генерирования большего числа детей в одном цикле размножения и отбраковывания более слабых потомков, либо ограничения точек скрещивания в соответствии с некоторыми (возможно изменяющимися динамически) правилами.

В имитации отжига главной идеей является разрешение слабым мутантам выживать, пока температура достаточно высока, но генетические алгоритмы также имеют возможность обеспечения выживания слабых членов множества решений, при этом у них всегда есть механизмы предпочтения более пригодных решений. Парадигма генетических алгоритмов может быть усилена путем включения параметра температуры или хотя бы некоторого динамического управления существующими параметрами.

Парадигма генетических алгоритмов может быть также существенным образом улучшена путем усвоения некоторых уроков поиска табу. Посредством манипулирования динамическим списком правил как $\mathbf{N}_1$, так и $\mathbf{N}_2$ могут быть сокращены или проранжированы по степени желательности. Хотя довольно просто использовать значение *value* в качестве меры пригодности, мы можем также использовать такого рода ранжирование как дополнительную или альтернативную меру пригодности. Использование

такого экспертного руководства процессом не следует ограничивать только оператором *create*. Необходимо иметь определения для каждого из операторов *initialise, finish, select, create, merge,* $\mathbf{N}_1$, и $\mathbf{N}_2$. Можно предусмотреть алгоритмы, в которых эти операторы будут весьма изощренными. (Например, моделирование эволюции по Ламарку, которая предусматривает прижизненную адаптацию индивидов, можно осуществить, применяя локальный спуск из каждой точки популяции.) Поиск табу предполагает включение экспертной системы, определяющей некоторые из этих сложных видов операторов. Более того, если такого рода экспертная система будет задана как набор правил, то они могут обновляться в процессе работы программы (например включением или исключением уровней аспирации).

Существенным является и то, что обобщенный поиск в практически всех его вариантах обладает большим потенциалом для распараллеливания. Оценивание решений, генерирование, оценка и отбор соседей, а также их комбинирование может выполнятся параллельно.

Обобщенный метод адаптивного поиска будет использовать компоненты парадигм поиска табу, имитации отжига и генетических алгоритмов. Задача кодируется как строка конечной длины, содержащая буквы конечного алфавита. Обозначим через **T** множество всех возможных строк.

Определим переменную *valued_string*, представляющую некоторое решение задачи, как пару (v, s), где $v \in \mathbf{W}$, $s \in \mathbf{T}$ и $v = value(s)$. **U** обозначает множество всех возможных *valued_string*.

1-окрестность некоторой *valued_string* определяется как отображение $N_1:\mathbf{U}\rightarrow 2^U$ и может быть построена, например, с использованием генетических операторов воспроизводства или мутации или обычным для локального поиска способом.

2-окрестность некоторой пары, состоящей из *valued_string*, определяется как отображение $N_2 : \mathbf{U}\times\mathbf{U} \rightarrow 2^U$. Эта окрестность может быть, например, построена с помощью генетического оператора скрещивания или каким-либо другим образом.

В общем случае k-окрестность некоторого набора, состоящего из k строк типа *valued_string*, определяется как $N_k : \mathbf{U}^k \rightarrow 2^U$.

Определим теперь общее понятие окрестности $N : 2^U \rightarrow 2^U$ как

$$N(\mathbf{S}) \subseteq \{N_1(s) : s \in \mathbf{S}\} \cup \{N_2(s_1, s_2): s_1, s_2 \in \mathbf{S}\} \cup \ldots$$

Пусть *CL* - список порождения, *SL* - список отбора, *ML* - список слияния. **P**, **Q**, **R** $\subset$ **U** - наборы строк *valued_string*.

Определим пять операторов следующим образом:

1. Оператор *initialise*.

а. Создается некоторая начальная популяция $\mathbf{P} \subset \mathbf{U}$, состоящая из n строк типа *valued_string*, где $n \geq 1$.

б. Создается список отбора *SL*. Это - множество некоторых правил, которые используются для настройки отбора строк в качестве соседей. Список является динамическим - на каждом шаге итерации содержание списка может быть модифицировано. Каждое правило имеет вид $\mathbf{U} \rightarrow \mathbf{W}$. Множество таких правил затем определяет функцию *select_score*: $\mathbf{U} \rightarrow \mathbf{W}$, например добавлением значений возвращаемых правил. Эта функция будет в общем случае отражать как значение решения, так и меру того, насколько желательным является поиск в его направлении.

в. Создаются списки порождения и слияния. Они являются динамическими множествами правил, которые тем же самым образом как и список отбора дают классификацию для каждой строки *valued_string*. Однако соответствующие им функции *create_score* и *merge_score* могут использовать альтернативные механизмы для вычисления своих значений.

2. Оператор *select_SL*(**P**). Подмножество $\mathbf{Q} \subset \mathbf{P}$ порождается в соответствии с некоторым механизмом селекции, который предпочитает решения с более высокими значениями *select_score*. Например, строки *valued_string* могут выбираться с повторениями из **P** и добавляться в **Q** с некоторой вероятностью, равной их относительным значениям *select_value*.

3. Оператор *create_CL*(**Q**). Создается подмножество $\mathbf{R} \subset \mathrm{N}(\mathbf{Q})$ с использованием некоторых механизмов, которые предпочитают решения с более высокими значениями функции *create_score*.

4. Оператор *merge_ML*(**P**, **Q**, **R**). Некоторое количество (от 0 до n) элементов множества **P** заменяется элементами множества **R** в соответствии с некоторым комбинирующим механизмом, который предпочитает решения с более высокими значениями функции *merge_score*. Это может зависеть и от множества **Q**, использованного при построении множества **R**.

5. Оператор *update*(*SL, CL, ML*). Правила добавляются, модифицируются или отбрасываются из списков порождения, отбора или замещения. Этот процесс сам может управляться экспертной системой или может зависеть от количества итераций или от того насколько близко мы подошли к локальному оптимуму.

Теперь представим в формализованном виде обобщенный алгоритм адаптивного поиска.

Обобщенный адаптивный поиск
initialise;
while not (*finish*(**P**)) **do**
begin
    Q := *select_SL*(**P**);
    R := *create_CL*(**Q**);
    P := *merge*(**P**, **Q**, **R**);

update(*SL, CL, ML*)

end

Таким образом представлен обобщенный метод адаптивного поиска, который включает в себя все основные идеи существующих подходов к построению адаптивного поиска, а также все возможные схемы обычного локального поиска. Более того, выполнено обобщение всех этих методов и указана единая методика, которая предоставляет возможности для построения новых алгоритмов. Кроме того, предложенная методика может эмулировать поиск табу, имитацию отжига, генетические алгоритмы и локальный спуск при надлежащем выборе и настройке структуры, т.е. выбирать стратегию решения сложных задач оптимизации. Необходимо также предусмотреть процедуру настройки параметров используемых алгоритмов при выбранной структуре. Конечно же можно и здесь использовать аналог экспертной системы, но мы еще не исчерпали возможности самоадаптации алгоритмов, которые могут быть расширены на пути дальнейшего обобщения понятия стратегии поиска.

Самоадаптация алгоритмов в процессе оптимизации. Можно указать три уровня самоадаптации поиска - уровень индивида, уровень субпопуляции и уровень популяции.

Адаптация на уровне популяции является структурной адаптацией алгоритма и означает, что могут существовать несколько различных стратегий поиска, т.е. фактически несколько популяций, каждая из которых применяет свою стратегию выживания, конкурируя за общий ресурс, причем лучшая популяция получает больше этого ресурса.

Внутри каждой популяции могут существовать субпопуляции, придерживающиеся одной и той же стратегии, но выбирающие по-разному параметры генетических операторов. Эти субпопуляции могут сотрудничать (моделирование половой рекомбинации) или конкурировать (видовая конкуренция). Например, одна субпопуляция выбирает высокую степень мутации и двуточечное скрещивание, а другая - низкую мутацию и одноточечное скрещивание. Возможно также варьирование типа селекции.

Адаптация на уровне индивида предусматривает изменение в процессе поиска таких его параметров как уровень мутации, вероятность скрещивания, точка скрещивания и т.п., т.е. способов порождения соседних точек, характерных именно для этого индивида.

С точки зрения оптимизации эти три вида адаптации можно определить как структурную адаптацию и настройку параметров. Уровень адаптации определяет также интервал времени, в течение которого эта адаптация выполняется - самый короткий для индивидов, более длинный для субпопуляций и самый длинный - для популяций. Чем длиннее интервал адаптации, тем меньше затраты на нее и тем ниже ее эффективность.

Индивиды конкурируют в процессе селекции и для их отбора существует понятие пригодности, определяющей вероятность, с которой индивид будет представлен при воспроизводстве потомства. Генетические операторы тоже могут иметь свою функцию пригодности (например - процент улучшенных индивидов, полученных с помощью данного оператора), которая определяет вероятность применения данного оператора. Для субпопуляций и популяций необходимо ввести свою функцию пригодности. С помощью этой функции можно будет определять лучшую популяцию и предоставить ей больше возможностей для воспроизводства. Пусть T - интервал адаптации, $b_i(t)$ равно единице, если i-я популяция в момент t содержит наилучшего индивида, $k = 0$ означает текущую ситуацию, $k = 1$ - предыдущую и т.д. Тогда качество популяции можно вычислить следующим образом:

$$q_i = \sum_{k=0}^{T-1} \frac{T-k}{k} \cdot b_i(t).$$

Изменение размеров группы можно выполнять, например, сокращением каждой проигравшей популяции (не достигшей минимального гарантированного размера) на 10-12 % и увеличением победившей популяции на число, равное сумме потерь проигравших. Общее число индивидов остается неизменным, а адаптация осуществляется достаточно быстро. Необходимо также предусмотреть *миграцию* лучшего индивида в другие популяции через определенное число поколений.

Описанный подход позволяет автоматически выбирать лучшую стратегию из имеющихся и включать ее в необходимый момент. Однако было бы полезно предусмотреть возможность более тонкой настройки стратегий, чем просто переключение. Такая возможность легко реализуется в рамках обобщенного адаптивного поиска, т.к. понятия хромосомы и “генетических” операторов стало настолько общим, что позволяет описывать даже стратегии оптимизации и их преобразования - “мутацию” и “скрещивание”. Достаточно только сформировать хромосому из элементарных концепций, входящих в понятие стратегии (стратегические гены). Например, включение различных типов локального поиска для “усовершенствования” индивидов, подключение структур поиска табу, включение “температуры” (как в имитации отжига) для преодоления локальной сходимости и многие другие. Скрещивание будет выполняться как обычно, формируя тем самым потомков, которые обладают комбинированными стратегиями из уже известных.

Синергетический эффект подхода заключается в том, что в такую систему можно встроить базу знаний и правил, формируемую человеком при построении задачи оптимизации. Так, например, проектировщик, ставящий задачу системе, конечно знает содержит ли задача такие особенности как многокритериальность, разношкальность, многоуровневость. Значит он может ввести в базу знаний по формированию стратегии оптимизации

правила, предписывающие при обучении стратегии всегда включать структуры, обрабатывающие упомянутые особенности. Достаточно указать, что сила срабатывания соответствующих правил - 100%, и система всегда будет настраиваться на решение именно такой задачи, которая и решается. В эту же систему вносятся и знания эксперта-оптимизатора с правилами различной силы срабатывания. Т.к. стратегии оптимизации включают и обычные поисковые алгоритмы и даже алгоритмы матпрограмминования, то выбор метода решения задачи будет осуществляться автоматически, с последующей заменой на более подходящий, если ситуация изменится. Пригодность стратегий определяется по обычным правилам - достигнутый результат, скорость улучшения, наличие в популяции наилучшего индивида, прогноз экспертной системы по полезности и т.п.

Таким образом, предложена стратегия обобщенного адаптивного поискового алгоритма, которая включает в себя все известные поисковые алгоритмы как частные случаи. Рекомендована многопопуляционная схема эволюционного поиска, которая позволяет автоматически выполнять структурную и параметрическую адаптацию алгоритма в процессе оптимизации. Многократное решение сложных практических задач показало, что предлагаемый подход позволяет эффективно комбинировать поисковые алгоритмы таким образом, что получаемая комбинация оказывается лучше, чем каждый из алгоритмов в отдельности. Следовательно построена эффективная самонастраивающаяся процедура решения сложных практически значимых задач оптимизации, которая может быть успешно применена во многих задачах оптимизации работы и оптимального проектирования сложных технических систем.

Ниже приведены два графика, иллюстрирующие работу описанной в данной статье схемы при решении задачи оптимального синтеза структуры технологического контура системы управления орбитальной группировкой космических аппаратов. Указанная задача содержит 112 разнотипных переменных. Целевая функция задана алгоритмически - для вычисления значения целевой функции необходимо решить 18 систем линейных уравнений с 40 переменными каждая. На рис. 1 иллюстрируется конкуренция популяций, адаптирующая структуру алгоритма. На рис. 2 - конкуренция субпопуляций обычного генетического алгоритма (№ 3 на рис. 1) в то время, когда он преобладал над другими стратегиями.

На предварительном этапе были выбраны пять различных стратегий адаптивного поиска, которые были оценены как “перспективные” для дальнейшей оптимизации. Процедура выбора состояла в том, что были эмулированы 20 различных схем адаптивного поиска и запущены 20 популяций по 25 индивидов в каждой. Конкуренция на данном этапе позволяла более сильным популяциям “уничтожать” более слабые так, что после 100 поколений остались только 5 победителей, а остальные полностью

исчезли не выдержав конкуренции. Самой сильной популяции были предоставлены 300 индивидов (60%), а остальным - гарантированный минимум (50 индивидов или 10%). После этого начался основной этап оптимизации, состоящий из 70 тысяч поколений соревнующихся популяций.

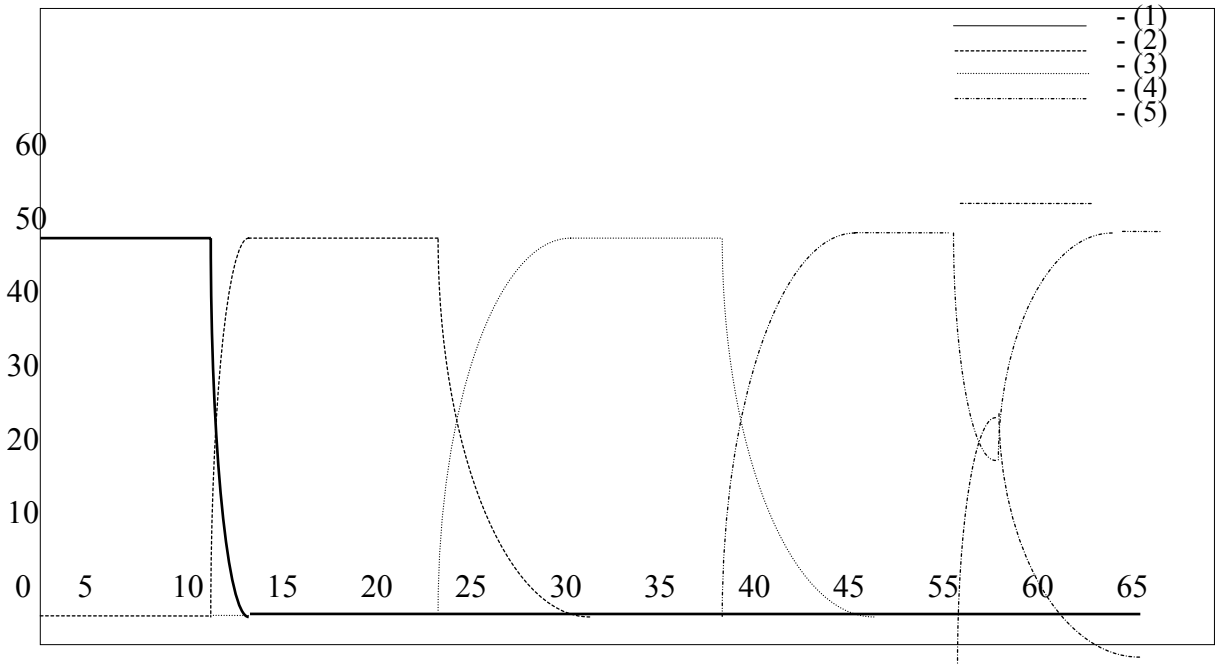


Рис. 1. Изменение размеров конкурирующих популяций в процессе оптимизации - адаптация структуры алгоритма (1 - ГА+список табу+ критерий аспирации+локальный спуск; 2 - ЭА с обобщенной хромосомой+список табу+критерий аспирации+локальный поиск; 3 - ГА; 4 - ЭА с обобщенной хромосомой; 5 - локальный поиск+список табу). Ось абцисс - количество поколений (тыс.), ось ординат - размер популяций в %).

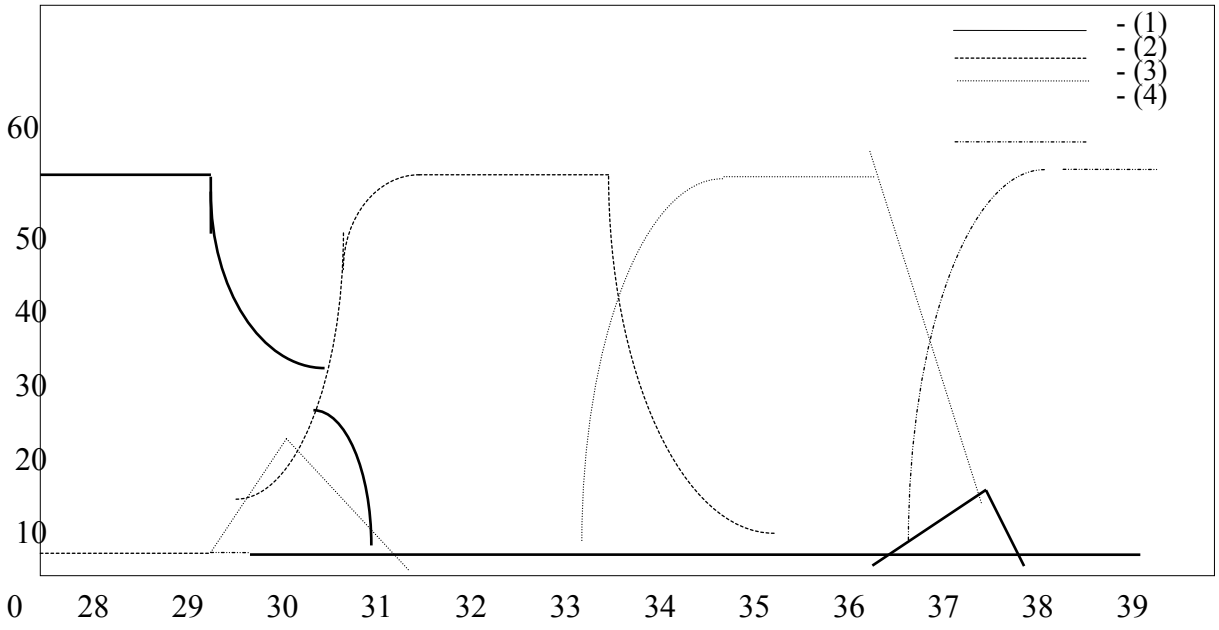


Рис. 2. Изменение размеров конкурирующих популяций в процессе оптимизации обычным ГА - адаптация параметров (1 - высокая мутация, турнирная селекция без элитарности; 2 - средняя мутация, пропорциональная селекция, низкая элитарность; 3 - средняя мутация, пропорциональная селекция, высокая элитарность; 4 - боксовая селекция, низкая мутация, высокая элитарность). Ось абцисс - количество поколений (тыс.), ось ординат - размер популяций в %).

На первой стадии доминировала популяция стратегия которой включала генетический алгоритм (т.е. бинаризацию разнотипных переменных) с экспертной системой (правила из списка табу предотвращают сильное нарушение ограничений и возврат к известному локальному минимуму, а критерий аспирации позволяет нарушать ограничения индивидам с очень хорошим значением целевой функции). Локальный спуск выполнялся из каждой точки популяции, моделируя прижизненную адаптацию индивидов. Затем доминирующей стала вторая популяция, отличающаяся тем, что эволюционный алгоритм работал с обобщенной хромосомой (т.е. поисковые точки берутся в той форме, в которой они изначально заданы, бинаризация не выполняется, а для каждого вида переменных строятся свои генетические операторы). Третьим лидером стал обычный генетический алгоритм (с бинаризацией хромосом и отбрасыванием недопустимых точек), который затем сменился эволюционным алгоритмом с обобщенной хромосомой (также без списка табу и локального спуска), завершившим процесс оптимизации (критерий останова - 1000 поколений без улучшений). Пятый алгоритм (локальный спуск под управлением поиска табу) только на короткое время становился доминирующим, но не полностью.

Кроме адаптации структуры алгоритма выполнялась также адаптация параметров на уровне субпопуляций. Таким образом, что каждый алгоритм (и доминирующий и рецессивные) постоянно настраивал свои параметры. На рис. 2 термин “элитарность” означает процент популяции, который отбирался для селекции и рекомбинации (вся популяция - без элитарности, 50% - средняя элитарность, 25% - высокая элитарность). Высокая мутация - 0.1, средняя - 0.01, низкая - 0.001. Адаптация индивидуальных параметров не выполнялась. Среднее время проработки алгоритма - 11%, остальные 89% - время, затраченное на вычисление целевой функции и ограничений задачи оптимизации. Все результаты усреднены по 50 экспериментам.

Были проведены контрольные расчеты двух типов. В первом случае всем пяти выбранным алгоритмам по очереди предоставлялись те же вычислительные ресурсы для решения задачи. Наилучший результат (полученный алгоритмом № 4) был на 8,5 % хуже найденного при комбинированном подходе.

Второй вид контрольных расчетов (по 10 экспериментов) состоял в применении описанного подхода с алгоритмами, которые были отброшены на предварительной стадии. При тех же затратах полученные результаты

всегда значительно хуже, описанных выше, за исключением комбинации, состоящей из алгоритмов №№1-4 и локальных спусков вместо алгоритма №5. В таких случаях получаемое решение, хотя и уступало наилучшему, но незначительно (на 1,5-6%). Интересно отметить, что исключение из набора алгоритмов №1 или 4 всегда приводило к значительному ухудшению решения и даже преждевременной сходимости.

Аналогичные результаты применения подхода были получены для других задач и для других контуров управления. Во всех случаях наблюдалось похожее поведение алгоритмов, хотя и не всегда их было именно пять. Состав группы алгоритмов также менялся. Типичным является также то, что при переходе от одного варианта задачи к другому набор алгоритмов и порядок их срабатывания тоже менялся. Это объясняет, почему при решении данных задач отдельными алгоритмами системы EvoOpt не удавалось обнаружить лучший алгоритм. Видимо структура задачи такова, что небольшое изменение параметров приводит к существенному с точки зрения поисковых алгоритмов изменению поверхности отклика.

В результате решения реальных задач и анализа результатов можно с уверенностью утверждать, что предложенный обобщенный алгоритм адаптивного поиска обладает высоким потенциалом в решении сложных оптимизационных задач.

ПРИМЕНЕНИЕ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ В ЗАДАЧАХ АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Гибридный алгоритм генетического программирования

При решении задачи символьной регрессии с помощью метода ГП часто возникает следующая проблема: в задачах, где решение имеет сложное выражение, деревья с более простой структурой (обычно линейные выражения) имеют пригодность выше, чем деревья со сложными структурами (которые обычно оказываются более перспективными). В результате простые решения начинают преобладать в популяции, и поиск замедляется. Это связанно с тем, что набор констант во множестве термов фиксирован, и в удачно выращенных структурах численные коэффициенты часто оказываются подобранны плохо. Таким образом, решение с очень хорошей структурой может иметь ошибку аппроксимации намного больше, чем простая структура, в которой коэффициентов меньше.

Следующий пример наглядно показывает, как плохо подобранные коэффициенты влияют на пригодность решения. Заданна функция $f(x) = 3 * \sin(x)$. Пусть в популяции есть два решения: $p_1(x) = x$ и $p_2(x) = x - x^3$. Графики этих функций показаны на рис. 1.

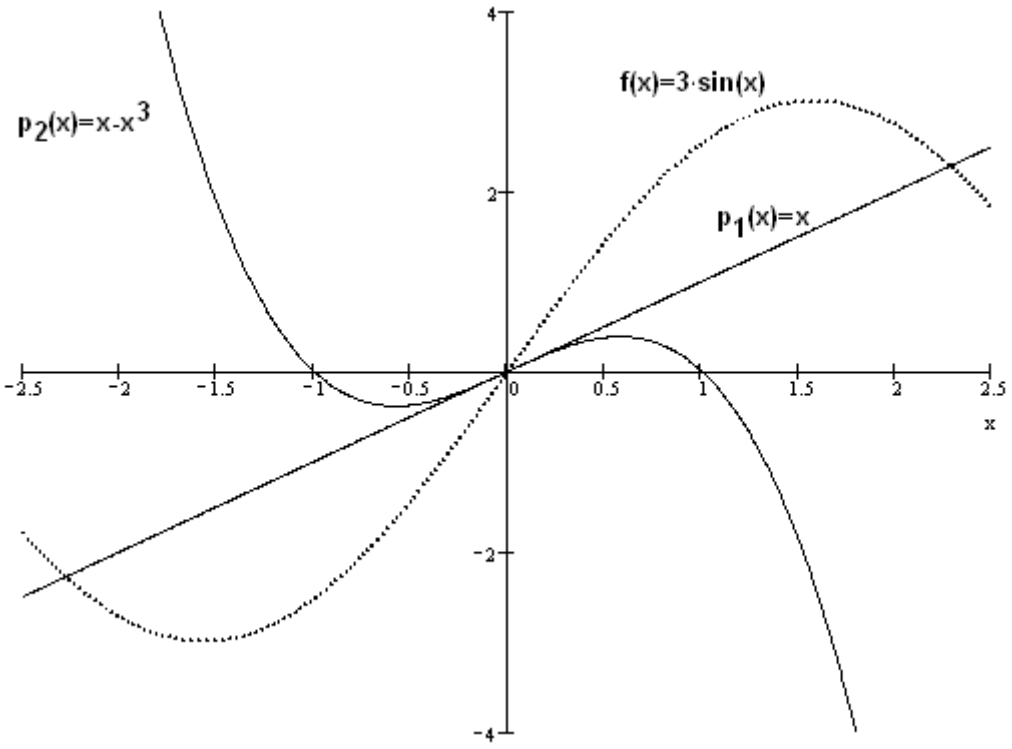


Рис. 1. Пример решений без подбора коэффициентов

Очевидно, что решение $p_1(x)$, несмотря на то, что оно плохо аппроксимирует функцию $f(x)$, окажется намного лучше, чем $p_2(x)$. Однако если в $p_2(x)$ подобрать коэффициенты, то решение $\hat{p}_2(x) = 2x - 0.2x^3$ даст меньшую ошибку аппроксимации, и теперь алгоритм отдаст предпочтение этому решению (рис. 2).

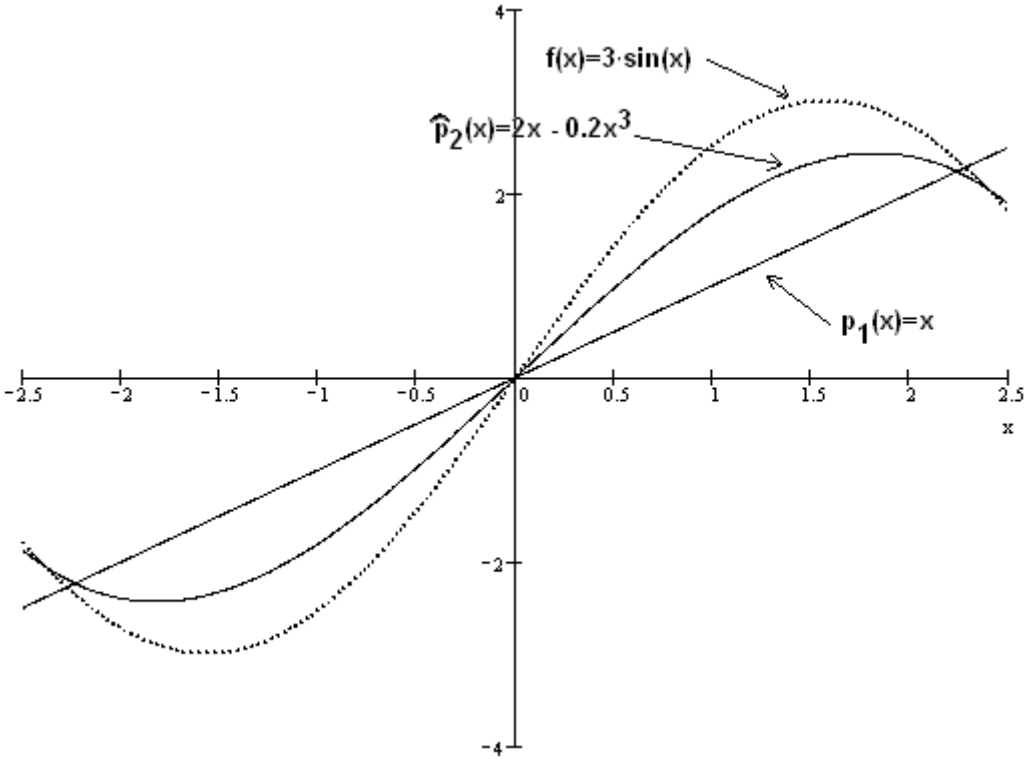


Рис. 2. Пример решений с подбором коэффициентов

Обычно метод ГП способен преодолеть данную проблему путем интенсивного применения оператора мутации, однако случайный характер оператора мутации может оказать и негативное воздействие (могут быть потеряны хорошо подобранные коэффициенты или изменена хорошо выращенная структура). Также подстройка коэффициентов возможна за счет появления поддеревьев, содержащих во внешних вершинах только константы. В результате вычисления значения такого поддерева, может появиться численный коэффициент, не включенный во множество термов. Тем не менее, зачастую алгоритм либо плохо справляется с подбором коэффициентов, либо коэффициенты настраиваются очень долго.

Для решения данной проблемы предложена следующая модификация обычного метода ГП. Из множества термов исключаются константы. Таким образом, полученные деревья решений будут состоять только из функций и переменных, следовательно, будут представлять не готовые решения, а их структуры (шаблоны, функциональные формы). При оценивании решений

численные коэффициенты структур настраиваются с помощью какой-либо внешней оптимизационной процедуры.

Очевидно, что при использовании предложенной модификации метода ГП, в случае появления в популяции удачных структур, после подбора численных коэффициентов, алгоритм отдаст предпочтение именно этим структурам. В результате, при аппроксимации сильно нелинейных зависимостей, алгоритм гораздо быстрее перейдет от линейных (или слабо нелинейных) функций к более сложным.

Во множество численных коэффициентов для заданной структуры будем включать: коэффициенты для каждой внешней вершины (переменной) и свободный член (аналогично смещению в нейронных сетях). Пример, определения множества коэффициентов показан на рис. 3 и 4.

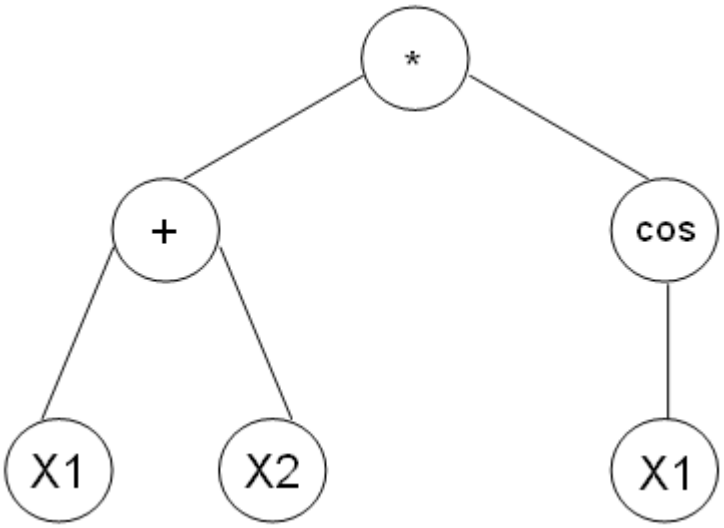


Рис. 3. Структура $F(x_1, x_2) = (x_1 + x_2) * Cos(x_1)$

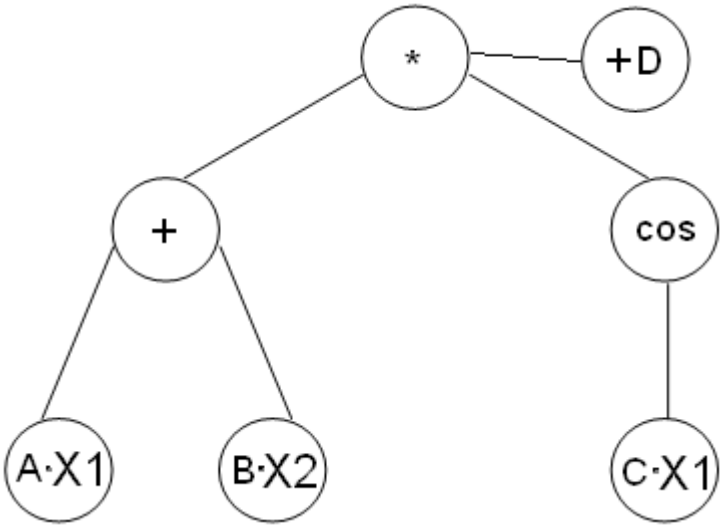


Рис. 4. Решение $F(x_1, x_2) = (a \cdot x_1 + b \cdot x_2) * Cos(c \cdot x_1) + d$

Очевидно, что количество численных коэффициентов для сложных структур будет велико. Максимальное количество внешних вершин у дерева глубины d равно 2^d. На практике глубина деревьев обычно равна 6-10.

Не трудно понять, что при настройке параметров заданной структуры, функция ошибки аппроксимации (критерий оптимизации при настройке параметров) будет нелинейна и многоэкстремальна. Как известно, ГА достаточно эффективно решают сложные задачи оптимизации, более того они используют тот же механизм, что и метод ГП. Поэтому в дальнейшем для настройки численных коэффициентов структур будем использовать именно ГА.

Для сокращения времени работы алгоритма оптимизации, будем использовать ВГА, который превосходит обычный ГА по быстродействию. Для более эффективного использования вычислительных ресурсов, будем использовать ВГА с малым размером популяции и сильной сходимостью (пропорциональная селекция, турнирная с большим размером турнира, элитарная), но высокой мутацией. Такой алгоритм несколько теряет глобальные свойства поиска, однако сходится значительно быстрее.

Общая схема гибридного алгоритма символьной регрессии, сочетающего стандартную процедуру генетического программирования для выбора структуры модели и вероятностный генетический алгоритм для настройки ее параметров, имеет следующий вид:

1. Определить функциональное множество и множество термов (переменные и априори известные константы). Задать параметры алгоритмов ГП и ВГА.
2. Инициализировать популяцию с помощью выбранного способа выращивания деревьев.
3. С помощью ВГА «грубо» настроить численные коэффициенты полученных структур. Оценить популяцию.
4. Применить операторы селекции, скрещивания и мутации. Сформировать новую популяцию.
5. Если заданная точность аппроксимации достигнута, то перейти к шагу 6. Иначе повторить шаги 3-5.
6. Произвести «точную» настройку численных коэффициентов лучшего найденного решения с помощью ВГА.

Алгоритм генетического программирования для отыскания частных решений диференциальных уравнений

Введение

Дифференциальные уравнения – такая же неотъемлемая часть математики, как математический анализ и алгебра. Дифференциальные уравнения имеют большое практическое применение. Они используются сейчас практически во всех областях науки и техники, при решении теоретических и практических задач математики, физики, химии, биологии и других областях, таких как авиация, космонавтика, в различных отраслях промышленности. Дифференциальные уравнения представляют собой математические модели естественных процессов в природе и технике. Результаты решения дифференциальных уравнений позволяют оптимизировать эти процессы по интересующим человека критериям. Решение дифференциального уравнения – задача нетривиальная, так как на сегодняшний день нет универсального способа нахождения решения произвольного уравнения.

Существует два метода решения обыкновенного дифференциального уравнения – это аналитический и численный методы. Аналитический метод заключается в нахождении функции (или множества функций) удовлетворяющей заданному ОДУ (и начальным условиям). Аналитический метод применяет для каждого типа уравнений определенный алгоритм нахождения решения. Численный метод состоит в нахождении конечного множества точек (на заданном интервале)- значений функции, каждая из которых удовлетворяет ОДУ (и начальным условиям). Численный метод находит решение, используя аппроксимацию производных конечно-разностными схемами. Каждый из этих методов имеет свои преимущества и недостатки. Преимущество аналитического метода состоит в строгом математическом доказательстве существования и единственности решения. Недостаток же заключается в том, что не каждое дифференциальное уравнение можно решить аналитически, так как не для каждого типа уравнения найден алгоритм нахождения решения. Численным методом можно решить с большей или меньшей погрешностью практически любое уравнение. Однако результаты решения численного метода не могут быть использованы для дальнейшего

аналитического исследования, и погрешность найденного решения иногда сводит на нет результат. Также при численном решении уравнения возникают трудности при определении сходимости и устойчивости методов.

В силу отсутствия универсального метода целесообразно объединить преимущества обоих методов. То есть построить в виде частного решения ОДУ найти аналитическую функцию, удовлетворяющую заданному ОДУ (и начальным условиям), аппроксимируя производные конечно-разностными схемами. И алгоритм генетического программирования позволяет это сделать.

Постановка задачи

Пусть дано дифференциальное уравнение n-го порядка в общем виде

$$F(x, y, y', y'', \ldots y^{(n)}) = 0 \qquad (1)$$

и начальные условия

$$y(x_0) = Y_0,\ y'(x_0) = Y_0',\ \ldots,\ y^{(n-1)}(x_0) = Y_0^{(n-1)} \qquad (2)$$

Здесь и далее $y = \varphi(x)$ – функция искомого решения (зависимая переменная), $y', y'', \ldots y^{(n)}$ – соответственно, первая, вторая, … , n-я производные функции $y = \varphi(x)$. А Y_0, Y_0', …, $Y_0^{(n-1)}$ – заданные вещественные постоянные.

Требуется найти функцию y(x), удовлетворяющую уравнению (1) и условиям (2).

В курсе обыкновенных дифференциальных уравнений доказана следующая теорема существования и единственности решения для уравнения n-го порядка:

Пусть дано уравнение $y^{(n)} = f(x, y, y', y'', \ldots, y^{(n-1)})$ (3) *и пусть в области B функция a и ее частные производные первого порядка по* $y, y', y'', \ldots, y^{(n-1)}$ *непрерывны, и точка* $(x_0, y_0, y_0', \ldots y_0^{(n-1)})$ *лежит внутри D. Тогда при начальных условиях* $y(x_0)=Y_0$, $y'(x_0)=Y_0'$, … , $y^{(n-1)}=Y_0^{(n-1)}$ *уравнение* (3) *имеет единственное решение.*

Итак, решение существует и оно единственно, следовательно, задача - корректна.

Характерные особенности существующих методов решения.

Оба подхода (аналитический и численный) не являются универсальными способами решения задачи. Каждый из них имеет свои

сильные и слабые стороны. Рассмотрим подробнее особенности каждого из них.

Аналитический метод основан на строгом математическом доказательстве существования и единственности решения, для каждого конкретного типа уравнения. Решением является аналитическая функция, которая пригодна для дальнейшего исследования, например, для нахождении я экстремумов, нулей функции и т.д. также нужно отметить, что если решение найдено, то оно точно для уравнения (1) и (2). Это и является главными преимуществом аналитического метода.

Главным же недостатком является, то что не каждое уравнение можно решить аналитически. Лишь для малой части дифференциальных уравнений существуют алгоритмы нахождения решения. Так, для линейных уравнений n-го порядка с постоянными коэффициентами существует относительно простой способ нахождения решения, однако уравнение с переменными коэффициентами уже не решается в общем случает и требует задания дополнительного условия(одного частного решения этого уравнения). Также нужно сказать, что если даже алгоритм нахождения решения существует, то во многих случаях возникают сложности при интегрировании ОДУ (несобственные или неберущиеся интегралы).

Рассмотрим теперь характеристики численного метода.

Численный метод основан на машинном вычислении разностных схем, которые предполагают огромное количество операций для нахождения решения. В силу того, что здесь используется аппроксимация производных – метод не зависит от вида уравнения и его сложности. Для любого уравнения можно построить схему (специфическую) и решить ее на компьютере. Это важное обстоятельство и составляет преимущество метода перед аналитическим.

Однако, у численного метода есть и значимые недостатки. Главный из них это дискретность получаемого решения, предполагающая невозможность дальнейшего исследования и применения в теоретических расчетах. Также значительные неудобства при работе с этим методом доставляет необходимость исследования разностной схемы на устойчивость. То есть чтобы метод сходился к решению уравнения часто требуется накладывать ограничения на шаг и размер сетки, что тоже ограничивает область применения данного метода. Также можно отметить, что каждое разностное

уравнение в большинстве случаев требует строительства специфической разностной схемы для решения уравнения. Универсальную схему для разных уравнений построить невозможно.

Итак, у каждого метода есть свои преимущества и недостатки. Универсального метода решения поставленной задачи нет.

Модифицированный алгоритм генетического программирования

В связи с тем что задача имеет много параметров и условий классический алгоритм генетического программирования не отвечает специфике задачи. Поэтому для решения поставленной задачи потребовалось несколько модифицировать классический алгоритм. Таким образом модифицированный алгоритм для решения задачи отыскания частного решения ОДУ можно схематично представить так

1. Генерируем начальную популяцию решений P(0) одним из методов выращивания, использую элементы множеств F и T. Каждое решение – бинарное дерево.
2. Берем каждое решение в популяции и оптимизируем вещественные коэффициенты, входящие в дерево решения, с тем чтобы уменьшить ошибку этого решения.
3. Вычисляем пригодность решений в популяции P(i) на основе обучающей выборки. Если допустимое решение найдено – выводим результат, останавливаем алгортм; иначе - переходим к следующему шагу.
4. Применяем операторы селекции, клонирования, скрещивания и мутации к поколению P(i) и формируем новую популяцию P(i+1). Переходим к шагу 2.

Рассмотрим шаги алгоритма подробней.

Метод выращивания деревьев начальной популяции

Для инициализации популяции используется *полный метод (full method)* выращивания деревьев (решений в пространстве поиска):

Задается глубина дерева d. В вершины на глубине n (n= 1, …, d-1) случайным образом выбираются элементы из функционального множества F. На глубине d выбираются элементы из терминального множества T.

Причем, если выбран вещественный коэффициент, он выбирается случайно из интервала (-100;100) для $x \in R^1$. Т.о. получается полное дерево глубины d.

Замечание. Существуют еще *метод выращивания* и *комбинированный метод*, но в условиях нашей задачи они не целесообразны, так как при выращивании деревьев этими способами, во-первых, часто получаются усеченные деревья, что значительно уменьшает генетическое разнообразие, во-вторых, еще больше увеличивают случайность процесса нахождения решения. Первое может значительно увеличивать время работы программы. Последнее является отрицательным фактором, так как алгоритм вообще может не найти решение.

Метод оптимизации вещественных коэффициентов решения ОДУ

Решение представляет собой произвольную функцию, получающуюся в результате случайного выбора термов и операций на первоначальном этапе. Поэтому очевидно преимущество методов прямого поиска (методов нулевого порядка). Преимуществом этих методов является отсутствие дополнительных ограничений (существование производных и т.д.) и малая вычислительная работа компьютера при использовании этих методов. В результате мы применяем метод к любой произвольной функции и при вычислении тратим мало времени на вычисления и мало памяти компьютера.

Среди методов прямого поиска был выбран метод Хука-Дживса. По сравнению с другими методами он выигрывает своей неприхотливостью к оптимизируемой функции и невысокими машинными затратами на вычисления. Также в его пользу говорит то, что на практике он является универсальным.

Однако при использовании его совместно с алгоритмом генетического программирования возникла необходимость его изменить (точнее сказать, усечь). В связке с основным алгоритмом метод Хука-Дживса значительно увеличивает вероятность захвата алгоритмом ГП локальных минимумов, которые доставляет погрешность решения, то есть захватывается решение локально оптимальное и алгоритм начинает сходиться к нему.

Модификацией алгоритма служит уменьшение количества итераций при оптимизации коэффициентов и накладываются ограничения на начальный шаг метода и условия остановки метода (точность).

При генерации коэффициентов из интервала (-100;100) апостериорно оказалось целесообразно брать первоначальный шаг =10, а условие остановки или точность 0.1 (метод работает пока шаг меньше либо равен 0.1). в процессе работы шаг делится на 10, как и в оригинальном алгоритме.

При оптимизации берется каждый индивид в популяции (решение ОДУ), выполняется полный обход дерева. Если встретился вещественный коэффициент, он оптимизируется относительно погрешности (или функции пригодности). Таким образом, выполняется попытка оптимизации каждого коэффициента, который входит в решение ОДУ.

Замечание. Если выполнено улучшение решения хотя бы при одном изменении коэффициента дальнейший поиск по образцу не проводится, и оптимизация решения продолжается для следующего в дереве коэффициента.

Вычисление пригодности решения

Для нашей задачи пригодность (как и погрешность) – результат композиции пригодности решения для заданного ОДУ **(1)** и пригодности решения для начальных условий **(2)**.

Так функцию пригодности (fitness) можно представить так:

$$fitness(P(i)) = \frac{1}{1 + Error(P(i))} \cdot (20 - K1 \cdot Number(P(i))) + \sum_{j=1}^{n} K2 * \left| y(\bar{x}_i) - Y_i \right|,$$

$$Error(P(i)) = \sqrt{\sum_{i=1}^{N} (evaluate(P(i), \bar{x}_i))^2}$$

где fitness(P(i)) – значение функции пригодности, Error(P(i)) – квадратичная ошибка аппроксимации, вычисляемая по всем точкам выборки , K1 – коэффициент штрафа за сложность дерева (K1 $\in$ (0;20*Number/(2^max_depth)), где max_depth – заданная максимальная глубина дерева) , K2 – коэффициент штрафа за начальные условия, Y_i – заданные начальные условия , Number – число вершин дерева P(i), evaluate (P(i), $\bar{x}_i$)- значение дифференциального уравнения при подстановке туда решения P(i) в точке $\bar{x}_i$.

Рассмотрим подробнее, как вычисляется ошибка для каждого решения.

Значения $\bar{x}_i$, i=1,…,N заданы. Это точки по которым идет аппроксимация решения ОДУ. ОДУ строится подобно решению только терминальное множество у него больше: T = { x , y , $y^{|}$, $y^{||}$,…$y^{(n)}$,

вещественные коэффициенты}. Далее дерево, представляющее ОДУ обходится и вычисляется. Терм y заменяется на решение и вычисляется в точке $\bar{x}_i$, для термов $y^{|}$, $y^{||}$, …$y^{(n)}$ строится таблица значений исходя из решения $y = \varphi(x)$.

Производные аппроксимируются с помощью центральной разности:

$$f_{\overset{\circ}{x}} = \frac{1}{2} * (f_x + f_{\bar{x}}) = \frac{f(x+h) - f(x-h)}{2*h}$$

А применительно к решению, получаем следующую аппроксимацию первой производной $y = \varphi(x)$:

$$\varphi_{\overset{\circ}{x_i}} = \frac{1}{2} * (\varphi_{x_i} + \varphi_{\bar{x}_i}) = \frac{\varphi(x_{i+1}) - \varphi(x_{i-1})}{x_{i+1} - x_{i-1}}$$

Аналогично значения второй производной строятся по значениям первой производной.

Очевидно, что при порядке уравнения n нужно для функцию $y = \varphi(x)$ дополнительно n точек в начале и n точек в конце, для чтобы n-я производная была вычислена в n точках.

Итак, получаем таблицу значений размером [n , N], количество строк совпадает с порядком уравнения, а число столбцов – с объемом выборки.

Таким образом, с помощью решения $y = \varphi(x)$ и таблицы значений производных вычисляем ошибку(пригодность) в ОДУ и начальных условиях.

Операторы генетического программирования

Оператор селекции – оператор, посредством которого индивиды выбираются для порождения потомков. Селекция обеспечивает возрастание среднего значения функции пригодности по популяции. Наиболее приспособленные особи должны выбираться с большей вероятностью для сохранения своих генов в следующем поколении. Таким образом, оператор селекции позволяет сконцентрировать поиск на наиболее многообещающих регионах пространства поиска.

Существует большое число различных моделей селекции. Но наиболее подходящими в условиях задачи являются ранговая и турнирная селекция. Также эти виды селекции мало склонны к преждевременной сходимости как, например, пропорциональная и элитарная селекции. И хотя существует много способов избавиться от этого недостатка, но эти способы значительно

увеличивают количество операции, что ведет к увеличению времени работы программы.

Рассмотрим подробнее ранговую и турнирную селекцию.

Ранговая селекция .

Индивиды сортируются (ранжируются) на основе их пригодности таким образом, чтобы $f_i \geq f_j$ для $i \leq j$, т.е. первым стоит наиболее приспособленный индивид. Затем каждому индивиду назначается вероятность p_i быть отобранным, взятая из заданного распределения с ограничением $\sum_i p_i = 1$.

Типичные распределения:

1. Линейное: $p_i = a \cdot i + b$ ($a < 0$).
2. Отрицательное экспоненциальное: $p_i = a \cdot exp(b \cdot i + c)$. Это эквивалентно назначению первому индивиду вероятности p, второму - p^2, третьему - p^3, и т.д.

Турнирная селекция.

Для отбора индивида создается группа из m ($m \geq 2$) индивидов, выбранных случайным образом. Индивид с наибольшей пригодностью в группе отбирается, остальные – отбрасываются. Параметр m называется размером турнира. Наиболее популярным является бинарный турнир.

Скрещивание является генетическим оператором поиска. Посредством скрещивание родительские гены переходят в хромосому потомка, тем самым, придавая ему новые свойства. Если эти свойства полезны, то, скорее всего они и далее останутся в популяции.

Скрещивание осуществляется следующим образом. Выбираются родительская пара. У каждого из родителей выбирается точка скрещивания (дуга в графе). Родители обмениваются генами (поддеревьями), находящимися ниже точки скрещивания. Полученная пара является потомками.

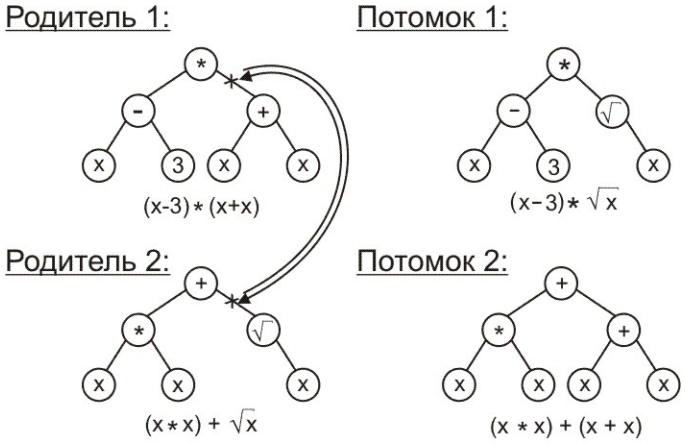


Рис. 1. Пример скрещивания в методе генетического программирования

При этом потомок, который переходит в новую популяцию выбирается случайным образом.

Замечание. Существует еще вариант скрещивания, когда точки скрещивания выбираются в одной точке дерева. Деревья накладываются друг на друга, выбирается общая точка и в этой точке они обмениваются поддеревьями. Но для нашей задачи случайный выбор точек скрещивания обеспечивает большее генетическое разнообразие, поэтому более предпочтителен.

Мутация состоит из выполнения (обычно небольших) изменений в значениях одного или нескольких генов в хромосоме. Мутация рассматривается как метод восстановления потерянного генетического материала, а не как поиск лучшего решения. Мутация применяется обычно с очень низкой вероятностью. Хорошим эмпирическим правилом считается выбор вероятности мутации из соотношения $p_m = \square$, где M - число бит в хромосоме (число узлов в дереве). Однако существует ряд задач, решаемых с помощью метода генетического программирования, где допустимое решение может быть найдено экстенсивным использованием оператора мутации.

При использовании *точечной мутации*, случайно выбранный узел в дереве меняется на случайно выбранный элемент того же типа. Т.е. выбранный ген заменяется случайно выбранным элементом терминального

множества, если этот ген принадлежит терминальному множеству, или элементом функционального, если ген – элемент функционального множества (рис. 2.2).

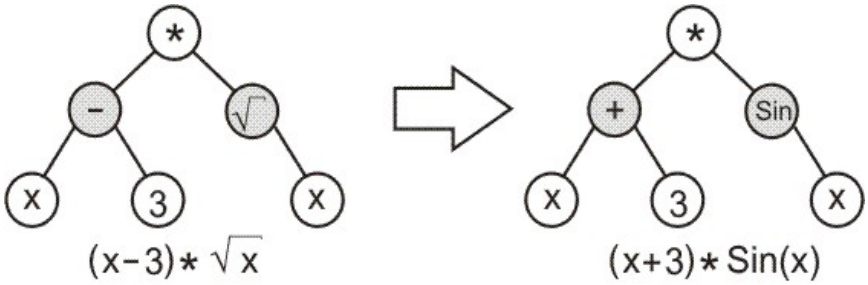


Рис. 2. Пример мутации.

Итак, алгоритм состоит из четырех этапов. На первом этапе один раз генерируется начальная популяция, остальные этапы повторяются. Таким образом, от поколения к поколению находится все более пригодное решение.

Примеры решения задач

Алгоритм генетического программирования реализован в программе, написанной в среде Microsoft Visual Studio 6.0 на языке C++. Программа включает символьный интерпретатор дифференциального уравнения, что позволяет работать с произвольным уравнением. Интерпретатор символьного выражения применяется также для построения дерева решения по введенной строке решения и для построения функции-ответа для тестовых примеров.

Программа работает относительно долго, но, так как задача является нетривиальной, время работы все-таки является приемлемым. Также нужно отметить, что время работы в значительной степени зависит от сложности дифференциального уравнения, а главной от сложности функции, являющейся решением, и элементов, входящих в функциональное множество F. Так один из наиболее сложных случаев – это аппроксимация тригонометрических функций когда F ={+,-,*,/}. Окно программы представлено на рисунке 3.

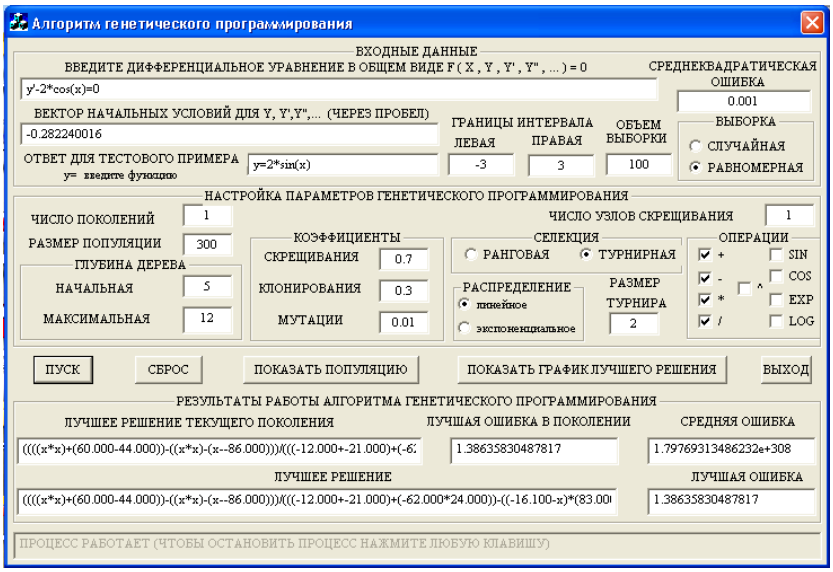


Рис 3. Рабочее окно программы

Результаты работы программы

В процессе тестирования рассматривались наиболее сложные задачи.

Задача1.

y’-2*cos(x)=0, x∈[-3,3],

y(0)= -0.282240016; Ответ– 2*sin(x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /

Размер популяции:	300
Начальная глубина деревьев:	6
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение:

(((x-((x-(((x+6.200)+-6.200)*(x*(x/48.900))))*(x*(x/31.900))))*(((((x*(x/29.700))+-4.000)+-6.200)+(x*x))/8.400))*((((((x+-0.000)*(x/37.200))+-15.000)+-15.300)+(x*x))/18.400))

После упрощения получается следующее выражение

0.01662624753*x^5 - 0.3330926297*x^3 + 1.999611801*x - 0.0003886344121*x^7 + 0.4402564200*10^(-5) *x^9

Оно совпадает с разложением функции 2*sin(x) в ряд Тейлора.

Ошибка решения - 0.00252728996473392

Программа работала приблизительно два часа.

Задача2.

y’-2*cos(x)=0, $x \in [-3,3]$,

y(0)= -0.282240016; Ответ– 2*sin(x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /,sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев:	6
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение: ((sin(x))/(0.500))

Ошибка решения 3.74875e-008, программа работала приблизительно полчаса.

Задача3.

y’’+2*sin(x)=0, x∈[-3,3],

y(0)= -0.282240016 -1.979984993; Ответ– 2*sin(x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /
Размер популяции:	300
Начальная глубина деревьев:	6
Максимальная глубина	12
Селекция:	Турнирная (2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение:

(((((((-22.000)-((((x)+((x)-(-20.400)))+(x))*((x)*(106.600))))/((((x)+(-78.600))-((x)-(-20.700)))+(((-105.800)-(148.100))-((72.800)+(x)))))*(((((((-33.000)/(1.600))-((-38.300)*(x)))-(((93.300)-(-51.600))+((x)-(14.300))))+(-104.600))+((x)/(57.300)))*(((x)-(x))+((x)/(4.900)))))+((178.200)*(23.400)))/((((x)+(-141.100))-((x)-(-121.000)))+(((-196.800)-(234.300))-((184.300)+(x)))))*(((((((196.000)/(26.700))+((((x)-(-25.900))+((x)*(3.900)))/(((72.000)+(2.000))+((x)-(64.000)))))-((((x)*(40.300))/((1.900)+(-30.300)))*(x)))-(((20.100)-(-17.300))+((x)-(61.900))))+(-48.100))+((x)/(1.300)))*(((x)-(x))+((x)/(32.600)))))

Ошибка решения - 0.00788372

Программа работала приблизительно два часа.

Задача4.

y’’-2*cos(x)=0, x∈[-3,3],

y(0)= -0.282240016 -1.979984993; Ответ– 2*sin(x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /,sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев::	6
Максимальная глубина	12

Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение: ((2.000)*(sin(x)))

Ошибка решения 8.30794e-008

Программа работала приблизительно 20 минут.

Задача5.

x*y+(x+1)*y'=0, x∈[0,3], y(0)= 1.0000; ответ: y= (x+1)*exp(-1*x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /
Размер популяции:	300
Начальная глубина деревьев::	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100

Было получено следующее решение

(((((((x*(((x*215.900)*(-149.000-44.100))-21.500))/(((-24.000+x)-(30.700+-57.800))+((x-0.600)*(x*x))))+x)-(((x*235.700)*(-135.700-16.000))-((139.600-x)-(x+x))))/((((10.600-(((x*124.700)*(-100.800--12.400))-((44.700-x)-(x+x))))/(((-18.100+x)-(17.300+-72.000))+((x-7.100)*(x*x))))-(((x*12.000)*(-49.900--36.900))-((128.900-x)-(x+x))))*((0.300--10.600)-(33.900/-146.700))))-(((x*76.900)*(-44.900--45.100))-((131.000-x)-(x+x))))/(((28.000+x)-(-7.100+-96.000))+((x--24.700)*(x*x))))

Ошибка составила 0.00402627846039396

Программа работала приблизительно 3 часа.

Задача6.

x*y'+y-y*y=0, x∈[-0,8,6],

y(-0.8)= 5;

Ответ y=1/(1+x)

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /
Размер популяции:	300
Начальная глубина деревьев::	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение

(((((2.200-((-6.700*3.300)/((x*60.100)-(x+-67.800))))-((x/(((1.800-x)-x)+52.500))/((x*((x--10.300)/(-8.600+8.800)))-(-62.800+24.300))))-((x/(((-3.400-x)-x)+54.000))/((x*((x--10.300)/(-8.600+8.800)))-(-62.800+24.300))))/((((11.200/(100.100+(-83.700+-68.300)))*75.200)/(7.300*x))*((x-x)-(-1.100-x))))--1.000)

После упрощения получается следующее выражение

$$-\frac{0.9896395326\,x}{x+1.100}-\frac{9.945877303\,x}{(x+1.100)\,(59.100\,x+67.800)}+\frac{0.4498361512\,x^2}{(x+1.100)\,(54.300-2\,x)\,(5.000000000\,x^2+51.50000000\,x+38.500)}+\frac{0.4498361512\,x^2}{(x+1.100)\,(50.600-2\,x)\,(5.000000000\,x^2+51.50000000\,x+38.500)}+1.000$$

Ошибка составила 0.0133882504642858

Программа работала приблизительно 3 часа.

Задача7. $y'-\frac{1}{\sqrt{2\pi}}e^{-x^2/2}=0$ - интеграл Лапласа. $x\in[0,3]$, $y(0)=0$;

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /, sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев::	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя

Объем выборки	100
Метод выращивания	полный

Было получено следующее решение

(((((((25.800)-((((6.900)-((x)+(x)))*(sin((x)+(-0.800))))+(x)))-((12.600)+(x)))*((((sin((16.900)-(48.400)))*(sin((x)+(4.600))))*(exp(((x)/((91.700)+(x)))-(sin(-35.200)))))/(exp((437.400)*((x)/(exp(13.800)))))))*(sin((x)+(-25.100))))-((-5.200)+(x)))*(((x)/(5.000))/(log(13.600))))

После упрощения получается следующее выражение на следующей странице

$$\frac{0.005229415870\ x\ \sin(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}+\frac{0.04650481794\ x\ \sin(x)\ \cos(x)\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$+\frac{0.001517559312\ x\ \cos(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}-\frac{0.001904488404\ x\ \sin(x)^3\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$-\frac{0.01497554325\ x\ \sin(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}\cos(x)}{e^{(0.0004442372054\ x)}}$$

$$+\frac{0.01688577443\ x\ \sin(x)\ \cos(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$+\frac{0.0005690568082\ x\ \cos(x)^3\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}+\frac{0.0005520256249\ x^2\ \sin(x)^3\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$+\frac{0.004340737173\ x^2\ \sin(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}\cos(x)}{e^{(0.0004442372054\ x)}}$$

$$-\frac{0.004894427372\ x^2\ \sin(x)\ \cos(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$-\frac{0.0001649440025\ x^2\ \cos(x)^3\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$-\frac{0.0007923357382\ x^2\ \sin(x)^2\ e^{\left(\frac{x}{91.700+x}\right)}}{e^{(0.0004442372054\ x)}}$$

$$-\frac{0.007046184533\ x^2 \sin(x)\cos(x)\ \mathbf{e}^{\left(\frac{x}{91.700+x}\right)}}{\mathbf{e}^{(0.0004442372054\ x)}}$$

$$-\frac{0.0002299332291\ x^2 \cos(x)^2\ \mathbf{e}^{\left(\frac{x}{91.700+x}\right)}}{\mathbf{e}^{(0.0004442372054\ x)}} + 0.3984567780\ x - 0.07662630346\ x^2$$

Ошибка составила 0.00263135

Программа работала приблизительно 4 часа.

Задача8. $\sqrt{2\pi}y''+\sqrt{2\pi}y'-e^{-x^2/2}(1-x)=0$ - интеграл Лапласа. $x \in [0,3]$,

$y(0)= 0$, $y'(0)=\frac{1}{\sqrt{2\pi}}$;

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * ,^, /, sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев:	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение

((((((x)/(4.900))/(74.400))*((exp(cos((cos((x)*(sin(59.200))))*((exp(cos(30.400)))+((sin(133.200))-((-21.900)/(-81.500))))))))+((sin(123.700))-((-39.200)/(-92.500)))))+(sin (((sin((x)*((exp(cos(39.7)))+((sin(96.8))-((-114.4)/(-81.3))))))*((exp(cos(19.8))) +((sin(95.000))-((-100.900)/(-58.7)))))+(sin(sin((x)/(3.7)))))))+(sin(((sin(((x)+(exp(-68.000)))*((exp(cos(39.100)))+((sin(96.800))-((-104.800)/(-81.100))))))*((exp(cos(19.700)))+((sin(95.000))-((-103.000)/(59.800)))))+(sin(sin((x)/(2.100))))))))

После упрощения получается следующее выражение

$$0.002743032698\, x\, e^{\cos(2.374808248\cos(0.4708556478\, x))} - 0.003696463275\, x$$
$$- \sin(0.752752803\sin(0.192243714\, x))\cos(\sin(\sin(0.2702702703\, x))))$$
$$+ \cos(0.752752803\sin(0.192243714\, x))\sin(\sin(\sin(0.2702702703\, x))) +$$
$$\sin(0.894967906\sin(0.447836383\, x))$$
$$\cos(0.1177340451\ 10^{-29}\cos(0.447836383\, x))\cos(\sin(\sin(0.4761904762\, x))) -$$
$$\sin(0.894967906\sin(0.447836383\, x))$$
$$\sin(0.1177340451\ 10^{-29}\cos(0.447836383\, x))\sin(\sin(\sin(0.4761904762\, x))) +$$
$$\cos(0.894967906\sin(0.447836383\, x))$$
$$\sin(0.1177340451\ 10^{-29}\cos(0.447836383\, x))\cos(\sin(\sin(0.4761904762\, x))) +$$
$$\cos(0.894967906\sin(0.447836383\, x))$$
$$\cos(0.1177340451\ 10^{-29}\cos(0.447836383\, x))\sin(\sin(\sin(0.4761904762\, x)))$$

Ошибка составила 0.0250383

Программа работала приблизительно 5 часа.

Задача 9. $y' - \frac{\sin(x)}{x} = 0$ - интегральный синус. $x \in [0,1,10]$, y(0,1)= 0.09994;

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /,sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев::	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение

(((((((cos(((x)+(x))*((x)/(52.200))))*((x)*(-50.300)))/(((cos(22.100))*((x)*(17.700)))-(((72.300)-(48.200))/((x)/(11.300)))))-((x)/(0.100)))/(((cos(cos(cos(9.400))))*((x)*(-0.300)))-(((156.100)-(32.900))/((x)/(2.500)))))+((log(exp(-8.200)))+((cos(x))*(sin(-6.700)))))+((log(exp(10.900)))+((cos(x))*(sin(3.100)))))/(((cos(52.000))*((x)*(-2.100)))-(((66.400)-(67.300))/((x)/(2.600)))))

После упрощения получается следующее выражение

$$-50.300\cos(0.03831417624\,x^2)\,x \Big/ \left(\left(0.3422806397\,x+\frac{2.340000000}{x}\right)\left(-0.2572261002\,x-\frac{308.0000000}{x}\right)\left(-17.59524310\,x-\frac{272.3300000}{x}\right)\right)$$

$$-\frac{10.00000000\,x}{\left(0.3422806397\,x+\frac{2.340000000}{x}\right)\left(-0.2572261002\,x-\frac{308.0000000}{x}\right)}+\frac{2.700000000}{0.3422806397\,x+\frac{2.340000000}{x}}-\frac{0.3632692582\cos(x)}{0.3422806397\,x+\frac{2.340000000}{x}}$$

Ошибка составила 0.00104538

Программа работала приблизительно 3 часа.

Задача10. $y''+\frac{y'}{x}-\frac{\cos(x)}{x}=0$ - интегральный синус. $x\in[0,1,10]$, y(0,1)= 0.09994, y’(0,1)= 0.998334166;

Терминальное множество:	$\{x, C\}$
Функциональное множество:	+ , - , * , /,sin,cos,exp,log
Размер популяции:	300
Начальная глубина деревьев::	5
Максимальная глубина	12
Селекция:	Турнирная(2)
Вероятность мутации:	средняя
Объем выборки	100
Метод выращивания	полный

Было получено следующее решение

((((((((x)+(-1.600))+((((x)+(8.800))*((1.800)/(-13.600)))*(((x)/(x))+(cos(x)))))+(sin((x)/(4.900))))/(exp(((x)+(2.000))*((-21.600)/(-35.100)))))+(1.800))+(((x)*((-2.100)/(-65.0)))*(((x)/(x))+(cos(x)))))+((((x)+(30.6))*((0.9)/(-90.200)))*(((x)/(x))+(cos(x)))))

После упрощения получается следующее выражение

$$\frac{0.2534117881\,x}{e^{(0.6153846154\,x)}}-\frac{0.8074816301}{e^{(0.6153846154\,x)}}-\frac{0.03865603548\,x\cos(x)}{e^{(0.6153846154\,x)}}-\frac{0.3401731124\cos(x)}{e^{(0.6153846154\,x)}}+\frac{0.2920678236\sin(0.2040816327\,x)}{e^{(0.6153846154\,x)}}+1.494678492+0.02232986526\,x+0.02232986526\,x\cos(x)-0.3053215078\cos(x)$$

Ошибка составила 0.0220934

Программа работала приблизительно 4 часа.

Как показывают рассмотренные примеры, алгоритм генетического программирования открывает новые возможности при решении дифференциальных уравнений – он позволяет находить в символьном виде решения таких уравнений, которые не интегрируются обычным способом, например, для решения которых надо попутно взять «неберущийся» интеграл.

Реализация алгоритма ГП для данной задачи – лишь один из возможных вариантов. Возможно, он не является оптимальным, в частности не решен вопрос, как наиболее эффективно использовать время работы алгоритма, в случае, когда оно ограничено. Поэтому целесообразно продолжать дальнейшие исследования в области решения поставленной задачи. Возможно, будут найдены более эффективные способы решения задачи.

Эволюционный алгоритм оптимизации структуры нейросетевой модели

Искусственные нейронные сети

Нейронные сети представляют собой модель строения и процессов, происходящих в коре головного мозга. Искусственная нейронная сеть (ИНС) – это набор формальных нейронов (ФН), соединенных между собой.

Формальный нейрон представляет собой упрощенную модель биологического нейрона. Структура формального нейрона представлена на рис. 1.

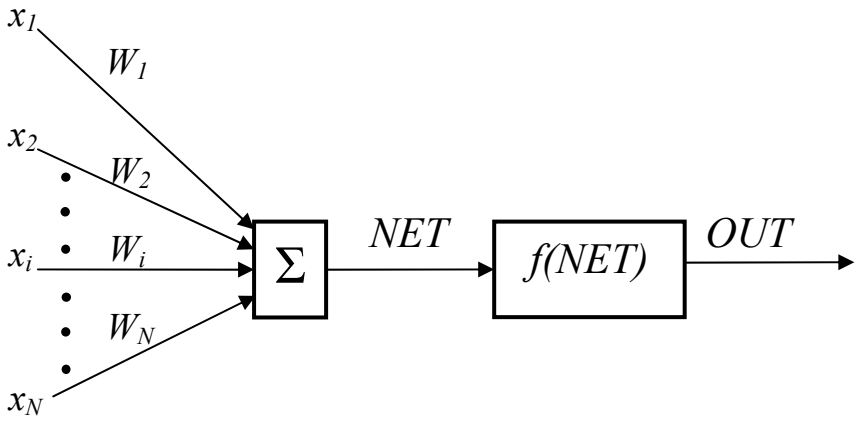


Рисунок 1 – Искусственный нейрон

С конструктивной точки зрения формальный нейрон – это устройство получения функции нескольких переменных с возможностью настройки его параметров. В общем случае нейрон имеет один выход и несколько входов (синапсов), которые осуществляют связь между нейронами. Математическая модель нейрона имеет следующий вид:

$$OUT = f(NET),$$

$$NET = \sum_{i=1}^{N} W_i \cdot x_i ,$$

где OUT – величина сигнала на выходе нейрона;

f – функция активации нейрона;

x_i – величина сигнала на i-м входе нейрона;

W_i – весовой коэффициент i-го входа нейрона.

Синаптические связи с положительными весами называют возбуждающими, а с отрицательными – тормозящими.

В соответствии с данной моделью сигнал на выходе нейрона формируется следующим образом: сумматор осуществляет сложение взвешенных входов, а преобразователь реализует некоторую активационную функцию от выхода сумматора. В общем случае активационная функция является нелинейной, однако практикуют и линейные функции. Чаще всего вид нелинейности не оказывает принципиального влияния на решение задачи. Однако удачный выбор может сократить время обучения нейросети в несколько раз. В искусственных нейросетях получили распространение следующие основные виды активационных функций, представленные в табл. ниже.

Таблица 1.

Типы активационных функций в искусственных нейронах

Наименование	Формула	Область значения
Пороговая	$f(s)=\begin{cases}0, & s<\alpha\\ 1, & s\geq\alpha\end{cases}$	[0, 1]
Знаковая	$f(s)=\begin{cases}-1, & s\leq\alpha\\ 1, & s>\alpha\end{cases}$	[-1, 1]
Сигмоидальная	$f(s)=\frac{1}{1+e^{-s}}$	(0, 1)
Полулинейная	$f(s)=\begin{cases}0, & s\leq 0\\ s, & s>0\end{cases}$	[0, ∞)
Линейная	$f(s)=\alpha\cdot s$	(-∞, ∞)
Радиальная базисная (гауссова)	$f(s)=e^{-s^2}$	(0, 1)
Полулинейная с насыщением	$f(s)=\begin{cases}0, & s\leq 0\\ s, & 0<s<1\\ 1, & s\geq 1\end{cases}$	[0, 1]
Линейная с насыщением	$f(s)=\begin{cases}-1, & s\leq -1\\ s, & -1<s<1\\ 1, & s\geq 1\end{cases}$	[-1, 1]
Гиперболический тангенс	$f(s)=\frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$	(-1, 1)

Треугольная	$f(s)=\begin{cases}1-\lvert s\rvert, & s\le 1\\ 0, & \lvert s\rvert>1\end{cases}$	[0, 1)

Классификация ИНС

Существуют различные способы классификации нейронных сетей.

С точки зрения топологии выделяют следующие типы ИНС:

- полносвязные;
- многослойные;
- слабосвязные (с локальными связями);
- неструктурированные (многосвязные) – к ним относят все нейросети, которые нельзя отнести ни к одной из предыдущих групп.

Возможны различные комбинации этих типов. Наиболее общим случаем является многосвязная ИНС, где каждый нейрон может быть связан с любым другим нейроном сети.

По характеру распространения сигнала нейронные сети подразделяют на:

- монотонные – частный случай многослойных сетей с дополнительными условиями на связи и нейроны;
- сети с прямым распространением – сигнал распространяется только в одном направлении (от входов к выходам);
- сети с обратными связями (рекуррентные) – выходы сети подаются на входы.

По типу сигнала нейронные сети подразделяют на:

- бинарные (двоичный сигнал);
- аналоговые (непрерывный сигнал).

По типам структур нейронов различают:

- гомогенные ИНС;
- гетерогенные ИНС.

По характеру изменения состояния нейронов во времени выделяют:

- синхронные ИНС;
- асинхронные ИНС.

К настоящему времени разработано и исследовано большое количество разновидностей нейросетевых моделей, специфически реализующих вышеперечисленные характеристики с целью реализации различных свойств биологических систем. Среди наиболее известных следует назвать:

- однонаправленные сети (многослойные персептроны);

- сети радиальных базисных функций;
- сети Хопфилда;
- сети Хемминга;
- сети Кохонена;
- сети встречного распространения;
- ДАП – двунаправленная ассоциативная память (нейронная сеть Коско);
- АРТ-сети (сети адаптивной резонансной теории), предложены Карпентером и Гроссбергом;
- когнитрон (автор К. Фукушима);
- неокогнитрон.

Наиболее общим случаем нейросетевых структур является многосвязная ИНС, где каждый нейрон может быть связан с любым другим нейроном сети.

Как правило, активационные функции всех нейронов сети фиксированы, а веса являются параметрами сети и могут изменяться. Функционирование нейросети состоит в преобразовании входного вектора в выходной вектор, причем это преобразование задается весовыми коэффициентами сети.

Процесс построения нейросетевой модели включает в себя два этапа:

1. Выбор структуры ИНС.
2. Настройка параметров (обучение) ИНС заданной структуры.

Методы настройки параметров ИНС

Настройка ИНС с априори заданной структурой заключается в поиске некоторого набора весовых коэффициентов, такого, что полученная нейросетевая модель в процессе эксплуатации позволяла бы наилучшим образом решать поставленные задачи.

Существуют три парадигмы обучения ИНС:

- «с учителем»;
- «без учителя» (самообучение);
- смешанная.

Обучение с учителем предполагает, что для каждого входного вектора существует соответствующий ему целевой вектор, представляющий собой требуемый выход (обучающая пара). Совокупность обучающих пар является

обучающей выборкой. В процессе обучения на входы нейросети в случайном порядке подаются входные векторы из обучающей выборки, и веса связей подбираются таким образом, чтобы минимизировать суммарное по всем обучающим парам расхождение между реальным и требуемым выходами – ошибку обучения нейросети.

При обучении без учителя имеется только заданный набор входных векторов, для каждого из которых в ходе настройки нейросети подбирается наилучший в некотором смысле, определяемом алгоритмом обучения, выходной вектор. К алгоритмам обучения этой группы относятся метод Хебба, метод соревнования, проекции Саммона, анализ главных компонентов, алгоритм Adaline и Madaline, алгоритм Кохонена, линейный дискриминантный анализ.

Смешанное обучение представляет собой промежуточный вариант между первыми двумя подходами к обучению: весовые коэффициенты одной группы нейронов настраиваются путем обучения с учителем, а другой группы – без учителя. Наиболее удачная модель, использующая подобный способ обучения – это сеть с линейным поощрением.

В том случае, когда имеется обучающая выборка и задан способ вычисления функции ошибки обучения, задача обучения ИНС представляет собой задачу многомерной оптимизации, которая заключается в поиске оптимального набора весов нейронной сети, минимизирующего целевую функцию ошибки. Можно выделить две большие группы алгоритмов, решающих эту задачу, – градиентные и стохастические. Градиентные алгоритмы обучения сетей основаны на вычислении частных производных функции ошибки по параметрам сети. К данной группе относятся, например, такие алгоритмы как градиентный алгоритм (наискорейшего спуска) и алгоритм сопряженных градиентов. Наиболее эффективным алгоритмом первой группы является алгоритм обратного распространения ошибки (error back-propagation algorithm), однако он применим только для обучения полносвязных сетей с послойным распространением сигнала. Градиентные методы обладают рядом недостатков, которые ограничивают их применение при решении рассматриваемой задачи – это локальная сходимость, зависимость от расположения точки старта, зависимость от выбора длины шага. В стохастических алгоритмах поиск минимума функции ошибки ведется случайным образом. К ним относятся такие алгоритмы как имитация

отжига, метод Монте-Карло, поиск в случайном направлении. Применение стохастических оптимизационных алгоритмов при построении нейросетевой модели требует выполнения большого количества вычислений. Из алгоритмов данной группы широкое распространение получили методы генетической адаптации.

Выбор структуры ИНС

Одним из неформализованных моментов практического применения нейросетевого моделирования является выбор структуры ИНС, которая определяется количеством нейронов, их активационными функциями, а также наличием связей между конкретными нейронами. Зачастую структура нейросети выбирается, исходя из особенностей решаемой задачи, а также на основе рекомендаций специалиста, имеющего достаточный опыт работы с подобными системами.

Можно выделить два диаметрально противоположных подхода, которые применяются при решении задачи выбора эффективной структуры ИНС, при этом каждый из подходов реализуется соответствующей группой алгоритмов:

1) деструктивные алгоритмы (алгоритмы сокращения);

2) конструктивные алгоритмы.

Сущность первого подхода заключается в том, что изначально задается сеть большего объема, чем это необходимо для решения задачи. После того, как сеть выбранной конфигурации обучилась на задачу, используя определенные методики, удаляют мало значимые нейроны и связи до тех пор, пока ошибка нейросети увеличивается незначительно. Процедура корректировки структуры ИНС прекращается, когда последующее изменение влечет за собой значительное увеличение ошибки. Для многослойных сетей обычно используют метод штрафных функций или метод проекций. Для сетей произвольной структуры эффективно применяется метод контрастирования. Идея данного метода заключается в том, что для каждого параметра НС (связи, нейрона, входа) вычисляется некоторый показатель значимости. Удаляем из сети параметр с наименьшими показателями значимости, после чего показатель значимости пересчитывается. Показателем значимости параметра χ_p^q при решении q-го примера называют величину, которая показывает, насколько изменится значение функции

оценки решения сетью q-го примера, если текущее значение параметра w_p изменить на ближайшее выделенное значение w_p^*. Существует несколько способов получения показателя значимости параметра, не зависящего от примера. Например, можно усреднить показатель значимости по всему обучающему множеству или выбрать максимальный из них.

Алгоритмы сокращения имеют, по крайней мере, три недостатка:

- не существует методики предварительного определения избыточного количества нейронов и связей для конкретной задачи;

- в процессе работы алгоритма сеть имеет избыточную структуру, что замедляет процесс обучения;

- неясно, что явилось причиной в том случае, когда процесс обучения завершился неудачей – избыточность структуры ИНС или, наоборот, недостаточная ее сложность.

Второй подход предусматривает постепенное наращивание некоторой изначально малой по объему структуры ИНС. Нейросеть с заданной структурой обучают, и в случае успешного обучения останавливаются на полученном варианте конфигурации сети, в противном случае добавляют нейроны или связи. При этом сохраняются навыки, приобретенные сетью до увеличения количества нейронов и связей.

Конструктивные алгоритмы различаются следующими особенностями реализации.

- Способами добавления новых нейронов и связей к имеющейся структуре ИНС: добавление случайным образом, добавление с целью минимизации ошибки обучения, эволюционные методы наращивания структуры.

- Способами задания параметров в новых, добавляемых в сеть, нейронах: весовые коэффициенты выбираются случайным образом или определяются путем расщепления одного из старых нейронов.

Конструктивные алгоритмы лишены недостатков, присущих алгоритмам сокращения. Однако если мы допускаем сети с межслойными и неполными связями, с различными видами нейронов, то не совсем ясно, каким образом на каждом шаге увеличивать архитектуру. Причем с каждым следующим шагом количество возможных вариантов увеличивается в несколько раз.

Общим недостатком обоих подходов является то, что они представляют собой локальные процедуры поиска в пространстве структур ИНС, и останов может произойти в ближайшей, более или менее удачной, структуре, которая будет далека от оптимальной.

Выбор эффективной структуры ИНС представляет собой многопараметрическую оптимизационную задачу, аналогично задаче настройки нейросетевых параметров. Для решения подобного рода задач успешно применяются генетические алгоритмы (ГА).

Приведем существенные аргументы в пользу применения ГА для структурного и параметрического синтеза ИНС.

Генетические алгоритмы:

- подходят для оптимизации нейросетевых моделей с произвольной структурой;
- обеспечивают глобальный просмотр пространства решений и позволяют избегать локальных минимумов;
- могут использоваться в задачах, для которых информацию о градиентах получить невозможно;
- потенциально обладают свойством массового параллелизма при обработке, поскольку особи в популяции функционируют независимо, что допускает эффективную параллельную реализацию ГА.

Несмотря на такую привлекательность генетических алгоритмов, невозможно дать гарантию того, что полученное решение будет хотя бы локальным экстремумом. Эффективность генетического алгоритма существенно зависит от его настроек.

Адаптация ГА для обучения ИНС

Эволюционный подход к обучению нейронных сетей состоит из двух основных этапов:

1. Выбор соответствующей схемы кодирования весов связей.
2. Выполнение процесса эволюции, основанного на генетическом алгоритме.

После выбора схемы хромосомного представления генетический алгоритм применяется к популяции особей (хромосом, содержащих закодированное множество весов нейронной сети) с реализацией типового цикла эволюции, состоящего из четырех шагов:

1. Декодирование каждой особи (хромосомы) текущего поколения для восстановления множества весов и конструирования соответствующей этому множеству нейронной сети с априорно заданной архитектурой.

2. Расчет общей среднеквадратичной погрешности между фактическими и заданными значениями на всех выходах сети при подаче на ее входы обучающих образов. Эта погрешность определяет приспособленность особи (сконструированной сети).

3. Репродукция особей, согласно выбранному способу селекции.

4. Применение генетических операторов – таких как скрещивание и мутация для получения нового поколения.

В качестве пригодности i-го индивида можно использовать функцию от общей среднеквадратической ошибки обучения:

$$f_i = \frac{1}{1 + GenError_i},$$

$$GenError_i = \frac{\sum_{j=1}^{m-1} Error_j}{m},$$

$$Error_j = \sqrt{\frac{\sum_{k=1}^{n} (OUT_k - y_k)^2}{n}},$$

где OUT_k – реальное выходное состояние нейрона k выходного слоя нейронной сети при подаче на ее входы j-го образа; y_k – идеальное (желаемое) выходное состояние этого нейрона.

Адаптация ГА для выбора структуры ИНС

Также как и в случае эволюции весов связей, на первом этапе эволюционного проектирования архитектуры принимается решение относительно соответствующей формы ее описания. Однако в данной ситуации проблема не связана с выбором между двоичным и вещественным представлением (т.е. действительными числами), поскольку речь может идти только о дискретных значениях. Необходимо выбрать более общую концептуальную структуру представления данных, например, в форме матриц, графов. Ключевой вопрос состоит в принятии решения о количестве информации об архитектуре сети, которая должна кодироваться соответствующей схемой. С одной стороны, полная информация об

архитектуре может непосредственно кодироваться в виде двоичных последовательностей, то есть каждая связь и каждый узел (нейрон) прямо специфицируется определенным количеством битов. Такой способ представления называется схемой непосредственного кодирования (сильная схема спецификации). С другой стороны, могут представляться только важнейшие параметры или свойства архитектуры – такие как количество узлов (нейронов), количество связей и вид активационной функции на нейроне. Этот способ представления называется схемой косвенного кодирования (слабая схема спецификации).

Остановимся подробнее на схеме непосредственного кодирования.

С целью кодирования необходимой информации, описывающей структуру ИНС, каждая связь нейронной сети непосредственно задается ее двоичным представлением. Матрица C размерностью $n \times n$, $C = [c_{ij}]_{n \times n}$ может представлять связи нейронной сети, состоящей из n узлов (нейронов), причем значение c_{ij} определяет наличие или отсутствие связи между i-м и j-м нейронами. Если $c_{ij} = 1$, то связь существует, если $c_{ij} = 0$, то связь отсутствует. При таком подходе двоичная последовательность (хромосома), представляющая связи нейронной сети, имеет вид комбинации строк (или столбцов) матрицы C.

Будем принимать во внимание только однонаправленные связи, что позволяет учитывать только те элементы матрицы C, которые задают связи данного узла (нейрона) со следующими узлами.

В общем виде матрица связей многосвязной однонаправленной нейронной сети будет выглядеть так, как показано на рис. 2.

Нетрудно видеть, что такая матрица связей обладает следующими особенностями, которые необходимо учесть при кодировании хромосом. А именно:

- первые столбцы матрицы связей, соответствующие входным нейронам, всегда содержат нули, поскольку никакие нейроны не могут быть связаны с нейронами входного слоя;

- последние строки матрицы, соответствующие выходным нейронам, всегда содержат нули, так как нейроны выходного слоя не могут быть связаны ни с какими нейронами;

- элементы матрицы связей, расположенные ниже главной диагонали, всегда равны нулю, так как мы учитываем только однонаправленные связи;

- на главной диагонали матрицы всегда стоят нули, поскольку нейроны не могут быть связаны сами с собой.

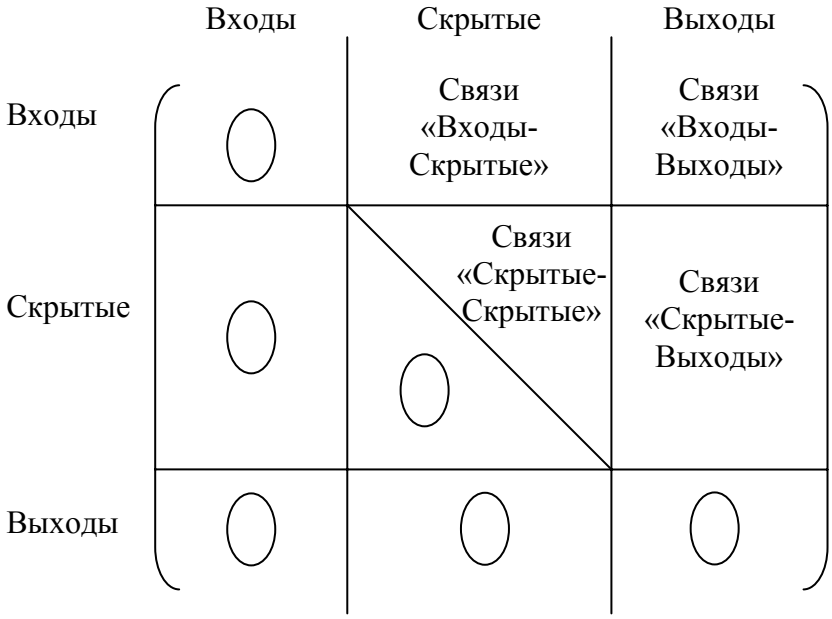


Рисунок 2 – Обобщенное представление матрицы связей

Из вышесказанного следует, что при кодировании хромосомы такие нули можно опустить.

Кроме этого, хромосома должна содержать информацию о виде активационной функции на каждом нейроне (двоичный порядковый номер этой функции в наборе всех используемых при решении задачи активационных функций). Данную информацию можно добавить, например, в конец последовательности бит соответствующего нейрона. Для случая восьми функций это будут три бита (от 000 для первой активационной функции до 111 для восьмой), в случае шестнадцати функций это будут четыре бита (от 0000 для первой активационной функции до 1111 для шестнадцатой). На входном слое активационная функция для каждого нейрона всегда имеет вид $F\_activ\_inp(x) = x$, и поэтому при кодировании не учитывается.

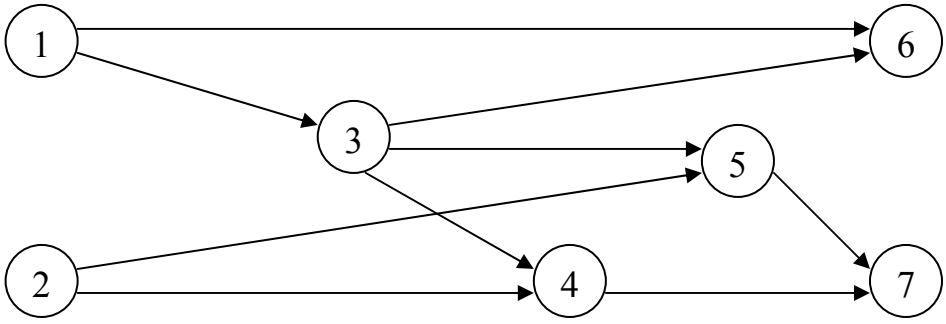


Рисунок 3 – Нейронная сеть

Рассмотрим данный способ кодирования для нейронной сети с двумя входами, двумя выходами и тремя промежуточными нейронами на примере сети, представленной на рис. 3.

Нейроны нумеруются по правилу: сначала входные нейроны, затем скрытые нейроны, в конце нейроны на выходе сети. Тогда матрица связей C будет выглядеть следующим образом:

$$C = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Будем использовать 8 активационных функций и зададим вектор их номеров для всех скрытых нейронов: $F\_activ\_hid = (1,8,2)$, а также для нейронов на выходе сети: $F\_activ\_out = (3,6)$.

В этом случае хромосома имеет следующий вид:

10010 01100 1110*000* 001*111* 01*001 010 101*,

где коды активационных функций выделены курсивом.

Второй этап эволюционного проектирования архитектуры нейронной сети состоит из следующих шагов:

1. Декодирование каждой особи текущей популяции для описания архитектуры нейронной сети.

2. Обучение каждой нейронной сети с архитектурой, полученной на первом шаге, с помощью генетического алгоритма подбора весов связей.

3. Оценивание приспособленности каждой особи (закодированной архитектуры) по достигнутым результатам обучения, т.е. по наименьшей общей среднеквадратичной погрешности обучения.

4. Репродукция особей, согласно выбранному способу селекции.

5. Применение генетических операторов – таких как скрещивание и мутация для получения нового поколения.

Эволюционный алгоритм автоматизированного генерирования нейронной сети

Автоматическое формирование нейронной сети методом генетического программирования

Для применения методов генетического программирования (ГП) при автоматизации процесса проектирования искусственных нейронных сетей (ИНС) необходимо решить две задачи – кодирование нейронной сети в виде дерева (в общем случае дерево может быть n-нарным, но обычно выбирают бинарное) и выбор функции пригодности.

Для кодирования ИНС при помощи дерева необходимо определить терминальное (T) и функциональное (F) множества.

При выборе терминального и функционального множеств необходимо соблюсти следующие условия:

1) замкнутость: любое дерево, составленное из элементов этих множеств, должно представлять собой некоторое решение из поискового пространства;

2) достаточность: набор множеств (терминального и функционального) должен позволять кодировать любое решение из поискового пространства.

Кроме того, выбранный способ кодирования должен позволять:

- представлять сети с межслойными связями;

- представлять сети со связями такими, что не обязательно каждый нейрон предыдущего слоя связан с каждым нейроном последующего слоя;

- представлять на одном слое нейроны с различными активационными функциями.

Эти требования направлены на обеспечение более гибкого поиска структуры ИНС.

Кодирование ИНС при помощи дерева.

Пусть терминальное множество состоит из частей (блоков), из которых по предположению исследователя должна состоять нейронная сеть. Это могут быть нейроны различных видов (в том числе входные), нейроны, уже

объединенные в некоторые блоки, являющиеся частью слоя, и т.п. Функциональное множество состоит из элементов, показывающих, каким образом соотносятся между собой элементы из T. Поясним сказанное на примере. Пусть имеется следующий набор T={slIn12, slIn34, sl2, sl3}, F={+sl, >}. Здесь slIn12 – блок, состоящий из первого и второго входных нейронов; slIn34 – блок, состоящий из третьего и четвертого входных нейронов; sl2 – блок, состоящий из двух нейронов с некоторыми активационными функциями S1 и S2; sl3 – блок, состоящий из трех нейронов с функциями активации S3, S4 и S5. Тогда дерево (рис. 1) соответствует нейронной сети (рис. 2).

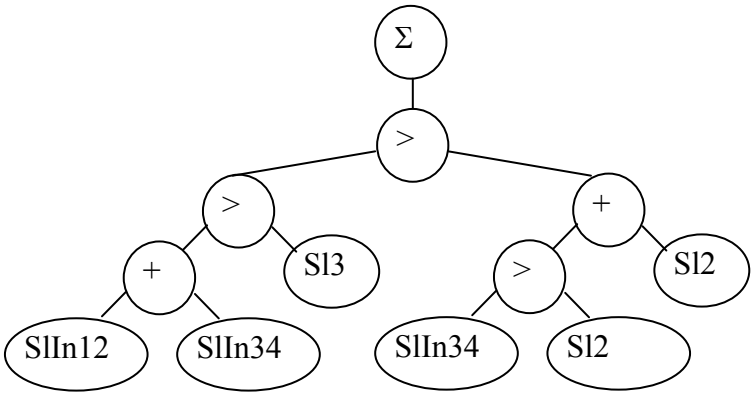


Рисунок 1 – Пример дерева, кодирующего ИНС (рис. 2)

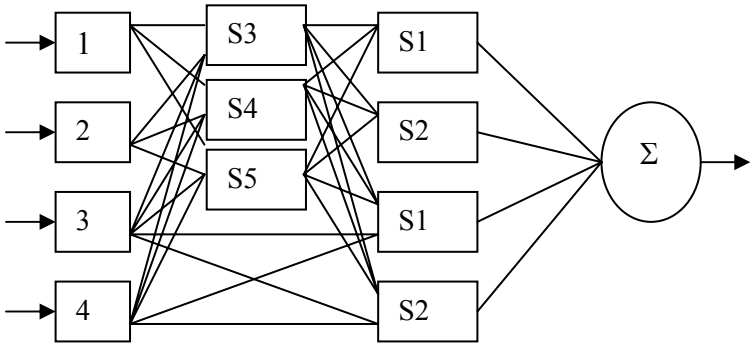


Рисунок 2 – ИНС, соответствующая бинарному дереву (рис. 1)

Однако при таком кодировании существует два возможных варианта, приводящих к недопустимым выражениям. Первый вариант, когда на входной нейрон (или блок, содержащий входной нейрон или нейроны) подается входной сигнал (рис. 3). Второй вариант, когда входной нейрон

(или нейроны), нужно объединить в слой с межслойным нейроном (межслойным блоком) (рис. 4).

В таких случаях можно просто заменить элемент функционального множества таким, который не приведет к недопустимым выражениям – адаптировать дерево (рис. 5, рис. 6).

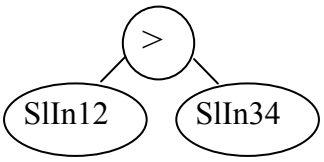


Рисунок 3 – Вариант 1. Недопустимое выражение

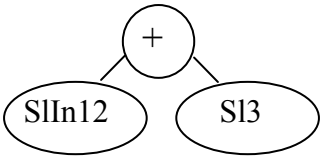


Рисунок 4 – Вариант 2. Недопустимое выражение

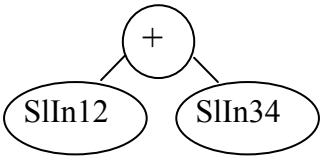


Рисунок 5 – Вариант 1. Адаптированное поддерево

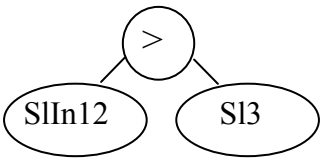


Рисунок 6 – Вариант 2. Адаптированное поддерево

Кроме того, при начальной генерации или при скрещивании может получиться ИНС, не имеющая ни одного блока с входными нейронами. На такую сеть подаются все входы таким образом, что каждый вход подается на каждый нейрон, не имеющий входного сигнала.

В этом случае некоторый набор деревьев с различными элементами после адаптации может свестись к одному дереву, а значит к одной и той же

ИНС. Это отрицательно сказывается на работе алгоритма, но позволяет находить эффективные решения для сложных задач.

Данный способ кодирования ИНС в виде бинарных деревьев совместно с указанными способами адаптации дерева удовлетворяет условиям замкнутости и достаточности. Любой элемент из F может принимать любые элементы из T (с возможностью адаптации) в качестве аргументов. В результате получаем блок ИНС, который в свою очередь может являться элементом из T и быть аргументом для любого элемента из F. Кодирование ИНС в виде бинарных деревьев позволяет получать сети с межслойными связями (в случае, если блок с меньшим количеством слоев объединяется с блоком, имеющим большее количество слоев – оператор +). При этом не обязательно каждый нейрон предыдущего слоя связан с каждым нейроном последующего слоя (в случае если в двух блоках сначала формируются межслойные связи – оператор >, а затем блоки объединяются в слои – оператор +). Предложенный способ кодирования позволяет получать ИНС, в которых на одном слое содержатся нейроны с различными активационными функциями (блоки с различными активационными функциями объединяются в слой – оператор +).

Чтение дерева, представляющего собой закодированную структуру ИНС, осуществляется снизу вверх.

Рассмотрим процесс чтения дерева на примере, взяв за основу дерево, приведенное на рис. 1.

Первый этап. На самом нижнем уровне терминальные листья «SlIn12» и «SlIn34» объединяются функциональным листом «+», а листья «SlIn34» и «Sl2» объединяются функциональным листом «>». Таким образом получаем элементы ИНС (рис. 7.).

Второй этап. Поднимаемся на уровень выше. Результат объединения листьев «SlIn12» и «SlIn34» (рис. 7, а) объединяется с элементом «Sl3» узлом

«>», а результат объединения листов «SlIn34» и «Sl2» (рис. 7, b) с элементом «Sl2» узлом «+», (рис. 8).

Третий этап. На этом этапе объединяем полученные на втором этапе элементы (рис. 8) в сеть (рис. 2).

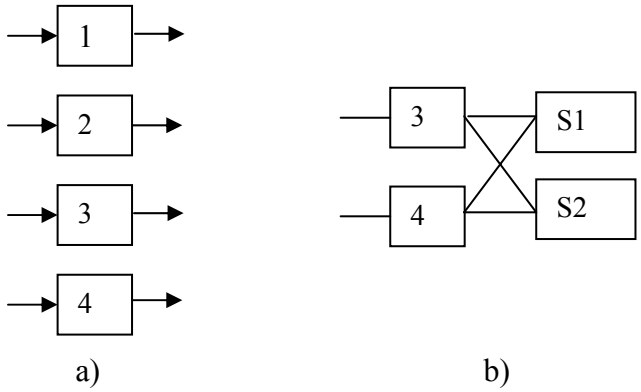


Рисунок 7 – Первый этап чтения дерева. Объединение листьев «SlIn12» и «SlIn34» - a, и листьев «SlIn34» и «Sl2» - b

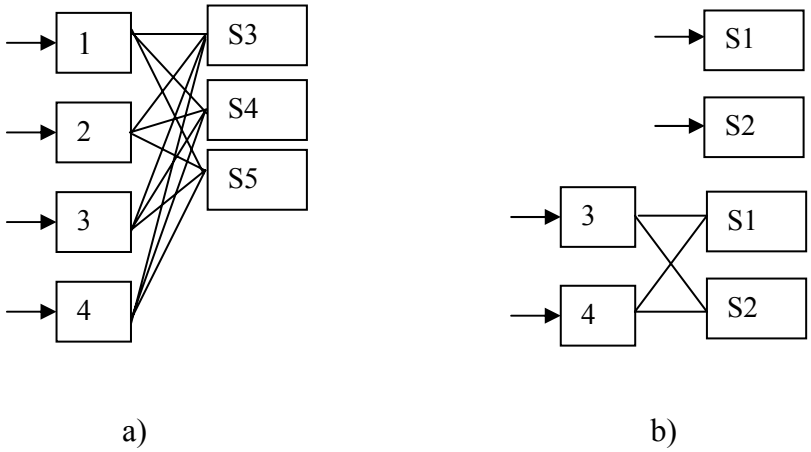


Рисунок 8 – Второй этап чтения дерева

Оптимизация нейросетевой модели

В ходе процедуры ГП могут формироваться нейросетевые модели с произвольной структурой. Оптимизация таких моделей не может выполнятся стандартными алгоритмами, применяющимися для оптимизации ИНС (такими, как рекуррентные формы алгоритма обратного распространения ошибки). В большинстве случаев стандартные алгоритмы применимы к полносвязным ИНС и не могут быть использованы для оптимизации произвольных нейросетевых моделей.

Оптимизация ИНС заключается в выборе оптимальных значений весовых коэффициентов сигналов ИНС и коэффициентов функций активации нейронов. Известно, что одним из наиболее эффективных методов глобальной оптимизации являются генетические алгоритмы. Генетические алгоритмы относятся к методам прямого поиска, т.е. не используют в процессе оптимизации информацию о полезных свойствах функции, а значит подходят для оптимизации нейросетевых моделей с произвольной структурой. Несмотря на такую привлекательность генетических алгоритмов, невозможно дать гарантию того, что полученное решение будет хотя бы локальным экстремумом. Эффективность генетического алгоритма существенно зависит от его настроек.

Для усиления эффективности генетического алгоритма, обычно применяют методы локального спуска. Существуют различные методы гибридизации генетического алгоритма и локального спуска. Например, на очередном этапе, получив популяцию, выполняют локальный спуск из каждого решения. Существует как минимум два варианта схем продолжения оптимизационной процедуры. Можно производить скрещивание потомков, полученных после локального спуска, либо полученных при предыдущем скрещивании (не улучшенных локальным спуском). Кроме того, для экономии временных ресурсов, можно разбить популяцию на кластеры и выполнять локальный спуск из центра каждого кластера. В случае, если группа потомков лежит в зоне притяжения одного экстремума, локальный спуск из этих потомков приведет к одному решению. Выполняя локальный спуск только из центра кластера, можно избежать действий, приводящих к одному решению, что существенно сэкономит время. Однако мы не можем гарантировать, что индивиды, принадлежащие к полученному кластеру, лежат в зоне притяжения одного и того же решения. Выбор той или иной схемы гибридизации зависит от задачи, для которой применяется алгоритм, и

разработчика. Например, для гарантии того, что конечное решение, полученное алгоритмом, будет как минимум локальным экстремумом, достаточно выполнить локальный спуск из решения, полученного генетическим алгоритмом. Такая схема привлекательна экономией вычислительных (а значит и временных) ресурсов. Так как в нашем случае оптимизация нейросетевых моделей производится многократно, то применение такой схемы является наиболее целесообразным.

Алгоритм локальной оптимизации также может быть различным. Так как генетический алгоритм оперирует с бинарными строками, то обычно выбирают локальные алгоритмы для оптимизации псевдобулевой функции. В нашем случае итогом оптимизационной процедуры является набор значений весовых коэффициентов, либо коэффициентов активационных функций нейронов. Известно, что одним из наиболее эффективных алгоритмов оптимизации ИНС является метод сопряженных градиентов. Однако метод сопряженных градиентов требует нахождения частных производных по каждой переменной. Так как структура нейросетевой модели может быть произвольной, то очень сложно разработать общие алгоритмы для нахождения частных производных, получаемых в ходе процедуры ГП, структур. Еще сложнее это сделать, если допустить наличие обратных связей. Более доступным является способ численного вычисления производной. Кроме вычисления частных производных, в методе сопряженных градиентов необходимо организовать процедуру одномерного поиска. Наиболее простым решением может быть применение методов прямого одномерного поиска, таких как поиск методом Фибоначчи или методом золотого сечения. Так как метод золотого сечения является предельным случаем поиска Фибоначчи и при достаточно большом количестве шагов дает результат лишь на 17 % хуже и при этом не требует вычисления чисел из ряда Фибоначчи, то выберем его.

Общая схема оптимизационной процедуры.

Шаг 1. Для полученной структуры ИНС сгенерировать случайным образом весовые коэффициенты и коэффициенты активационных функций нейронов. Обычно генерируют случайные числа в достаточно малом диапазоне (от –1 до 1 для весовых коэффициентов и от 0 до 1 для коэффициентов активационных функций).

Шаг 2. Оптимизируем весовые коэффициенты ИНС методом генетического алгоритма при текущих значениях коэффициентов активационных функций нейронов.

Шаг 3. Улучшаем решение, найденное на шаге 2 (если это не экстремум) методом сопряженных градиентов.

Шаг 4. Оптимизируем коэффициенты активационных функций нейронов методом генетического алгоритма при текущих значениях весовых коэффициентов.

Шаг 5. Улучшаем решение, найденное на шаге 4, методом сопряженных градиентов (если это не экстремум).

Шаг 6. Для улучшения ошибки ИНС можно повторить шаги 2-5.

Общая схема эволюционного алгоритма, автоматически формирующего нейросетевые модели

Шаг 1. Задать начальные настройки алгоритма. Количество индивидов, количество популяции, вид скрещивания, уровень мутации для процедуры генетического программирования и для процедуры генетического алгоритма. Уровень дерева при начальной генерации, максимально допустимый уровень дерева. Количество шагов для процедуры сопряженных градиентов, точность вычисления частной производной, точность одномерного поиска и др. Сгенерировать исходные блоки, из которых будут выращиваться нейронные сети, определить количество повторений в оптимизационной процедуре на шаге 6.

Шаг 2. Сгенерировать начальную популяцию деревьев с учетом установленных настроек на шаге 1 случайным образом. Представить полученные деревья в виде структур ИНС.

Шаг 3. Найти значения весовых коэффициентов и коэффициентов активационных функций нейронов с помощью разработанной оптимизационной процедурой для каждой из структур ИНС для текущей популяции.

Шаг 4. Вычислить пригодность для каждой нейросетевой модели.

Шаг 5. Проверить условие останова. Если условие выполняется перейти на шаг 7, иначе – на шаг 6.

Шаг 6. Сгенерировать новую популяцию деревьев в соответствии с выбранными настройками на шаге 1. Представить полученные деревья в виде структур ИНС. Перейти на шаг 3.

Шаг 7. Выдать в качестве решения ИНС с наилучшей пригодностью. Завершить алгоритм.

Функция пригодности

Функции пригодности могут быть различными. Например, функция пригодности может быть равна обратной величине ошибке ИНС после некоторого, наперед заданного, количества итераций настройки весовых коэффициентов:

$$fitness = \frac{1}{1 + E + k * n}$$

$$E = \left(\frac{\sqrt{\sum_{i=1}^{N} (X_i - X_i^*)^2}}{N} \right)$$

где n – количество уровней в дереве; k – некоторый коэффициент, задаваемый пользователем; X_i – i-й прогноз; X_i^* – i-е реальное значение; N – количество примеров в выборке.

Либо это может быть число итераций настройки весовых коэффициентов, необходимых для получения некоторого заданного уровня ошибки ИНС:

$$fitnes = \frac{1}{1 + N + k \cdot n},$$

где N – количество итераций обучения ИНС; n – количество уровней в дереве; k – некоторый коэффициент, задаваемый пользователем.

Могут быть и другие варианты оценки пригодности в зависимости от задачи, решаемой ИНС.

Применение данного подхода при решении практических задач показало его работоспособность.

Эволюционный алгоритм автоматизированного проектирования нечеткого контроллера

Нечеткие системы управления

Нечеткие системы управления (нечеткие контроллеры, НСУ) являются наиболее важным приложением теории нечетких множеств. Функционирование нечетких логических контроллеров отличается от работы обычных контроллеров. Это отличие состоит в том, что обычные контроллеры основаны на аналитическом выражении, описывающем объект управления, в то время как нечеткие контроллеры используют знания экспертов. Эти знания могут быть выражены естественным образом с помощью так называемых лингвистических переменных.

Лингвистические переменные могут рассматриваться либо как переменные, значения которых являются нечеткими числами, либо как переменные, значения которых определяются в лингвистических термах.

Лингвистические переменные характеризуются пятеркой (x, $T(x)$, U, G, M), в которой x – имя переменной, $T(x)$ – множество термов x, т.е. множество имен лингвистических величин x, каждое значение которой есть нечеткое число, определенное на U, G – синтаксические правила для выработки имен величин x, M – семантические правила для связывания каждой величины с ее смыслом.

Предназначение НСУ состоит в том, чтобы следить за значениями переменных состояния управляемой системы и получать величины переменных управления путем определенных связей, которые представляют собой базу правил (БП) системы.

Нечеткие логические системы управления обычно состоят из четырех основных частей:

- интерфейс фаззификации (введение нечеткости на четких входных данных);
- база нечетких правил;
- машина нечеткого вывода;
- интерфейс дефаззификации (исключение нечеткости для обеспечения четкого выхода).

Интерфейс фаззификации – оператор фаззификации переводит четкие данные в нечеткие множества.

База нечетких правил – содержит информацию о связях нечетких входных и выходных значений.

Машина нечеткого вывода – механизм сопоставления по БП нечетких значений входных параметров нечетким значениям выходных.

Интерфейс дефаззификации – оператор дефаззификации переводит нечеткие множества в четкие значения.

В общем виде принцип работу НСУ представлен на рис. 1.

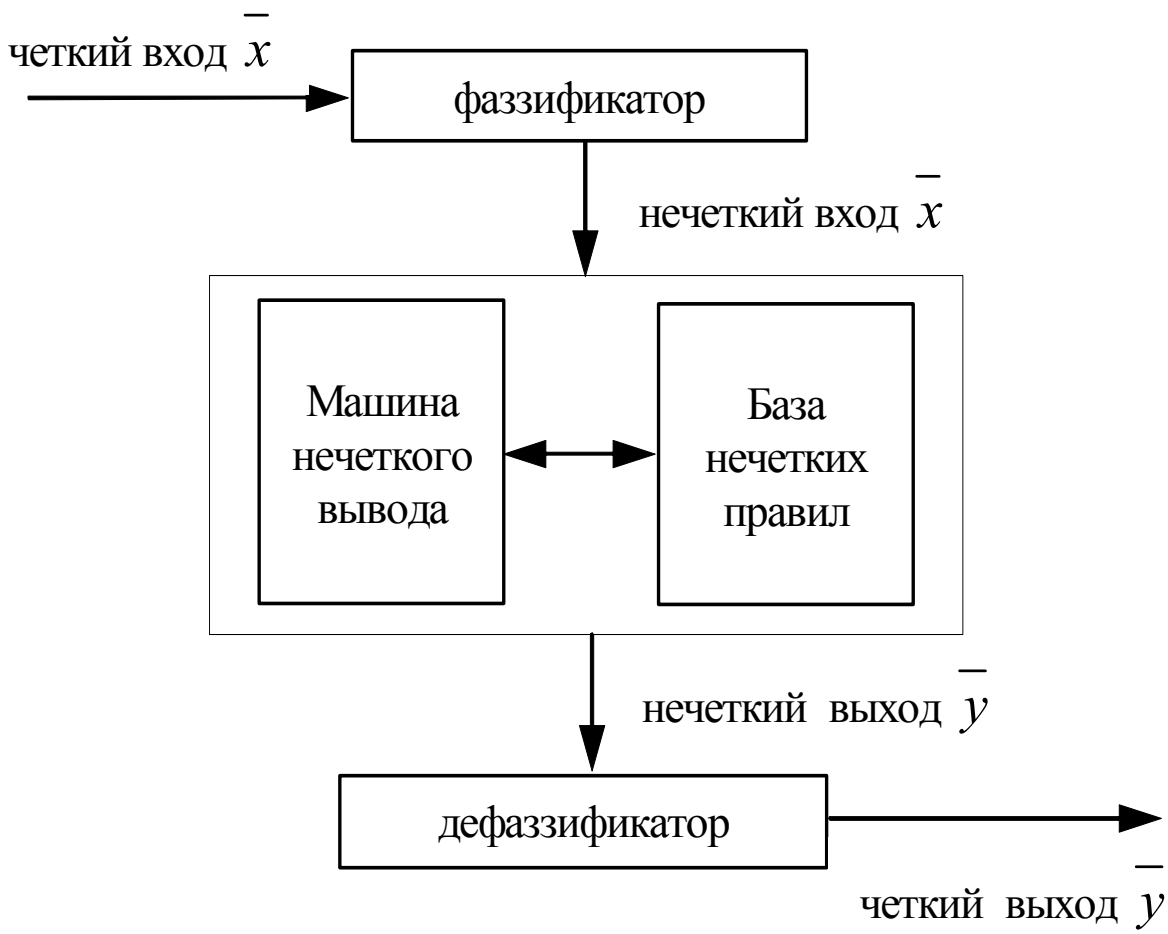


Рисунок 1 – Структурная схема нечеткого логического контроллера

Методы и алгоритмы автоматического формирования нечетких систем управления

Для упрощения работы с БП при оптимизации предлагается следующий способ преобразования БП из вида *if* $\boldsymbol{x} = \boldsymbol{A}$ *then* $\boldsymbol{y} = \boldsymbol{B}$ к целочисленному виду. Каждому терму лингвистических переменных присваивается номер, соответствующий порядку расположения терма на

числовой оси. Например, в случае описания переменной 5-ю термами, им будут присвоены следующие номера – «Отрицательное большое» = 0, «Отрицательное малое» = 1, «Ноль» = 2, «Положительное малое» = 3 и «Положительное большое» = 4 (рис. 2).

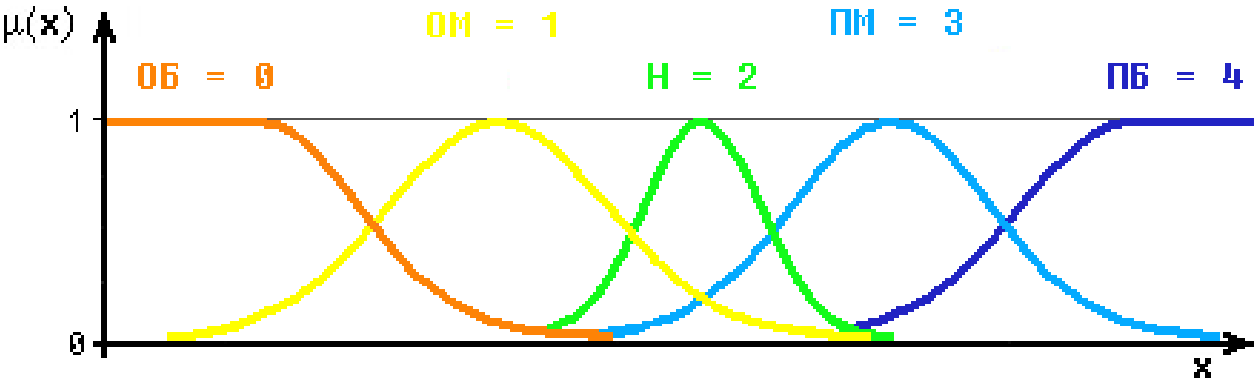


Рисунок 2 – Порядок нумерации термов лингвистических переменных

Тогда, в случае *n* входов и одного выхода, БП будет представлять собой *n*-мерный массив. Порядковые номера термов входных лингвистических переменных будут служить индексами данного массива, а элементами массива будут являться номера термов выходной переменной.

Настройка семантики лингвистических переменных алгоритмом, основанным на понятии точек оперирования

Алгоритм автоматической настройки семантики лингвистических переменных состоит из трех основных этапов:

1. Построение кривой интенсивности состояний переменной и выделение точек оперирования.
2. Аппроксимация кривой интенсивности состояний.
3. Построение термов лингвистических переменных.

Точками оперирования (ТО) называются точки, соответствующие пиковым значениям на кривой интенсивности состояний объекта. Под кривой интенсивности состояний подразумевается график частот нахождения переменной в той или иной области множества допустимых значений. Поскольку в одних состояниях объект может находиться чаще, чем в других, то график интенсивности состояний будет не равномерным, а скорее похожим на кривую, изображенную на рис. 3.

Восстановление кривой интенсивности состояний производится с использованием любого из доступных методов, например, непараметрической регрессией, по имеющейся выборке наблюдений за поведением объекта в ходе экспериментов с ним. Для выделения ТО кривая интенсивностей сглаживается, что приводит к исчезновению мелких малозначимых пиков и более сильному выражению крупных пиков. Сглаживание кривой интенсивностей состояний производится с помощью метода непараметрической регрессии. Затем, на сглаженной кривой определяются точки экстремумов, соответствующие максимумам кривой интенсивности состояний. Данные точки являются ТО (рис. 3).

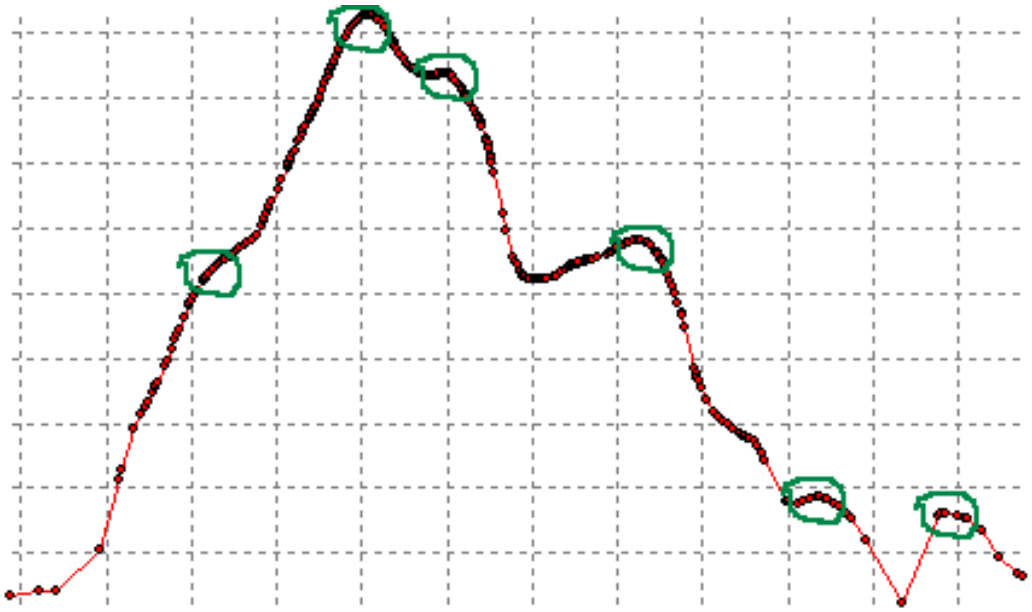


Рисунок 3 – Точки оперирования на кривой интенсивности состояний объекта

На втором этапе алгоритма происходит аппроксимация кривой интенсивности состояний переменной функциями Гаусса (рис. 4).

Значения ТО выступают в качестве математических ожиданий (МО) функций Гаусса, а их дисперсии подбираются в процессе оптимизации. Поиск оптимальных дисперсий для набора функций Гаусса осуществляется с помощью алгоритмов глобальной оптимизации, например, с использованием генетических алгоритмов.

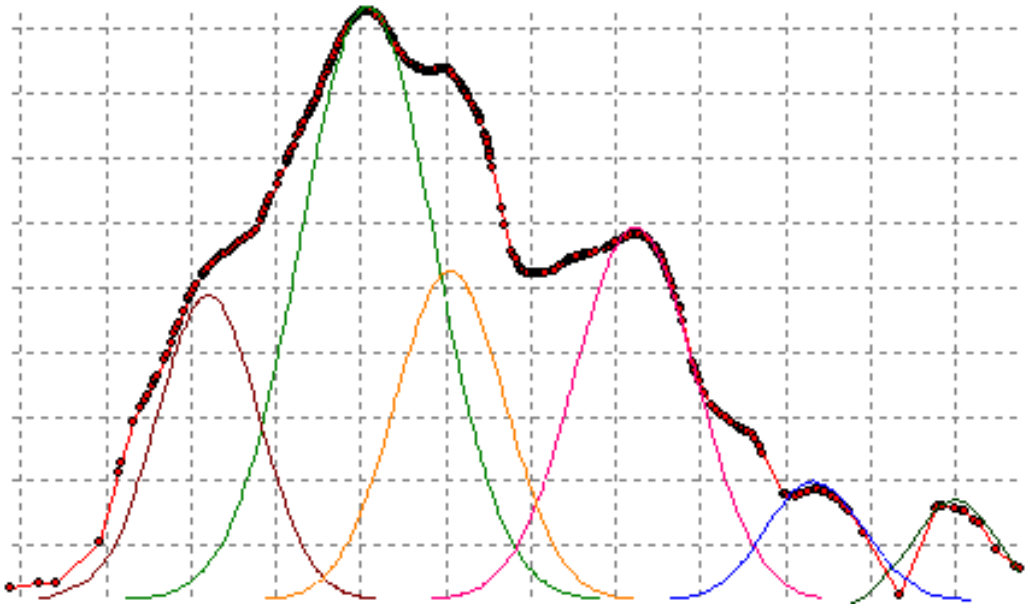


Рисунок 4 – Аппроксимация кривой интенсивности состояний функциями Гаусса

Для определения необходимого для аппроксимации числа функций Гаусса используется метод перебора. Число используемых функций варьируется от 1 до 10. Наилучшим результатом считается тот набор аппроксимирующих функций, который содержит наименьшее их количество и обеспечивает уровень ошибки аппроксимации не выше заранее установленного. Итогом работы второго этапа настройки являются найденное число значимых ТО и МО и дисперсия для каждой функции Гаусса.

Далее, на третьем этапе происходит формирование термов лингвистических переменных по параметрам, полученным на предыдущих этапах алгоритма (рис. 5).

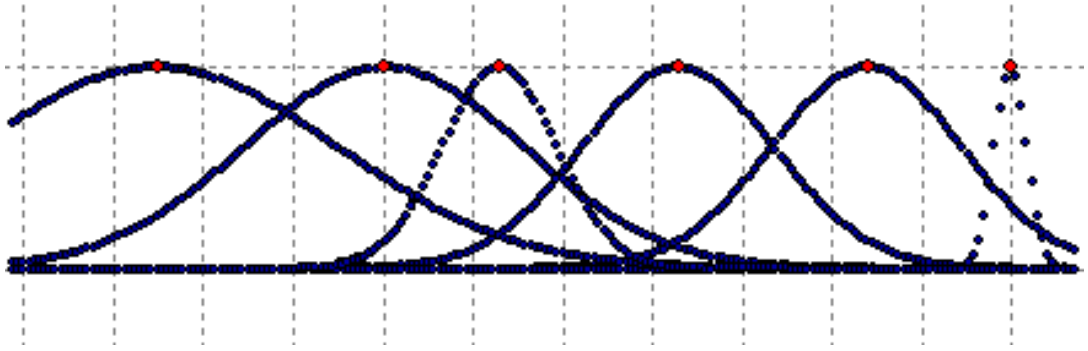


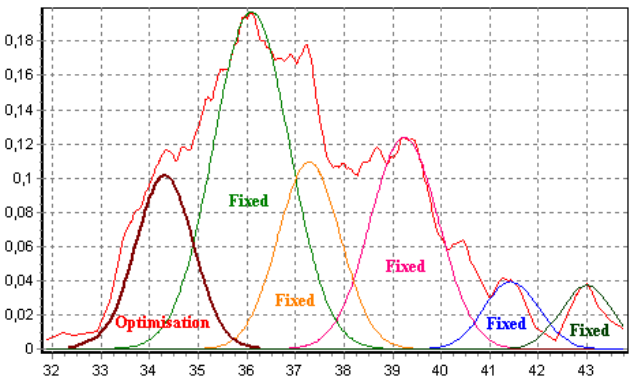
Рисунок 5 – Термы лингвистической переменной

В качестве функций принадлежности выступают функции Гаусса, отмасштабированные до единицы в точке своего МО. При необходимости, к сформированным термам переменной добавляются еще один или два терма. Добавляемые термы покрывают интервалы от минус бесконечности до левой границы множества допустимых значений переменной и от правой границы до плюс бесконечности.

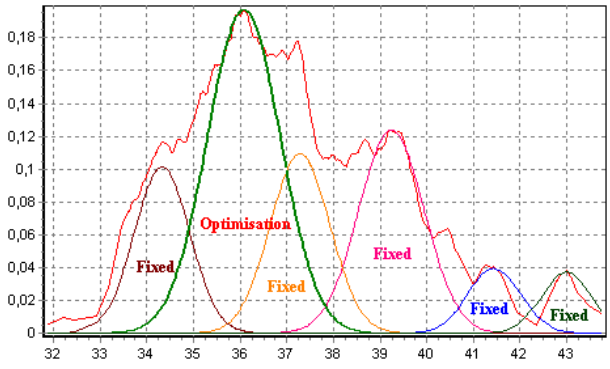
Анализ данных по экспериментам и настройка термов по каждой лингвистической переменной происходит вне зависимости от параметров, полученных для других переменных.

Настройка семантики лингвистических переменных алгоритмом пошаговой оптимизации

Данный алгоритм является модификацией предыдущего и позволяет повысить скорость и качество оптимизационного процесса. Суть данного алгоритма заключается в том, что на каждом из этапов оптимизации производится подбор параметров (математического ожидания и дисперсии) не для всех используемых аппроксимирующих функций, а только для одной их них (рис. 6 а, б).



а) настройка первого терма при фиксированных параметрах остальных термов



б) настройка параметров второго терма при фиксированных параметрах остальных терм

Рисунок 6 – Пример процесса настройки семантики пошаговым алгоритмом

Значения параметров остальных же функций берутся наилучшие из оптимизации на предыдущих этапах алгоритма. При завершении подбора параметров для текущего терма, его новые значения фиксируются, и производится оптимизация параметров следующего за ним терма и т.д.

Схема работы алгоритма представлена ниже:

1. Выбрать k=1, где k – число функций Гаусса, используемых для аппроксимации кривой интенсивностей состояний объекта.
2. Установить s=k, s – номер текущего терма (терма, параметры которого в данный момент оптимизируются).
3. Фиксировать параметры всех функций Гаусса, используемых для аппроксимации, кроме s-ой.
4. Оптимизировать параметры s-ой функции Гаусса (МО и дисперсию), минимизирую ошибку среднеквадратичного отклонения построенной модели от исходных данных.
5. Запомнить наилучшие найденные параметры s-ой функции.
6. Если s > 1, то s = s-1 и идти на шаг 3.
7. Запомнить число k, текущие параметры функций Гаусса и значение ошибки аппроксимации Err_k.
8. Если k < N (где N – максимальное допустимое число функций Гаусса, используемых для аппроксимации), то k = k+1 и идти на шаг 2.
9. Выдать число k и параметры функций Гаусса, соответствующих наименьшему значению Err_k. Остановиться.

Таким образом, производится подбор параметров для всех функций Гаусса, используемых для аппроксимации кривой интенсивности состояний. При необходимости, цикл оптимизационных процедур (шаги 2-6) повторяется несколько раз.

По завершении настройки семантики лингвистических переменных происходит переход ко второму этапу разработки нечеткого контроллера – формирование базы нечетких правил (БП).

Модифицированный метод формирования базы нечетких правил

Формирование БП осуществляется по совокупности наблюдений за объектом и при настроенных термах лингвистических переменных. Алгоритм формирования БП состоит из двух основных этапов:

1. Фаззификация данных наблюдений за объектом.
2. Формирование БП по нечетким (фаззифицированным) значениям.

Фаззификация данных для каждой лингвистической переменной производится по соответствующим ей термам. При фаззификации входных переменных используется обычный механизм перевода четких значений в

нечеткие, т.е., согласно правилам фаззификации, каждому четкому значению входной переменной x_i ставится в соответствие пара чисел (s_i, w_i), где s_i – имя (номер) терма *i*-ой переменной, которому принадлежит значение x_i, w_i – степень принадлежности значения x_i к терму s_i. При фаззификации выходных переменных применяется обратный модифицированный механизм дефаззификации по первому максимуму. Аналогично как и в случае с входными переменными, каждому значению y_i будет соответствовать своя пара (s_i, w_i). В итоге мы преобразуем все данные экспериментов в нечеткие числа, которые будут представляться в виде выражений (имя терма, степень принадлежности).

Поскольку БП представляет собой набор выражений ЕСЛИ … ТО …, то формирование БП будет представлять собой процесс сопоставления нечетких величин входных переменных (имен термов входных переменных) нечетким величинам выходных переменных (именам термов выходных переменных). Используемый метод настройки семантики лингвистических переменных позволяет практически полностью сформировать БП, однако не исключает присутствие «пробелов» в создаваемой БП. Это обстоятельство во многом будет определяться числом проведенных экспериментов над объектом и их разнообразием.

Генетический подход к формированию БП для нечеткой системы управления

Каждый индивид в ГА представляет собой решение – БП, содержащую все правила, необходимые для управления системой, и принадлежащую множеству всех возможных БП.

В ГА БП кодируется бинарной строкой фиксированной длины. Поскольку БП представляет собой *n*-мерный целочисленный массив, то ее кодировка не составляет особой сложности (рис. 3). Если учесть тот факт, что БП симметрична относительно точки равновесия (точка в БП, где все входные параметры принимают нулевые значения), то длину хромосомы можно уменьшить в два раза.

Тем самым уменьшается время, затрачиваемое на поиск лучшего решения, и одновременно увеличиваем “качество” найденных решений.

После кодирования БП в хромосому запускается механизм обычного ГА.

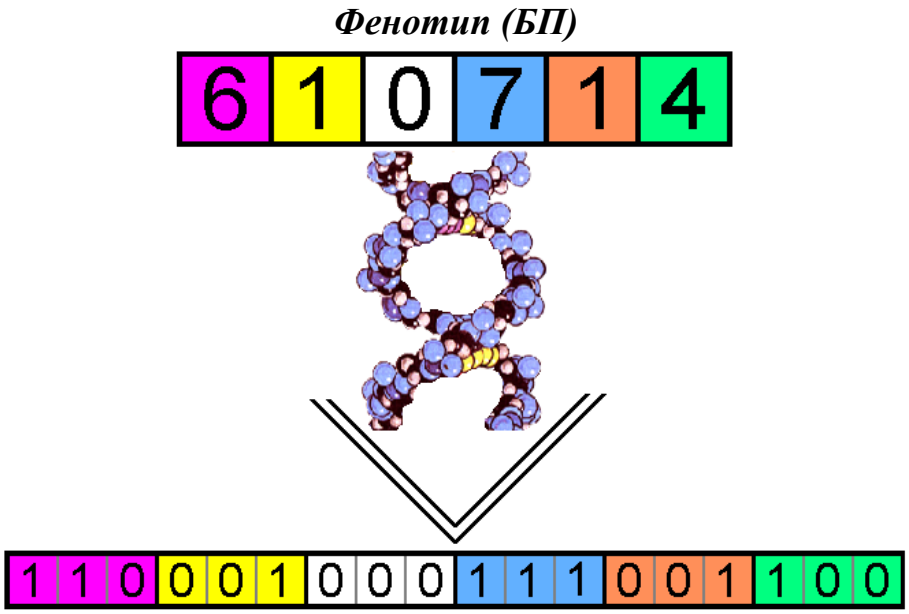


Рисунок 3 – Кодирование решений в ГА

На первом этапе работы ГА происходит случайная генерация решений - БП (начальной популяции). Затем, эта популяция проходит процесс эволюции в форме естественного отбора. В каждом поколении относительно хорошие решения дают потомков, которые заменяют относительно плохие решения, исчезающие из популяции.

Функция пригодности играет роль внешней среды, которая определяет хорошие и плохие решения, т.о. функция пригодности назначает каждому индивиду число, пропорциональное мере адаптации индивида к “внешней среде”.

Функция пригодности, значение которой необходимо минимизировать в процессе оптимизации, имеет следующий вид:

$$Fit = \frac{1}{n}\sqrt{\frac{\sum_{i=1}^{n}(y_i - y_i^{\text{mod}} - 2\cdot y_{\min})^2}{(y_{\max} - y_{\min})}} \rightarrow \underset{\text{ÁÏ, Ñâîéñòâà}}{min}$$

где y_i – значение выходной переменной на i-й набор входных параметров из задачника;

y_i^{mod} – значение выходной переменной модели на i-й набор входных параметров из задачника;

y_{min} – минимальное значение выходной переменной в задачнике;

y_{max} – максимальное значение выходной переменной в задачнике;

n – количество примеров в задачнике;

Fit – оценка качества текущей модели (в процентах).

Согласно данному критерию, перед вычислением ошибки аппроксимации производится нормализация исходных данных (данные преобразуются таким образом, чтобы значения выходной переменной изменялись от нуля и до единицы), и лишь затем вычисляется среднеквадратичное отклонение поведения модели от исходного объекта. Минимизация критерия осуществляется за счет наилучшей настройки БП и семантики лингвистических переменных.

Эволюционный алгоритм автоматизированного проектирования экспертной системы на нечеткой логике

Формирование систем на нечеткой логике.

При проектировании систем на нечеткой логике исследователю необходимо решить две основные задачи.

Первая задача состоит в создании лингвистических переменных. Существует несколько методик создания лингвистических переменных. Например, групповые прямые методы. Создается группа экспертов, которой предъявляется элемент из множества, каждый эксперт либо причисляет элемент к данному множеству, либо нет. Принадлежность вычисляется как отношение утвердительных ответов, к общему числу ответов. Другим методом является использование относительных частот по данным эксперимента в качестве значений принадлежности. Практикуется также использование типовых форм кривых для задания функций принадлежности.

Вторая задача состоит в проектировании базы правил (БП). В процессе создания базы правил извлекаются знания из эксперта и сохраняются в виде, пригодном для компьютерной обработки на этапе нечеткого вывода. Извлечение знаний – это процедура взаимодействия эксперта с источником знаний, в результате которой становятся явными процесс рассуждений специалистов при принятии решения и структура их представлений о предметной области.

Методы извлечения знаний можно разделить на две основные группы: коммуникативные и текстологические. Коммуникативные методы извлечения знаний охватывают методы и процедуры контактов инженера по знаниям непосредственно с экспертом, а текстологические включают методы извлечения знаний из документов (методик, пособий, руководств) и специальной литературы (статей, монографий, учебников). В свою очередь коммуникативные методы можно также разделить на две группы: активные и пассивные. Пассивные методы подразумевают, что ведущая роль в процедуре извлечения как бы передается эксперту, а инженер по знаниям только протоколирует рассуждения эксперта во время его реальной работы по принятию решений или записывает то, что эксперт считает нужным самостоятельно рассказать в форме лекции. В активных методах, напротив, инициатива полностью в руках инженера по знаниям, который активно

контактирует с экспертом различными способами – в играх, диалогах, беседах за круглым столом и т.д.

Все методы извлечения знаний направлены на обучение инженера по знаниям. Очевидно, что этот процесс может затянуться на несколько лет.

В настоящее время большинство разработчиков экспертных систем (ЭС) отмечает, что процесс извлечения знаний остается самым «узким» местом при построении ЭС. При этом им приходится практически самостоятельно разрабатывать методы извлечения, сталкиваясь со следующими трудностями:

- организационные неувязки;
- неудачный метод извлечения, не совпадающий со структурой знаний в данной области;
- неадекватная модель для представления знаний;
- терминологический разнобой;
- отсутствие целостной системы знаний в результате извлечения только «фрагментов».

Процесс извлечения знаний – это длительная и трудоемкая процедура, в которой инженеру по знаниям, вооруженному специальными знаниями по когнитивной психологии, системному анализу, математической логике и пр., необходимо воссоздать модель предметной области, которой пользуются эксперты для принятия решения.

Создание автоматических методов проектирования систем на нечеткой логике позволят сократить время и затраты на разработку.

Формирование лингвистических переменных

Настройка семантики лингвистических переменных генетическим алгоритмом

Оптимизационному алгоритму необходимо найти массив точек, которые представляют собой вершины термов лингвистических переменных.

Основную сложность при оптимизации термов лингвистических переменных составляет кодирование терм в бинарную строку. Кодирование – это представление в виде бинарной строки ключевых точек термов, по

которым в последствии восстанавливаются термы. Однако следующие проблемы осложняют процесс оптимизации:

1. Каждая лингвистическая переменная имеет свой диапазон изменения, соответственно, разные переменные изменяются в разных диапазонах, а для использования оптимизационного алгоритма необходимо, чтобы все переменные имели одинаковый диапазон варьирования;

2. В каждой лингвистической переменой должен соблюдаться строгий порядок расположения ее термов (к примеру, … «отрицательное среднее», «отрицательное малое», «ноль», …), который в ходе оптимизации не должен нарушаться;

3. Необходимо, чтобы термы покрывали весь интервал изменения переменной, т.е. между термами нежелательно существование разрывов.

Данные проблемы можно обойти, если наложить на решение множество ограничений, но тогда вместо задачи безусловной оптимизации пришлось бы решать задачу условной оптимизации, что гораздо сложнее. Следовательно, необходимо разработать такой способ кодирования и декодирования термов, который позволил бы решить обозначенные выше проблемы.

Первая проблема решается достаточно просто: для каждой лингвистической переменной составляется своя масштабная функция, в результате применения которой диапазоны изменения переменных становятся одинаковыми. Данная функция имеет вид:

$$f_{\text{М}}(x) = k \cdot (x\text{-}l_1) + l_{M1},$$

где x – исходная координата терма по оси абсцисс;

$k = d/D$ – коэффициент масштаба;

$d = (l_2 – l_1)$ – ширина интервала изменения переменной до масштабирования;

$D = (l_{\text{M2}} – l_{\text{M1}})$ – ширина интервала изменения переменной после масштабирования;

l_1 и l_2 – левая и правая границы изменения переменной до масштабирования;

l_{M1} и l_{M2} – левая и правая границы изменения переменной после масштабирования;

$f_M(x)$ – промасштабированная абсцисса вершины.

При декодировании полученного решения используется обратная к масштабной функция:

$$f_M^{-1}(x_M) = k_M \cdot (x_M - l_{M1}) + l_1,$$

где $k_M = 1/k = D/d$ – коэффициент обратного масштаба и $x_M = f_M(x)$.

Решение второй проблемы также, на первый взгляд, сравнительно просто. Необходимо лишь применить сортировку к массиву точек, представляющих собой вершины термов, найденных оптимизационным алгоритмом, и затем присваивать отсортированные значения вершинам термов. Порядок следования вершин (нумерация) устанавливается заранее (рис. 1).

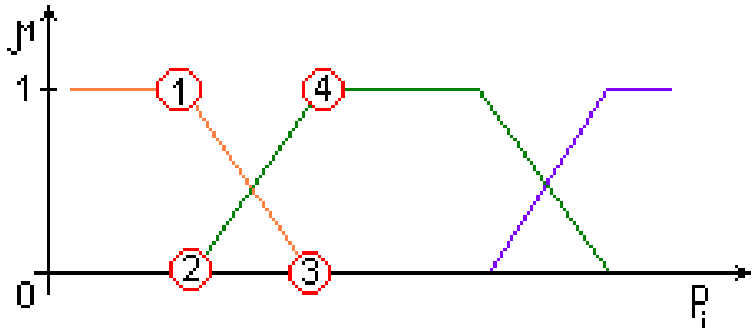


Рисунок – 1 Порядок кодирования точек при настройке семантики

Возникает вопрос: какая из точек (1) и (2), (3) и (4) должна находиться левее, т.е. (1)>(2) или (2)>(1), (3)>(4) или (4)>(3)? Конечно, если предварительно определить очередность их расположения, например, (1)>(2), (3)>(4), то можно будет избежать данной проблемы. Однако, убрав из рассмотрения остальные варианты расположения точек, мы искусственно сократим поисковое пространство, что не допустимо при оптимизации.

Для решения данной проблемы предлагается следующий подход. Для каждой пары таких точек ((1) и (2), (3) и (4), …; рис. 1), назовем их спорными, определяется одна приоритетная точка, к примеру, первая точка пары. Затем найденный массив точек термов сортируется в порядке возрастания их значений. Каждой паре спорных точек в сортированном массиве соответствует своя двойка значений. Первой спорной паре первого терма соответствуют первый и второй элементы отсортированного массива, второй паре второго терма – (3) и (4) элементы массива и т.д. Когда

возникает вопрос о том, какая из точек будет лежать левее, из двух значений массива приоритетной точке приравнивается то число, которое в исходной строке (не сортированной) встречается раньше. Таким образом, отпадает какая-либо необходимость ввода ограничений на решение или искусственного сокращения пространства поиска.

Рассмотрим этот алгоритм на примере. Предположим, что некоторая лингвистическая переменная описывается тремя термами и в ходе оптимизации было получено решение в виде массива {1,2,0,7,5,4,8,6}. Для каждой пары спорных точек (1 и 2, 3 и 4, 5 и 6, …; рис. 1) назначаем приоритетной первую точку, т.е. для 1 и 2 приоритетной является точка 1, для 3 и 4 – это 3 и т.д. После сортировки массив точек примет вид {0,1,2,4,5,6,7,8}. Далее, решается вопрос о расположении точек. Так, точка (2) терма будет лежать левее точки (1), поскольку элемент приоритетной точки {1} в исходном массиве находится левее точки {0}, т.е. точка (2) = {0} терма будет лежать левее точки (1) = {1}.

Для предупреждения разрывов между термами порядок расположения таких точек, как (2) и (3) (рис. 1), оговаривается заранее и не претерпевает изменений в процессе оптимизации. В данном случае точка (2) должна лежать левее, либо совпадать с точкой (3), в противном случае возникнет разрыв функции, что не допустимо.

Оптимизация лингвистических переменных методом сопряженных градиентов

После того, как получено решение генетическим алгоритмом, его можно попытаться улучшить локальным оптимизационным алгоритмом. Например, методом сопряженных градиентов. Кроме того, при оптимизации локальным алгоритмом не возникает тех проблем, которые присутствуют при оптимизации генетическим алгоритмом. Если исходное решение было допустимым, то и улучшенное также будет допустимым.

Общая схема оптимизационной процедуры

Шаг 1.Установить начальные настройки:

- лингвистические переменные: выбрать количество и форму термов для каждой лингвистической переменной;

- генетический алгоритм: установить количество индивидов и популяций, выбрать селекцию, рекомбинацию, уровень мутации и т.д.

- метод сопряженных градиентов: точность вычисления производной, количество итераций, интервал неопределенности для алгоритма Фибоначчи.

Шаг. 2. Для текущей БП получить решение методом генетического алгоритма.

Шаг. 3. Улучшить найденное решение на шаге 2 методом сопряженного градиента.

Формирование базы правил методом генетического программирования

БП ставит в соответствие набору значений входных параметров значение выходных параметров некоторой системы. Например, пусть существует некоторая система, описывающаяся двумя входными параметрами и одним выходным: X={x1, x2, x3}, Y={y1, y2, y3} – вход, U={u1, u2, u3} – выход. Тогда БП может иметь такой вид:

if (x1&y1) **then** u1
else if (x1&y2) **then** u1
else if (x1&y3) **then** u1
else if (x2&y1) **then** u2
else if (x2&y2) **then** u1
else if (x2&y3) **then** u3
else if (x3&y1) **then** u2
else if (x3&y2) **then** u1
else if (x3&y3) **then** u3.

Данная БП является «полной», т.е. перечислены все возможные комбинации входных параметров, и им в соответствие ставится значение выходного параметра. Такие БП обычно очень объемные и, по сути, представляют собой модель черного ящика. На практике чаще встречаются «упрощенные» БП. Например, следующая БП равна по смыслу исходной, однако, содержит меньше правил, и более понятна человеку (эксперту):

if (x1) **then** u1
else if (y2) **then** u1
else if (y1) **then** u2
else if (y3) **then** u3

Правила, из которых состоит упрощенная БП, назовем обобщенными, т.к. они в себе содержат несколько правил из полной базы правил. И чем больше в обобщенном правиле содержится правил, тем ценнее оно с точки зрения практического использования.

Генетическое программирование (ГП) производит не только численную адаптацию (как генетический алгоритм), но и структурную, и позволяет создать «упрощенные» БП.

Для решения данной задачи методом ГП необходимо решить следующие задачи: кодирование БП и выбор функции пригодности.

Кодирование БП

Для кодирования БП необходимо выбрать терминальное и функциональное множества, удовлетворяющие условиям замкнутости и достаточности. Выбор терминального множества в данном случае определяется однозначно: значение выходных параметров и их комбинации. В нашем случае терминальное множество будет T={u1, u2, u3}. Выбор функционального множества, напротив, может решаться различными способами. Функциональный элемент делит, по какому-либо правилу, вектор входных параметров на два вектора (если для кодирования используются бинарные деревья) и более. Для кодирования вышеуказанной («упрощенной») БП достаточно следующего функционального множества: F={%X, %Y}. Оператор %X делит вектор значений параметра X пополам (если количество элементов четное) или с перевесом в один элемент (если количество элементов нечетное). Аналогично действует оператор %Y.

Чтение дерева

В отличие от дерева, кодирующего нейронную сеть (НС), дерево, представляющее БП, читается с корня. На корень дерева подается входной вектор (рис. 2). В зависимости от функционального узла вектор делится на подвекторы. В итоге каждому терминальному узлу соответствует некоторый набор значений входных параметров (рис. 2).

Функция пригодности выражает количественно, насколько одно решение предпочтительнее другого. Выбор конкретной функции пригодности зависит от задачи.

Решив вышеперечисленные вопросы, переходим непосредственно к процедуре ГП.

Условие замкнутости

Такое представление решения в виде бинарного дерева существенно отличается от представления решения, предложенного для нейросетевых моделей, и того, что используют для задач символьной регрессии. Это

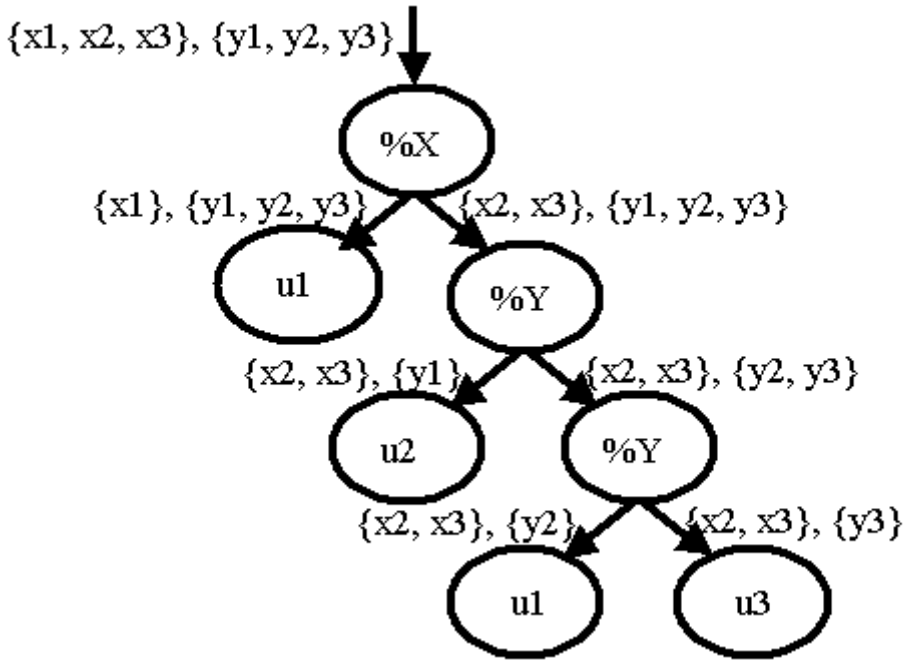


Рисунок – 2. Пример дерева, представляющего упрощенную БП

связано с тем, что чтение дерева в данном представлении происходит сверху, а чтение деревьев, представляющих собой решение для задач символьной регрессии или формирования структуры НС, происходит снизу. И в отличие от последних, аргументом функционального элемента для задачи формирования базы правил будет не поддерево, корнем которого он является, а, наоборот, вектор, пришедший из корня, стоящего выше. А это значит, что для любой структуры дерева и любого функционального элемента аргумент будет всегда корректен, за исключением одного единственного случая, когда вектор, пришедший для данного функционального множества, состоит из единственного значения и его не возможно поделить на две части (рис. 3). В этом случае, чтение дерева

продолжается всегда по одной ветке (например, левой), а правое поддерево будет игнорироваться. Разумеется, это создаст некоторые множества постоянства, однако позволит применить мощный механизм генетического программирования для задачи формирования базы обобщенных правил.

Таким образом, полученные в процессе эволюции деревья будут всегда представлять некоторую БП.

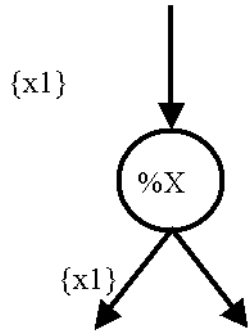


Рисунок – 3. Решение проблемы с недопустимым аргументом

Условие достаточности

Наборов терминального и функционального множеств, составленных из описанных выше элементов, достаточно для решения любой задачи о формировании БП. Это объясняется тем, что предельный случай в таком представлении БП – полная БП, которая способна решить любую задачу, но не эффективна в использовании.

Общая схема эволюционного алгоритма, автоматически формирующего нечеткие экспертные системы

Шаг 1. Установить начальные настройки для лингвистических переменных, генетического алгоритма, алгоритма сопряженных градиентов, алгоритма генетического программирования.

Шаг. 2. На основании априорной информации сгенерировать стартовые лингвистические переменные (например, таким образом, чтобы термы равномерно покрывали интервал изменения переменной).

Шаг. 3. Для имеющихся лингвистических переменных получить БП методом генетического программирования.

Шаг. 4. Оптимизировать имеющиеся лингвистические переменные для полученной на шаге 3 БП.

Шаг. 5. Повторять шаги 3-4 до тех пор, пока ошибка нечеткой системы в целом будет уменьшаться значительно.

Шаг. 6. Если точность полученной на шаге 5 нечеткой экспертной системы не удовлетворительна, увеличить значения настроек (количество индивидов, количество популяций, количество итераций оптимизационного алгоритма и т.п.) и перейти на шаг 2, иначе – принять ее в качестве решения и завершить процедуру.

Примеры решения практических задач эволюционными алгоритмами

Выбор эффективной конфигурации многопроцессорных систем обработки информации и управления генетическими алгоритмами

Один из путей повышения производительности систем обработки информации состоит в переходе от однопроцессорных к многопроцессорным системам, в распараллеливании процессов вычислений и обработки информации, как на уровне процессоров многопроцессорных систем, так и на уровне выполнения элементарных операций внутри процессоров. На базе многопроцессорных систем обработки информации и управления (МСОИУ) реализуются системы управления для многих отраслей: космической отрасли, авиации, для систем противовоздушной и противоракетной обороны и многих других. Однако производство МСОИУ затруднено высокой стоимостью работ на всех его стадиях. В результате общая стоимость системы часто делает ее недоступным инструментом. Использование современных оптимизационных методов и алгоритмов на этапе разработки МСОИУ позволило бы снизить затраты на ее производство и при этом система соответствовала бы предъявляемым к ней требованиям.

В работе разрабатывался алгоритм позволяющий по заданным требованиям создавать проекты структур МСОИУ оптимальные с точки зрения производительности и стоимости таких систем.

Формальная запись построенной оптимизационной модели выглядит так:

$$П_в(n, m_1, \ldots, m_i, \ldots, m_N, T_{01}, \ldots, T_{0i}, \ldots, T_{0N}) \to \max,$$

$$С_в(n, m_1, \ldots, m_i, \ldots, m_N, T_{01}, \ldots, T_{0i}, \ldots, T_{0N}) \to \min$$

В данной модели приняты следующие обозначения:

$П_в$ - критерий оценки производительности,

$С_в$ - критерий оценки стоимости.

n – количество шин, $1 \le n \le 16$,

m_i – количество процессоров i-го типа, $1 \le m_i \le 16$, $i = \overline{1, N}$,

T_{0i} – среднее время выполнения одной команды в процессоре i-го типа, $T_{0i} > 0$, $i = \overline{1, N}$.

В данной работе кроме выбора оптимальной структуры МСОИУ, т.е. оптимизацию по параметрам n, m_1, …, m_i, …, m_N производится также настройка параметров быстродействия спецпроцессоров T_{0i}. Выбор эффективного быстродействия спецпроцессоров необходим по той причине, что при фиксированном количестве шин, объединяющих процессоры с оперативной памятью, производительность МСОИУ в целом при простом увеличении быстродействия спецпроцессоров может уменьшаться за счет увеличения числа конфликтов, возникающих при одновременном обращении процессоров к оперативной памяти. Поэтому спецпроцессоры должны

проектироваться именно с эффективным, а не максимальным, как до сих пор, быстродействием.

Критерий стоимости вступает в противоречие с критерием производительности. Кроме того, при выборе метода решения данной задачи необходимо учитывать следующие факторы: переменные задачи выражены в различных шкалах измерения (целочисленные и вещественные переменные), оптимизация производится по нескольким экстремальным критериям одновременно, априорные сведения о свойствах целевых функционалов отсутствуют (оптимизация производится только по измерениям функционалов в фиксированных точках). Метод оптимизации должен учитывать условия накладываемые на критерии. Таким образом при выборе эффективной конфигурации МСОИУ возникает необходимость решения задачи многокритериальной условной оптимизации с алгоритмически заданными функциями разношкальных переменных высокой размерности.

Всеми необходимыми качествами для решения такой задачи обладает метод стохастической оптимизации, имитирующий естественный отбор в природе - генетический алгоритм. Для работы генетического алгоритма необходимо закодировать каждое решение в бинарную строку конечной длины – хромосому. Однако при решении рассматриваемой задачи алгоритм должен работать с вариантами МСОИУ произвольной конфигурации. То есть, заранее не известно не только значение параметров МСОИУ, но и их количество. Действительно, в качестве вариантов МСОИУ, которые рассматривает алгоритм, могут оказаться такие, которые отличаются количеством типов процессоров, а значит и количеством наборов данных. Поэтому для решения задачи выбора эффективной конфигурации МСОИУ генетический алгоритм должен быть существенно модифицирован, для работы с хромосомами разной длины не могут быть использованы традиционные операторы генетического алгоритма. Для решения этой проблемы был предложен и реализован оригинальный оператор процентного скрещивания, позволяющий скрещивать хромосомы разной длины. Схема одноточечного процентного скрещивания для случая, когда в качестве точки скрещивания выбирается точка – 25%, представлена на рисунке 1. Схема двухточечного процентного скрещивания представлена на рисунке 2, где в качестве точек скрещивания выбраны точки 25% и 70%.

Использование в модифицированном генетическом алгоритме хромосом переменной длины позволяет значительно сократить время вычисления оценок производительности МСОИУ, а также позволяет алгоритму самому определять структуру МСОИУ. Алгоритм с фиксированной длиной хромосом должен искать решение в пространстве размером 2^{90}, просматривая хромосомы с большим количеством неиспользуемых бит. Модифицированный генетический алгоритм, благодаря использованию хромосом переменной длины, ищет решение в пространстве

размером примерно 2^{60}. Кроме того, осуществляя настройку быстродействия спецпроцессоров, алгоритм способен повышать производительность МСОИУ, в том числе и с оптимальными структурами, найденными ранее.

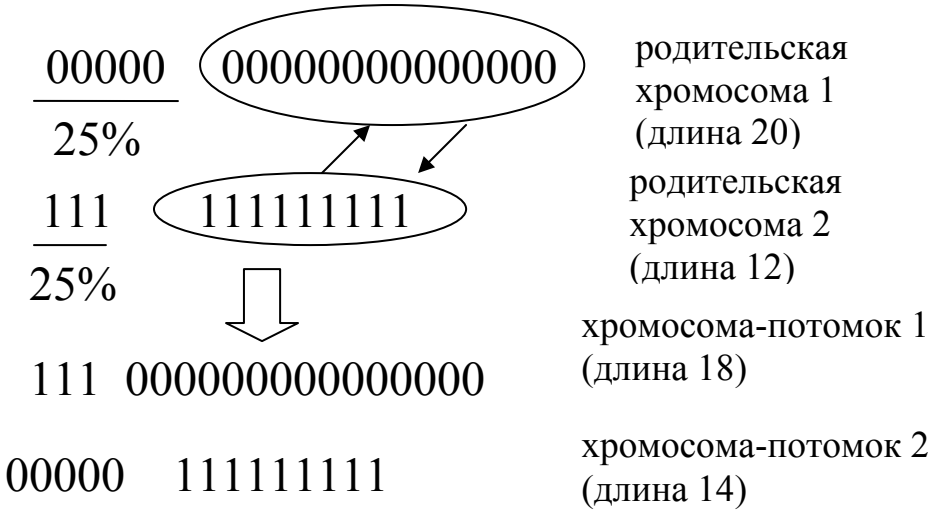


Рисунок 1 – Схема одноточечного процентного скрещивания

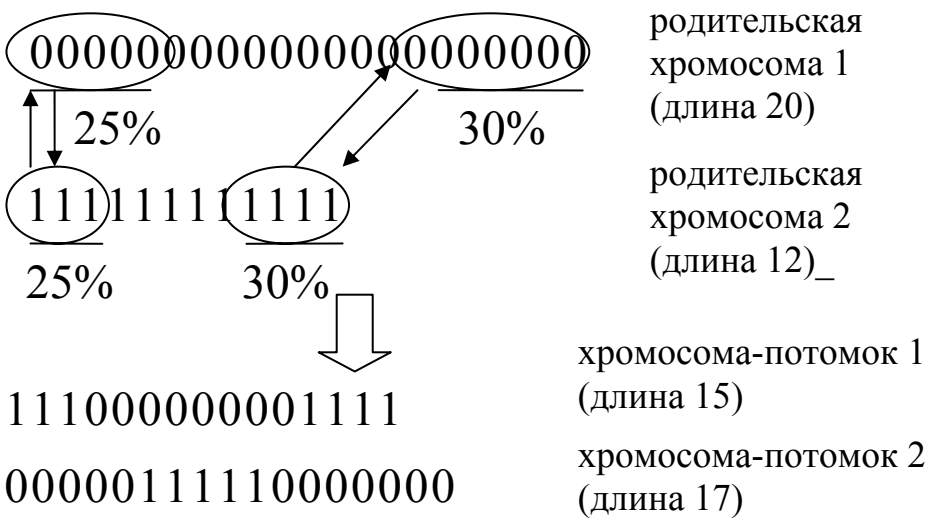


Рисунок 2 – Схема двухточечного процентного скрещивания.

Для работы с несколькими критериями была предложена гибридная схема многокритериального генетического алгоритма, которая представляет собой метод двуальтернативной адаптации алгоритмов FFGA и VEGA. Параметром адаптации является длительность задействования альтернативных алгоритмов, определяемая показателями эффективности их работы. В результате работы по новой схеме генетический алгоритм удачно сочетает в себе сильные стороны обоих алгоритмов. Благодаря алгоритму VEGA, решения в финальной популяции не концентрируются в узкой

области пространства решений, а благодаря работе алгоритма FFGA, решения распределяются в пространстве решений достаточно равномерно.

Была проведена оптимизация реально существующей многопроцессорной системы обработки информации ПС-2000. Эта система получила, в свое время, самое широкое применение. Областью использования ПС-2000 стала геофизика (НПО "Геофизика", Москва).

Таким образом, ПС-2000 можно представить как МСОИУ со следующими характеристиками: количеством шин n от 1 до 4, количество однотипных процессоров – 8, 16, 32 или 64, их быстродействие T_{0i} =0,32 мкс, скорость обработки запроса в памяти τ_i=0,96 мкс. Подбирая быстродействие сразу для всех процессоров входящих в состав одного УО, мы имеем до 8 типов процессоров отличающихся быстродействием.

Имея фиксированное количество типов процессоров, количество процессоров каждого типа и количество шин, нет необходимости выбора оптимальной структуры МСОИУ. Решая данную задачу, достаточно определить оптимальное быстродействие для всех типов процессоров. Это позволяет значительно сократить поисковое пространство. Даже при небольшом количестве циклов работы генетического алгоритма удалось получить такой вариант набора параметров для МСОИУ, который превосходил изначальный, предложенный разработчиками ПС-2000, по относительной производительности более чем в 2 раза, и не уступал при этом по показателю стоимости. В таблице 1 приведены данные по трем МСОИУ. В первой строке находятся данные по модели МСОИУ ПС-2000, в таблице представлены быстродействия процессоров всех типов, а так же рассчитанная стоимость и производительность. Во второй и третьей строке представлены данные по двум моделям МСОИУ, предложенных СППР в качестве наиболее предпочтительных решений.

Таблица 1

	Стоимость	Производительность	T_{01}	T_{02}	T_{03}	T_{04}	T_{05}	T_{06}	T_{07}	T_{08}
1	12.2358	1.64394	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32
2	15.045	4.69568	5.86	2.91	9.33	1.63	0.64	0.45	9.33	5.21
3	11.1412	3.65461	1.15	0.51	8.309	2.31	1.29	0.72	2.31	1.15

Данные по ПС-2000 и решениям полученным СППР

Также было подобрано оптимальное быстродействие спецпроцессоров для МВК «Эльбрус-2».

Вычислительные комплексы “Эльбрус-2” разрабатывались для систем ПРО, ЗРК С-300. МВК «Эльбрус-2» и сейчас используются в качестве управляющей машины в ПРО Москвы.

Система «Эльбрус-2» имеет производительность до 125 млн. оп./с над полноразрядными операндами при 10 процессорах, оперативная память с глубоким расслоением для увеличения быстродействия является общей. Блоки оперативной памяти соединяются с любым центральным процессором

(их может быть до 10) через быстродействующий коммутатор. К этому же коммутатору подсоединяются периферийные процессоры, обслуживающие внешнюю память, периферию и линии связи.

Ранее с помощью имитационных моделей были получены оценки производительности для различных конфигураций МВК «Эльбрус-2».

В таблице 2 приведены результаты расчетов оптимального быстродействия для МВК «Эльбрус-2», различных конфигураций. Для всех вариантов МВК быстродействие шин 1.2 мкс, три типа процессоров. Для реальной МВК «Эльбрус-2» быстродействие процессоров различных типов равно соответственно 0.66 мкс, 8 мкс и 57 мкс.

Таким образом, использование СППР для выбора эффективного быстродействия процессоров многопроцессорной системы обработки информации и управления позволяет не только значительно облегчить и ускорить процесс разработки МСОИУ, а значит сократить затраты на него, но и получать такие варианты МСОИУ, которые по некоторым параметрам превосходят системы, разрабатываемые экспертами обычным образом.

Таблица 2

Кол-во шин	Кол-во процессоров			Стоимость реальной	Стоимость полученой	Быстродействия процессоров, мкс
	1-го типа	2-го типа	3-го типа	Производи-тельность реальной	Производи-тельность полученой	
1	1	1	1	3.70017	3.47593	2.8537; 8; 62.375
				0.693788	1.15317	
	5	5	5	3.70017	3.53116	2.9268; 8; 73.125
				1.48895	4.81033	
	10	5	5	3.70017	3.50974	2.926; 9.12; 59.68
				1.54061	5.62712	
2	1	1	1	4.70017	4.47698	2.9269; 8; 61.03
				0.70855	1.17629	
2	5	5	5	4.70017	4.47031	2.9268; 8; 59.687
				2.61329	5.69036	
	10	5	5	4.70017	4.50362	2.92; 8.37; 63.718
				2.74942	9.01551	
5	1	1	1	7.70017	7.47698	2.9268; 8; 61.03
				0.711456	1.17813	
	5	5	5	7.70017	7.47698	2.9268; 8; 61.03
				3.43083	5.86831	
	10	5	5	7.70017	7.48336	2.9268; 8.75; 57
				5.99941	9.69333	

Данные по МВК «Эльбрус-2» и решениям, полученным СППР

Задача оптимизации работы электростанции на топливных элементах в стационарном режиме

Основной объект данного исследования - малогабаритная ТЭС на топливных элементах для бытового энергообеспечения, исследуемая в институте автоматизации управления при Высшей технической школе города Ульм. ТЭС имеет типичную для PEMFC (топливо – природный газ) структуру, основные элементы которой представлены на рисунке 3.

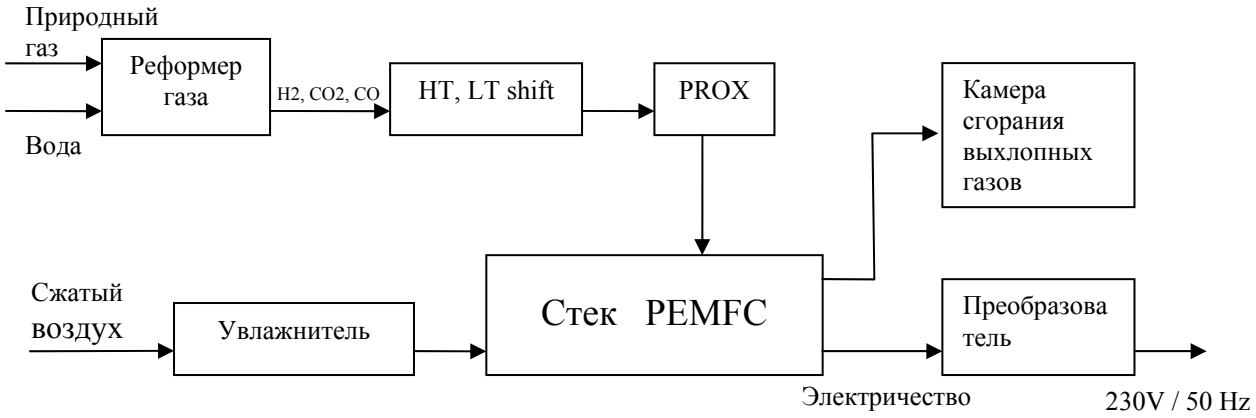


Рис. 3. Основные элементы ТЭС

Вода и пригодный газ поступают в преобразователь (реформер), где образуется водород и одноокись углерода. Далее часть одноокиси углерода преобразуется в двуокись в реакторе высокой температуры (HT shift, 260-320 ^0C), оставшаяся часть – в реакторе низкой температуры (LT shift, 200-260 ^0C). Далее газ поступает в реактор-окислитель (PROX - Preferential Oxidation Reactor), увлажняется и подается в топливные элементы (в исследуемой ТЭС – 40 ячеек, ~5кВт). Отработавший газ сжигается. Вырабатываемое электричество через преобразователь поступает в электрическую сеть.

Ранее для исследуемой электростанции была построена математическая модель, основанная на физико-химической теории процессов, протекающих в топливных элементах, и экспериментальных статистических данных о функционировании подсистем объекта. Библиотека компонентов модели содержит все элементы ТЭС. Модель реализует практически все возможные режимы функционирования станции, а также содержит применяемую на электростанции систему управления. Модель была реализована в среде MatLab Simulink. Параметры системы управления и подсистем ТЭС выбраны в соответствии с техническими ограничениями.

Поставлена следующая задача оптимизации – максимизация электрической производительности (выход тепла в рассмотрение не берется)

для установившегося (стационарного) режима работы. Были выбраны 12 параметров, которые устанавливаются перед запуском электростанции и которые должны оказывать существенное влияние на эффективность работы ТЭС.

Математически критерий оптимальности выглядит следующим образом:

$$\eta = \frac{E_{выход}}{E_{вход}} \cdot 100\%,$$

где $E_{выход} = E_2 - E_1$, где E_2 - полученная электроэнергия, E_1 - суммарное потребление энергии для обеспечения нормального функционирования ТЭС.

$E_{вход} = 802.5 \cdot 10^3 \cdot \dot{n}_{природн.газ}$ - подводимая к системе энергия, пропорциональная количеству используемого топлива, где $802.5 \cdot 10^3$ $\left[Дж/моль \right]$ - константа, связанная с физикой процесса (теплота сгорания), $\dot{n}_{природн.газ}$ - интенсивность расхода природного газа. Таким образом, критерий оптимальности имеет вид:

$$\eta = \frac{E_2 - E_1}{802.5 \cdot 10^3 \cdot \dot{n}_{природн.газ}} \cdot 100\%.$$

Сложность задачи оптимизации состоит в следующем: априорная информация о характере целевого функционала в явном виде отсутствует (не известно является ли функционал непрерывным, выпуклым, квадратичным), зависимость выходных переменных от входных сложная и нелинейная. Получение оценок производных затруднено. Существуют значения параметров (области в пространстве поиска), где функционал не определен - объект не выходит на стационарный режим работы. Очевидно, что использование стохастических методов прямого поиска в данной задаче более предпочтительно, чем использование классических методов оптимизации. Для решения данной задачи использовался вероятностный генетический алгоритм (ВГА), обсуждаемый ранее.

Как известно, вычисления критерия оптимальности могут быть дорогими (в смысле времени, вычислительных затрат и т.д.), поэтому необходимо избегать повторного вычисления критерия в уже исследованных точках. Данная модель ТЭС требует до 20 минут реального времени (в худшем случае) для достижения установившегося режима, поэтому исследованные алгоритмом точки заносятся в базу данных, реализованную с помощью средств Matlab. Поэтому после остановки ВГА, было принято решение по данным, полученным в ходе оптимизации, построить модель зависимости производительности электростанции от контролируемых параметров. За время работы ВГА было исследовано 575 различных решений, которые вместе с рассчитанной производительностью были сохранены в базе данных.

Символьная модель строилась с помощью гибридного алгоритма генетического программирования, обсуждаемого ранее.

Использовались следующие настройки метода ГП:

- множество функций: $\{+,-,*,\div, Sin\}$,
- множество термов: $\{X_i, i = \overline{1,12}\}$,
- размер популяции: 100,
- тип селекции: ранговая,
- тип мутации: точечная, вероятность - 0.1.

Использовались следующие настройки ВГА:

- размер популяции: 10,
- число итераций: 20,
- тип селекции: ранговая,
- тип мутации: равномерная,
- значения параметров модели меняются на интервале $[0; 3.2]$ с шагом 0.1,
- размер булевого вектора: 5 бит на параметр.

В результате после 3000 итераций было принято решение остановить работу алгоритма, т.к. в течение последних нескольких сотен итераций заметного улучшения ошибки аппроксимации не произошло. График изменения ошибки показан на рис. 5 (ось с номерами итераций имеет логарифмический масштаб).

Получено решение:

$\eta(\bar{X}) =$ *(X10-X6*X4-X8+X3/X1-X4+(X3/X1+5*X6-X2+X6*X4-X12--X9/X1+2*X10-3*X4+2*X7-X5-X8)/(1-(7/10-25/9*X5*X1)/(3/5*X3/X10--X6+X2))+X9/X1^2)/(2*X2-X6-X10+X3+X1*X4/(X8-X5*(X11-X3/X1+X8)))*.

После преобразования получим:

$$\eta(\bar{X}) = \frac{\left(X_{10} - X_6 X_4 - X_8 + \frac{X_3}{X_1} - X_4 + \frac{X_9}{X_1^2}\right) + A}{2X_2 - X_6 - X_{10} + X_3 + \frac{X_1 X_4}{X_8 - X_5 \cdot \left(X_{11} - \frac{X_3}{X_1} + X_8\right)}},$$

где $A = \dfrac{\left(\frac{X_3}{X_1} + 5X_6 - X_2 + X_6 X_4 - X_{12} - \frac{X_9}{X_1} + 2X_{10} - 3X_4 + 2X_7 - X_5 - X_8\right)}{\left(1 - \frac{0.7 - 2.78 X_5 X_1}{0.6\frac{X_3}{X_{10}} - X_6 + X_2}\right)}$

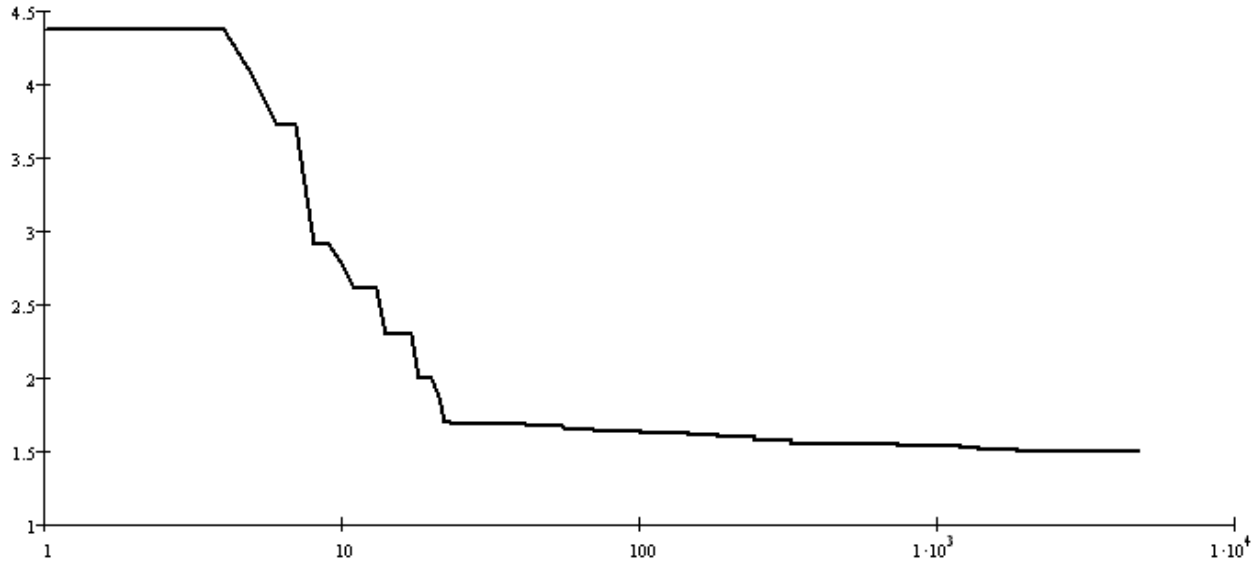


Рис. 5. График изменения ошибки аппроксимации

Ввиду сложности исследуемого объекта, символьное выражение зависимости производительности от контролируемых параметров в стационарном режиме также имеет сложную структуру. Анализ данного выражения дополнительной информации для оптимизации не дал. Тем не менее, вычисление производительности по символьной модели намного выгоднее в смысле времени, т.к. одно вычисление на инженерной модели может занимать до 20 минут, вычисление по формуле осуществляется практически мгновенно.

В дальнейшем оптимизация параметров ТЭС проводилась в том числе и с использованием полученного аналитического выражения.

Для слежения за динамикой работы ВГА изменение пригодности лучшего и среднего по популяции решения выводятся на экран.

Поскольку ГА являются эвристическими методами случайного поиска, строгой теории позволяющей оптимально выбирать параметры алгоритма, нет. Однако многолетний опыт использования ГА дает нам следующие практические советы:

- Соотношение размер популяции – число итераций должно быть примерно равным. Соотношения 100-10 или 10-100 хуже, чем 25-40 или 40-25.
- Следует использовать высокую мутацию, чтобы избежать захвата локальными оптимумами.
- Следует использовать ранговую или турнирную селекцию, чтобы избежать стагнации.

Были выбраны следующие параметры ВГА:

- Число переменных задачи – 12.

- Интервалы изменения переменных: [0.034,0.3; 3.7,10; 17,50; 1123,1223; 1.2,2; 323,343; 473,723; 413,513; 1.3,4; 0.1,1; 0.1,2.5; 0.1,2.5].
- Точность поиска: [0.01; 0.1; 3; 10; 0.1; 1; 10; 10; 0.1; 0.1; 0.1; 0.1].
- Размерность задачи после бинаризации – 55.
- Размер популяции – 25.
- Селекция – турнирная (размер равен 2).
- Мутация – высокая.

Ниже представлен график изменения пригодности (значения критерия оптимальности) для лучшего решения – сплошная линия и среднего значения по популяции – пунктирная (рис. 5).

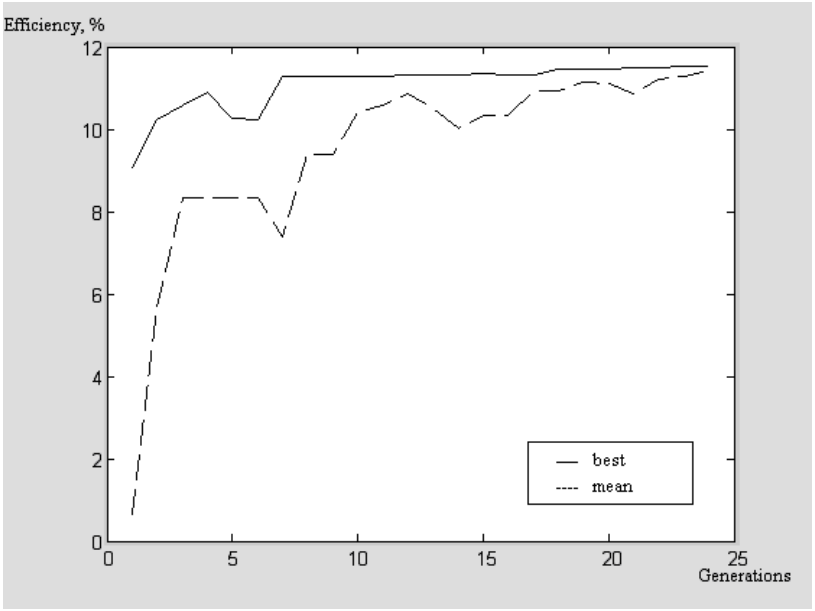


Рис. 5. График изменения пригодности

Из графика видно, что на последних итерациях среднее по популяции значение пригодности слабо меняется и близко к значению лучшего. Это может означать, что большая часть популяции сконцентрирована в области одного и того же оптимума. Поэтому после 23 итераций было принято решение прекратить поиск. Значения найденных параметров представлены в следующей таблице:

Таблица 3.

Parameter	BZ_np_Luft	PM2_Qsoll	Verdichter_f
Value	0.12839	7.3e-07	22
Parameter	T_Ref_Br_soll	BZ_p_soll	st_Tsoll_BZ_Wasser
Value	1223	1.4286	323

Parameter	st_Tsoll_HT_Kuehlung	st_Tsoll_NT_Kuehlung	V23.psoll
Value	522	420	2.9548
Parameter	prox.X_luft	Bef_A.Q_ein	Bef_K.Q_ein
Value	0.16	0.87419e-05	1.2613e-05

Полученный набор параметров дает значение электрической производительности равное 11.5%. Значение производительности, полученное для известных параметров используемых на исследуемой ТЭС – 5.5%. Следовательно, полученное решение позволяет повысить эффективность (на модели) в 2 раза.

Поскольку найденное решение должно быть реализовано на реальной ТЭС, необходимо проверить является ли решение устойчивым. Т.е. определить, как меняется значение электрической производительности, при отклонении параметров от оптимального значения.

Каждый параметр варьировался в пределах 20% от его значения при фиксированных значениях остальных параметров. Найденное решение оказалось достаточно устойчивым. Небольшие изменения параметров не привели к существенным изменениям в значении производительности. Исключение лишь параметр расхода природного газа – можно увеличить значение этого параметра для смещения в область устойчивости.

После запуска алгоритма покоординатного спуска из найденной точки, было найдено решение, дающее значение производительности – 11.9%.

Найденный с помощью ВГА набор параметров был реализован на реальной ТЭС, исследуемой в институте автоматизации управления Высшей Технической школы г. Ульм, Германия. Полученное значение электрической производительности оказалось меньше полученного на модели (предположительно из-за некачественных топливных элементов, используемых на ТЭС), но существенно больше, чем значение производительности при ранее известных параметрах.

Прогнозирование деградации электрических характеристик солнечных батарей космического аппарата.

Необходимо по имеющимся результатам измерения параметров солнечных батарей (БС) в полёте (параметры секций БС3 и БС4 на космическом аппарате (КА) ЭКСПРЕСС-А №2) и результатом измерения параметров с КА ЭКСПРЕСС-А и GOES в годы активного Солнца (25 событий за период с 04.04.00 по 22.11.01, от экстремально мощных 14.07.00 и 22.11.01 до обычных - 16.10.00, примеры спектра) построить модель, прогнозирующую деградацию электрических характеристик БС.

Модель настраивается на определение электрических характеристик солнечных батарей в зависимости от следующих факторов (входной вектор):

- интегральный флюенс протонов с различными энергиями (от 1 до 100 МеВ);
- интегральный флюенс электронов с различными энергиями (от 0,6 до 2 МеВ);
- ресурс - это параметр, который задан как количество дней с момента контакта отделения КА, характеризует повреждения от метеоритных тел и от ультрафиолетового излучения;
- коэффициент освещенности КА – величина, характеризующая степень освещенности аппарата, зависит от взаимного положения КА, Земли и Луны, имеет принципиальное значение, т.к. некоторые выходные характеристики ФП, в частности сила тока короткого замыкания, зависят от освещенности БС.

Вышеперечисленным данным в соответствие ставятся: напряжение холостого хода U_{xx} и сила тока $I_{кз}$ БС (вектор значений или выходной).

Таким образом, на основании ТМИ (полетных данных) и строится нейросетевая модель (пример полученной НС приведен на рисунке 3.12). В нейронной сети формируются внутренние закономерности, позволяющие предсказывать значения выходных параметров для «пробного» набора внешних воздействий.

Далее выполняется прогноз параметров БС на конец ресурса для заданных в ТЗ характеристик ИИКП.

Результат. Для решения данной задачи были получены нейросетевые модели и модели на нечеткой логике. Для обучения из выборки (объемом 220 записей) случайным образом было удалено 9% примеров (20 записей). На оставшейся выборке было получено четыре нейросетевые модели и четыре модели на нечеткой логике для прогнозирования I_БС3 (сила тока БС3), I_БС4 (сила тока БС4), Uxx_БС3 (напряжения холостого хода БС3), Uxx_БС4 (напряжения холостого хода БС4) соответственно (Таблица 3.1.). Ошибка считалась по формуле (1.4).

Данная задача решалась с помощью комплексной процедуры использования эволюционных алгоритмов, автоматически формирующих нечеткие системы управления и нейронные сети, суть ее изложена далее.

Известно, что нейронные сети (НС) и нечеткие системы управления (НСУ) являются взаимозаменяемыми, т.е. одну и туже задачу можно решить, как с помощью технологии искусственных НС, так и с помощью НСУ. Однако существуют следующие различия. НС является моделью «черного ящика», поэтому исследователь, обучив НС на решение конкретной задачи, не имеет представления о том, как решается задача. При этом НС успешно применяются в исследованиях и способны решить задачу с хорошей точностью. НСУ являются моделью «белого ящика», поэтому исследователь, настроив НСУ на решение задачи, может получить полное представление о том, каким образом решается задача, имеет представление о внутренней

структуре объекта, связанного с задачей. С другой стороны, при решении некоторых практических задач, точность решения некоторых задач технологиями НСУ уступает решению НС. В этом смысле НСУ и НС являются взаимодополняемыми.

Комплексная процедура использования эволюционных алгоритмов, автоматически формирующих НСУ и НС, состоит в следующей последовательности действий:

1. Сгенерировать и обучить на задачу НСУ.

Получить базу правил (БП) на сгенерированных лингвистических переменных, равномерно покрывающих интервал изменения переменной;

На полученной базе обучить с помощью оптимизационного алгоритма (генетического или метода сопряженных градиентов);

Повторять шаги 1.1 и 1.2 до тех пор, пока точность решения будет улучшаться.

Если задача решается с необходимой точностью, то необходимость применения НС отпадает. Для улучшения точности решения можно варьировать начальными параметрами ГП (количество индивидов, количество поколений, тип рекомбинации, уровень мутации и т.д.) и настройками лингвистических переменных (количество и вид термов по каждой переменной). Если полученная точность не удовлетворительна – переход к 2.

2. Сгенерировать и обучить на задачу НС.

2.1. Изначально задаются небольшие значения настроек эволюционного алгоритма (начальный уровень деревьев, количество поколений, количество индивидов, количество шагов оптимизационных алгоритмов, количество оптимизационных алгоритмов и т.д.);

2.2. Если точность полученного решения удовлетворительная – переход к шагу 2.3, иначе – увеличить значение настроек и повторить шаг 2.1;

2.3. Анализ полученной НС. Необходимо выяснить, какие входы использует НС для решения поставленной задачи. Чем более сложная структура НС получилась, тем более сложной будет БП в НСУ и тем большей глубины необходимо задавать дерево при начальной инициализации БП.

3. Обучить НСУ с использованием информации, полученной на шаге 2. Для исследований целесообразнее использовать ту модель, которая дает лучшую точность.

Однако даже если в целом НСУ решает задачу с неудовлетворительной точностью, отдельные правила из БП могут достаточно точно решать задачу в своем классе условий. Для выявления таких правил необходимо проанализировать БП в целом, оценить количество срабатываний и ошибку каждого правила в отдельности:

$$E(j) = \frac{\sqrt{\sum_{i=1}^{n_j} (x_j^i - x_j^{i*})^2}}{n_j}$$

Здесь $E(j)$ - ошибка *j*-го правила, n_j - количество срабатываний *j*-го правила, x_j^i - *i*-й вывод *j*-го правила, x_j^{i*} - истинное значение для *i*-го вывода *j*-го правила. Доверие к правилу с низкой частотой срабатывания должно быть низким, т.к. оно может отображать не истинные закономерности, а шум. В полученной БП могут присутствовать правила с нулевой частотой срабатывания. Такие правила необходимо игнорировать при анализе НСУ. Правило с высокой частотой срабатывания и малой ошибкой – является ценным с точки зрения практического использования, анализ таких правил позволяет «просветлить» внутреннюю структуру исследуемого объекта.

Вероятность ошибки правила может снизиться, если оно скорректировано экспертом.

Для того чтобы специалисты, не являющиеся экспертами в интеллектуальных информационных технологиях, могли использовать предложенную комплексную процедуру, была выполнена программная реализация обоих подходов в виде лабораторных установок с дружественным к пользователю интерфейсом и интуитивно ясной структурой управления программой. Данные программы прошли апробацию среди студентов в Сибирском государственном аэрокосмическом университете, Красноярском государственном университете и, в заключение, в Красноярской государственной медицинской академии (студенты-медики). Затем программные системы прошли экспертизу отраслевого фонда алгоритмов программ и были зарегистрированы на отраслевом и государственном уровне.

Решение описанной задачи с помощью разработанной процедуры интеллектуального анализа данных дало следующие результаты.

Таблица 4.

Результат моделирования

Выход	Ошибка НС, обучающая выборка	Ошибка НС, тестовая выборка	Ошибка НСУ, обучающая выборка	Ошибка НСУ, тестовая выборка
I_БС3	0.002174 (1.19%)	0.006080 (3.34%)	0.002151 (1.82%)	0.005307 (2.91%)
I_БС4	0.000884 (0.48%)	0.005419 (2.97%)	0.001164 (0.63%)	0.003771 (2.07%)
Uxx_БС3	0.000370 (0.30%)	0.001060 (0.87%)	0.000383 (0.31%)	0.001260 (1.04%)
Uxx_БС4	0.000236 (0.19%)	0.000906 (0.75%)	0.000265 (0.21%)	0.001201 (0.99%)

Из таблицы 3.1 видно, что для прогнозирования силы тока короткого замыкания лучше использовать модели на основе НСУ, а для прогнозирования напряжения холостого хода лучше использовать модели на основе НС.

Приведем описание соответствующих моделей.

Нейросетевые модели. Для решения данной задачи был применен эволюционный алгоритм с параметрами: количество индивидов – 15; количество популяций – 5; селекция – турнирная; количество в турнире – 4; рекомбинация – стандартная; мутация – 0.01; штраф – 0.00001; начальная глубина дерева – 3; максимальная глубина дерева – 15; оптимизация весовых коэффициентов – генетический алгоритм и алгоритм сопряженных градиентов; оптимизация пороговых значений нейронов – генетический алгоритм; блоки входных нейронов – 3 блока (1-й блок состоит из 1, 2, 3 входов, 2-й блок состоит из 4, 5, 6 входов, 3-й блок состоит из 7-го входа), блоки на скрытых слоях состоят из одного нейрона следующих активационных функций: сигмоидальная, гиперболический тангенс, линейная, линейная с насыщением, гауссова, пороговая. В результате получена НС для прогнозирования Uxx_БС3 с неполным входным слоем (рисунок 6).

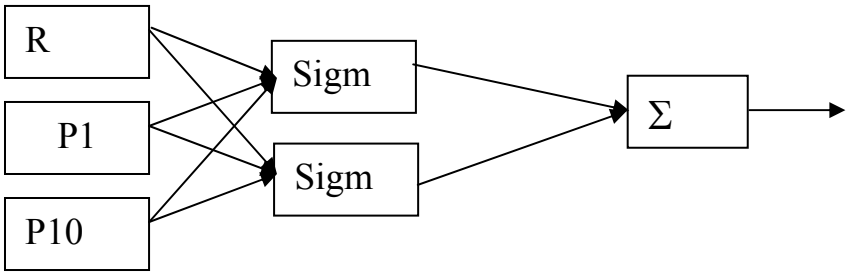


Рисунок 6. Нейронная сеть для прогнозирования Uxx_БС3

Здесь R – ресурс, P1 – протоны с энергией большей 1 MeV, протоны с энергией большей 10 MeV, Sigm – функция активации сигмоидного типа.

На вход НС подаются только ресурс, интегральный флюенс протонов с энергиями более 1 и 10 MeV.

Т.к. природа выходов Uxx_БС3 и Uxx_БС4 одинакова, то возможно для прогнозирования выхода Uxx_БС4 также достаточно только этих входов. Чтоб проверить это утверждение переобучим НС, прогнозирующую Uxx_БС3, для Uxx_БС4. В результате чего ошибка прогноза уменьшилась на обучающей выборке и незначительно увеличилась на тестовой.

Простота полученных нейронных сетей позволяет преобразовать их в несложные аналитические выражения вида:

$$\mathrm{Uxx} = \frac{W_1}{1+\exp(-\alpha(w_{11}\cdot R + w_{12}\cdot P1 + w_{13}\cdot P10))} + \frac{W_2}{1+\exp(-\beta(w_{21}\cdot R + w_{22}\cdot P1 + w_{23}\cdot P10))}$$

Модели на нечеткой логике. Для решения данной задачи был применен эволюционный алгоритм с параметрами: количество индивидов – 700, количество популяций 60, селекция – турнирная, количество в турнире – 35, рекомбинация – стандартная, мутация – 0.01, штраф – 0.00001, начальная глубина дерева – 4, максимальная глубина дерева – 10, оптимизация лингвистических переменных – генетический алгоритм и алгоритм сопряженных градиентов.

Полученные нечеткие правила с помощью настроенных лингвистических переменных были приведены в четкий вид. В таблице 3.2. и 3.2 приведены правила для прогнозирования I_БС3 и I_БС4 соответственно.

Таблица 5

Правила для прогнозирования I_БС3

Правило	Ошибка правила	Кол. срабатываний
Если (Инт. флюенс протонов с энергией > 1 МеВ МЕНЕЕ $3.1*10^9$) и (Инт. флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (коэффициент освещения МЕНЕЕ 0.94), ТО сила тока к.з. [0.885 0.908]	0.004318 (%2.39)	38
Если (Инт. флюенс протонов с энергией > 1 МеВ БОЛЕЕ $3.1*10^9$) и (Инт. флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (коэффициент освещения МЕНЕЕ 0.94), ТО сила тока к.з. [0.855 0.875]	0.004938 (%2.74)	22
Если (коэффициент освещения МЕНЕЕ [0.94 0.961]), ТО сила тока к.з. [0.885 0.908]	0.004617 (%2.56)	52
Если (Ресурс МЕНЕЕ 122 дня) и (коэффициент освещения [0.961 0.981]), ТО сила тока к.з. [0.915 0.936]	0.003587 (% 1.99)	8
Если (Ресурс [122 171] дней) и (Инт. флюенс протонов с энергией > 10 МеВ БОЛЕЕ $9.2*10^8$) и (коэффициент освещения [0.961 0.981]), ТО сила тока к.з. [0.967 0.993]	0.014244 (% 7.91)	5
Если (Ресурс [171 221] дней) и (Инт. флюенс протонов с энергией > 10 МеВ БОЛЕЕ $9.2*10^8$) и (коэффициент освещения [0.961 0.981]), ТО сила тока к.з. [0.915 0.936]	0.005191 (% 2.88)	8
Если (Ресурс БОЛЕЕ 221 дней) и (Инт. флюенс протонов с энергией > 10 МеВ БОЛЕЕ $9.2*10^8$) и	0.005936 (%)	17

(коэффициент освещения [0.961 0.981]), ТО сила тока к.з. [0.885 0.908]		
Если (Инт. флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (Инт. флюенс электронов с энергией > 2 МеВ МЕНЕЕ $4.75*10^{10}$) и (коэффициент освещения [0.981 1]), ТО сила тока к.з. [0.915 0.936]	0.005023 (% 2.79)	7
Если (Инт. флюенс электронов с энергией > 2 МеВ МЕНЕЕ $4.75*10^{10}$) и (коэффициент освещения БОЛЕЕ 1), ТО сила тока к.з. [0.936 0.964]	0.006296 (%3.49)	8
Если (Инт. флюенс протонов с энергией > 100 МеВ БОЛЕЕ $1.275*10^{7}$) и (Инт. флюенс электронов с энергией > 2 МеВ [$4.75*10^{10}$ $7.7*10^{10}$]) и (коэффициент освещения БОЛЕЕ 0.981), ТО сила тока к.з. [0.967 0.993]	0.010596 (%5.88)	17
Если (Инт. флюенс электронов с энергией > 0.6 МеВ БОЛЕЕ $2.2*10^{12}$) и (Инт. флюенс электронов с энергией > 2 МеВ БОЛЕЕ $7.7*10^{10}$) и (коэффициент освещения БОЛЕЕ 0.981), ТО сила тока к.з. [0.936 0.964]	0.005532 (%3.07)	39

Таблица 6

Правила для прогнозирования I_БС4

Правило	Ошибка правила	Кол. срабатываний
Если (Интегральный флюенс протонов с энергией > 10 МеВ МЕНЕЕ $8.3*10^{8}$) и (Интегральный флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (коэффициент освещения МЕНЕЕ 0.906), ТО сила тока к.з. БС4 [0.893 0.91]	0.005169 (%3.54)	29
Если (Интегральный флюенс протонов с энергией > 10 МеВ МЕНЕЕ $2.75*10^{8}$) и (коэффициент освещения [0.906 0.934]), ТО сила тока к.з. БС4 [0.918 0.934]	0.003669 (%2.51)	7
Если (Интегральный флюенс протонов с энергией > 10 МеВ БОЛЕЕ $8.3*10^{8}$) и (Интегральный флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (коэффициент освещения МЕНЕЕ 0.934), ТО сила тока к.з. БС4 МЕНЕЕ 0.889	0.003042 (%2.08)	13

Если (Интегральный флюенс протонов с энергией > 10 МеВ [$8.3*10^8$ $1.4*10^8$]) и (Интегральный флюенс электронов с энергией > 0.6 МеВ БОЛЕЕ $2.2*10^{12}$) и (коэффициент освещения МЕНЕЕ 0.934), ТО сила тока к.з. БС4 [0.893 0.91]	0.005235 (%3.58)	6
Если (Интегральный флюенс протонов с энергией > 10 МеВ БОЛЕЕ $1.4*10^8$) (Интегральный флюенс электронов с энергией > 0.6 МеВ БОЛЕЕ $2.2*10^{12}$) и и (коэффициент освещения МЕНЕЕ 0.934), ТО сила тока к.з. БС4 [0.918 0.934]	0.008451 (%5.78)	1
Если (коэффициент освещения [0.935 0.971]), ТО сила тока к.з. БС4 [0.918 0.934]	0.001787 (%1.22)	78
Если (коэффициент освещения [0.971 1]), ТО сила тока к.з. БС4 [0.96 0.977]	0.001512 (%1.03)	79
Если (Интегральный флюенс электронов с энергией > 0.6 МеВ МЕНЕЕ $2.2*10^{12}$) и (коэффициент освещения БОЛЕЕ 1), ТО БОЛЕЕ 0.977	0.002832 (%1.94)	8

Лингвистические переменные для I_БС3 и I_БС4 приведены на рисунках 7 - 14 и 15 - 22 соответственно.

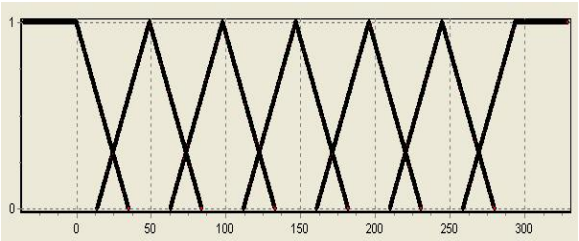


Рисунок 7. I_БС3. Ресурс

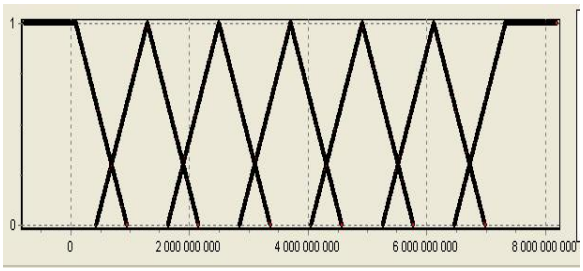


Рисунок 8. I_БС3. Интегральный флюенс протонов с энергией > 1 MeV

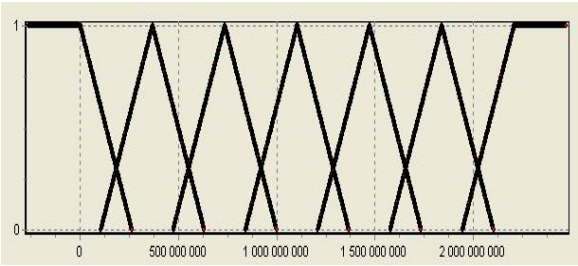


Рисунок 9. I_БС3. Интегральный флюенс протонов с энергией > 10 MeV

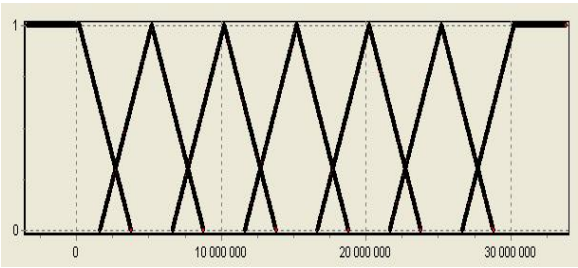


Рисунок 10. I_БС3. Интегральный флюенс протонов с энергией > 100 MeV

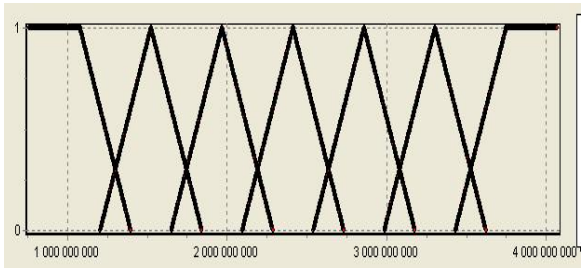


Рисунок 11. I_БС3.Интегральный флюенс электронов с энергией > 0,6 MeV

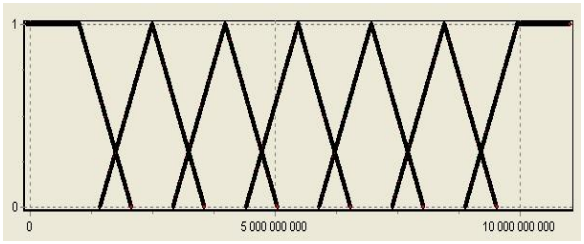


Рисунок 12. I_БС3. Интегральный флюенс электронов с энергией > 2 MeV

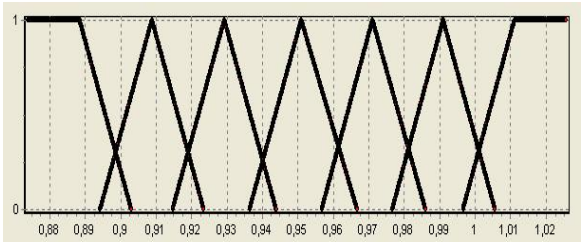


Рисунок 13. I_БС3. Коэффициент освещения

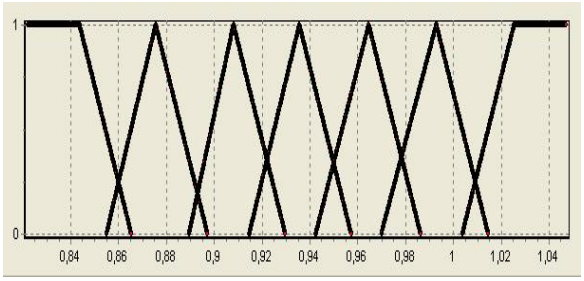


Рисунок 14. I_БС3.Выход

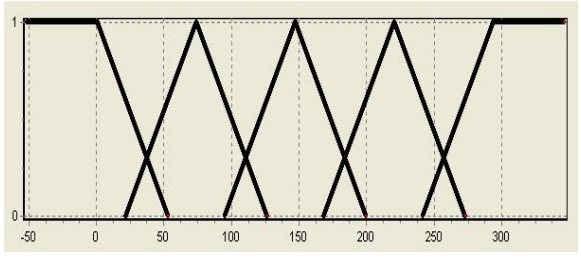


Рисунок 15. I_БС4. Ресурс

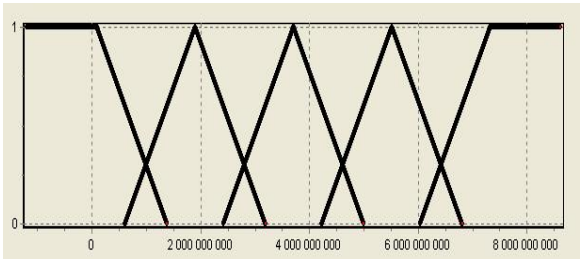


Рисунок 16. I_БС4. Интегральный флюенс протонов с энергией > 1 MeV

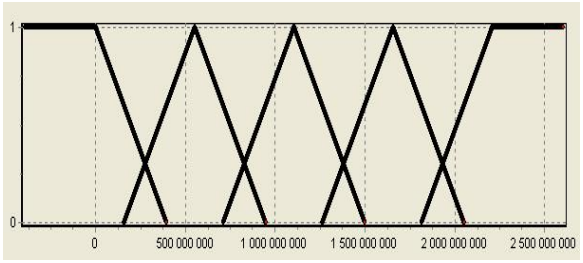


Рисунок 17. I_БС4. Интегральный флюенс протонов с энергией > 10 MeV

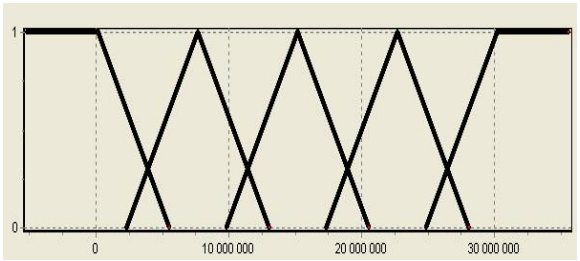


Рисунок 18. I_БС4. Интегральный флюенс протонов с энергией > 100 MeV

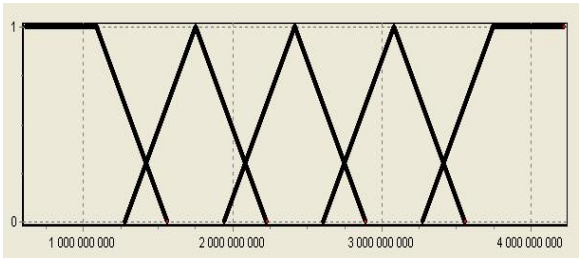


Рисунок 19. I_БС4.Интегральный флюенс электронов с энергией > 0,6 MeV

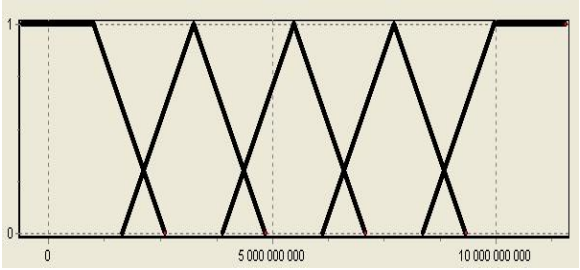


Рисунок 20. I_БС3. Интегральный флюенс электронов с энергией > 2 MeV

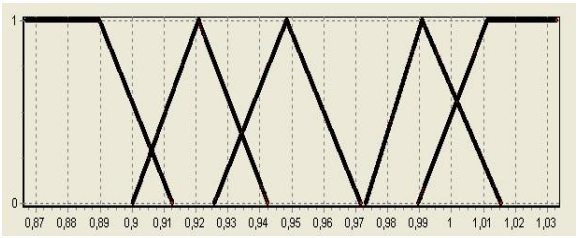


Рисунок 21. I_БС4. Коэффициент освещения

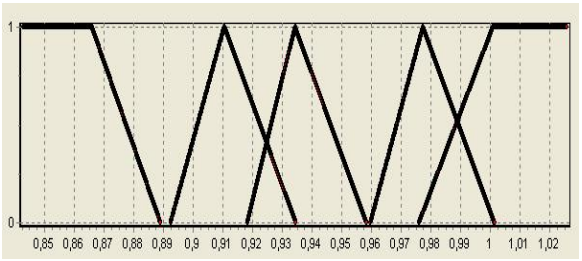


Рисунок 22. I_БС4.Выход

Задача формирования кредитного портфеля банка: нахождение оптимального набора кредитных заявок при согласованном виде структуры активов – пассивов

Управление формированием оптимального кредитного портфеля при наличии жестких ограничений по суммам имеющихся в наличии свободных кредитных ресурсов, их стоимости, процентным ставкам на выдаваемые кредиты, срокам привлечения ресурсов, максимальному размеру кредита на одного заемщика - постоянная процедура, которую выполняют специалисты банка. При этом от правильности этих решений зависит финансовая стабильность банка, а значит, и защита интересов его инвесторов. Серьезные проблемы с ликвидностью, испытываемые основной массой коммерческих банков, требуют повышения эффективности процесса управления формированием активных и пассивных операций. Это в полной мере касается кредитных операций, составляющих до 60% активов.

Анализ проблемы и постановка задачи. “Кредит - предоставление банком или кредитной организации денег заемщику в размере и на условиях, предусмотренных кредитным договором, согласно которого заемщик обязуется возвратить полученную сумму и уплатить проценты по ней”.

Роль кредитной политики банка заключается в определении приоритетных направлений развития и совершенствования банковской деятельности в процессе аккумуляции и инвестирования кредитных ресурсов, развитии кредитного процесса и повышении его эффективности. Каждый конкретный банк определяет свою собственную кредитную политику. Совокупный риск банка повышается, если последний не имеет собственной кредитной политики, либо имеет кредитную политику невысокого качества.

Основными элементами процесса кредитования, определяющими решения специалистов банка при выдаче кредитов, являются:

1. Сумма *j*-го кредита - объем средств, предоставляемых банком клиенту на платной, возвратной и обеспеченной основе на определенные цели, соответствующие его финансово - хозяйственной деятельности.

2. Риск невозврата *j*-го кредита - вероятность того, что заемщик не выполнит взятые на себя обязательства по возврату полученного им кредита и процентов за его использование.

3. Максимальный размер выдаваемого банком *j*-го кредита одному заемщику - максимальная сумма средств, которую может предоставить в долг банк клиенту. При этом данная величина не зависит от клиента. Она определяется по методике Центрального Банка России, исходя из финансового состояния банка, размера его уставного капитала и других параметров.

4. Процентная ставка, по которой банк предоставляет кредит *j*-му заемщику.

5. Срок кредитования *j*-го заемщика - период, на который предоставлен кредит. По его истечении сумма кредита и проценты по нему должны быть возвращены банку.

6. Объем поданных кредитных заявок - сумма всех кредитных заявок, поданных клиентами банка на получение кредитов.

7. Объем свободных кредитных ресурсов в момент рассмотрения кредитных заявок - сумма средств, которыми располагает банк для инвестирования в кредиты.

8. Ликвидность временной структуры активов - пассивов банка - способность банка выполнять принятые на себя обязательства перед кредиторами.

9. Возможный доход от кредитования заемщика - ожидаемый доход в виде платы за пользование предоставленным кредитом *j*-му заемщику.

10. Максимально допустимый размер риска невозврата, определенный для себя банком - величина риска невозврата, определяемая кредитной политикой банка как максимальная (кредитные заявки с большим риском невозврата не удовлетворяются).

Необходимым условием для рассмотрения кредитной заявки является предоставление технико-экономического обоснования эффективности запрашиваемого кредита, договоров по кредитуемому проекту. Специалисты банка на основе этих документов определяют обоснованность просьбы клиента о выделении кредита, а также финансово - экономическую возможность заемщиком возвратить запрашиваемый кредит и проценты по нему в срок и в полном объеме. В итоге анализа всех вышеперечисленных документов, а также юридических документов заемщика, наличия опыта работы с ним ранее, банк определяет величину кредитного риска, присущего рассматриваемому кредиту. Безусловно, кредитование является одной из самых рисковых операций банка, сопряженной с вероятностью потерь в случае невозврата кредита и процентов; однако одновременно и самой доходной из них, что предопределяет то, что банки всегда будут поводить этот вид операций.

Начальный этап выбора кредитных заявок заключается в проверке ограничений, накладываемых на их параметры:

- сумма запрашиваемого *j*-го кредита не должна превышать максимальный размер кредитного риска, установленного Инструкцией ЦБ РФ в размере 25% от капитала банка;
- период использования запрашиваемого кредита не должен превышать максимальный срок привлечения самого “длинного” источника кредитных ресурсов;
- желаемая процентная ставка запрашиваемого кредита не может быть ниже минимальной цены разм ещения кредитных ресурсов;
- установленный специалистами уровень риска невозврата запрашиваемого кредита не может быть выше предельно допустимого по банку.

В случае неудовлетворения хотя бы одного из предъявленных требований кредитная заявка не допускается к дальнейшему рассмотрению.

Определим понятие ликвидности, как степень постоянной готовности банка выполнить свои обязательства перед кредиторами. Теоретически, вышеназванному критерию ликвидности соответствует равенство активов и пассивов в любом временном интервале. Логичным продолжением этого является то, что сумма удовлетворяемых кредитных заявок в рассматриваемый момент времени не может быть больше суммы имеющихся в этот же момент времени у банка свободных кредитных ресурсов с аналогичным сроком привлечения.

Известно несколько стратегий приведения временной структуры активов - пассивов к согласованному виду. Согласно одной из них, формирование кредитного портфеля в данном временном интервале может производиться без учета ликвидности (согласованности) предыдущих интервалов. В этом случае банк стремится в максимально короткий срок достичь равенства активов и пассивов в каждом временном интервале. Для этого размещает в них кредиты в суммах, равных дефициту ликвидности или не более суммы свободных кредитных ресурсов при равенстве активов и пассивов.

В работе строится реализующая данную стратегию математическая модель задачи формирования кредитного портфеля банка, оптимального по критериям:

1) доходность кредитования;

2) риск невозврата;

3) ликвидность временной структуры активов - пассивов.

В предлагаемой постановке задачи формирование кредитного портфеля производится путем формирования кредитных портфелей в каждом из временных интервалов, на которые поделена временная структура баланса банка.

Математическая формулировка задачи. Для формализованной записи критерия получения максимальной доходности от проводимых банком кредитных операций при соблюдении требования минимизации риска невозврата вводятся следующие обозначения:

F_i - сумма свободных пассивов, которыми располагает банк из i - го временного интервала;

k_{ij} - сумма кредита, запрашиваемая j - м заемщиком с погашением долга в i - м временном интервале, $i = \overline{1, I}; \quad j = \overline{1, J_i}$;

t_{ij} - период размещения средств в k_{ij} - кредит;

x_{ij} - булева переменная, принимающая значения: 1, если кредит k_{ij} выдается и 0, если заявка на получение кредита отклоняется банком;

d_{ij} - проценты за пользование k_{ij} - м кредитом (предполагается, что d_{ij} выплачиваются единовременно с возвратом самого кредита);

P_{ij} - вероятность невыполнения заемщиком обязательств по возврату кредита и процентов по нему ($k_{ij} + k_{ij} d_{ij}$). В предлагаемой постановке задачи предполагается два варианта обслуживания долга заемщиком: 100% возврат суммы кредита и процентов по нему в установленный срок, либо полное отсутствие платежей в погашение кредита и процентов по нему.

Выдача кредитов рассматривается не только как доходный инструмент банка, но и как инструмент, позволяющий повысить ликвидность временной структуры активов – пассивов банка. С этой целью полагаем, что пассивы из временного интервала i в случае их недоиспользования не могут быть инвестированы в кредиты из других временных интервалов. Поэтому недоиспользованная сумма пассивов из i - го временного интервала, равная

$$F_i - \sum_{j \in J} k_{ij} x_{ij},$$

принимает участие в формировании кредитного портфеля с $d_{ij} = 0$ и $P_{ij} = 0$, где J – множество индексов принятых кредитных заявок.

Тогда ожидаемые проценты от комбинации кредитных заявок будут определяться по формуле:

$$E(x_{ij}) = \sum_{j \in J} (k_{ij} + k_{ij} d_{ij} t_{ij}) x_{ij}$$

Рискованность рассматриваемой кредитной заявки:

$$R(x_{ij}) = \frac{1}{|J|} \cdot \sum_{j \in J} P_{ij} x_{ij}$$

В случае согласованной временной структуры активов - пассивов (равенства в каждом временном интервале), кредитование производится в строгом соответствии с объемом и срочностью свободных кредитных ресурсов, которыми располагает банк в момент принятия решения о кредитовании заемщиков. Ограничение, накладываемое на объем выдаваемых кредитов в i - м временном интервале, т. е. в интервале с равными (близкими) значениями активов и пассивов, будет иметь вид:

$$\sum_{j=1}^{J_i} k_{ij} x_{ij} \le F_i$$

Преобразуем к целевой функции:

$$L(x_{ij}) = \left\{ \begin{array}{c} +\infty, if \sum_{j=1}^{J_i} k_{ij} x_{ij} > F_i \\ \hline -\sum_{j=1}^{J_i} k_{ij} x_{ij} + F_i, if \sum_{j=1}^{J_i} k_{ij} x_{ij} \le F_i \end{array} \right\}$$

Решениям, превышающим свободные ресурсы, назначается большой остаток средств и они, таким образом, оказываются малопригодными по функции ликвидности. Решения, которые имеют наименьший остаток средств, являются более пригодными.

Существует второй вариант свертки ограничения в критерий:

$$L(x_{ij}) = \left\{ \begin{array}{c} \sum_{j=1}^{J_i} k_{ij} x_{ij} - F_i, if \sum_{j=1}^{J_i} k_{ij} x_{ij} > F_i \\ \hline 0, if \sum_{j=1}^{J_i} k_{ij} x_{ij} \le F_i \end{array} \right\}$$

Основная идея второго подхода - использовать информацию от недопустимых индивидов в ходе оптимизации. Например, недопустимый индивид может иметь очень хорошее значение прибыли и станет тянуть в свою сторону допустимого индивида с очень плохой прибылью. Это теоретически может помочь найти допустимого индивида с хорошей прибылью.

Таким образом, целевая функция задачи максимизации дохода, минимизации кредитного риска, может быть представлена в следующем виде:

$$E(x_{ij}) \to \max_{x_{ij}},$$

$$R(x_{ij}) \to \min_{x_{ij}}$$

$$L(x_{ij}) \to \min_{x_{ij}}$$

Задача нахождения набора кредитных заявок, оптимально по трем критериям (доходность, риск невозврата и ликвидность) приведена к задаче целочисленного линейного программирования.

Исходные данные для задачи оптимизации набора кредитных заявок представлены в таблице 7.

Таблица 7

№№	Сумма кредита	% ставка	период	Риск
1	10 000 000	25	75	0,042
2	5 300 000	28	80	0,039
3	2400000	25	91	0,029
4	50 000 000	23	84	0,033
5	1 000 000	28	64	0,026
6	500 000	30	76	0,046
7	250 000	37	91	0,044
8	100000	30	86	0,012
9	330 000	26	90	0,026
10	5600000	28	88	0,039
11	7 300 000	25	76	0,02
12	1 220 000	27	80	0,037
13	2 900 000	31	84	0,03
14	950 000	29	86	0,041
15	4 360 000	25	88	0,021

Продолжение таблицы 7

16	3 700 000	26	90	0,035

17	400 000	26	79	0,029
18	280 000	28	84	0,03
19	5200000	30	91	0,039
20	1 280000	27	90	0,04
21	8400000	25	86	0,035
22	670 000	29	80	0,015
23	790 000	28	84	0,024
24	950 000	26	83	0,034
25	580 000	27	90	0,038
26	640 000	24	91	0,042
27	440 000	28	67	0,029
28	460 000	28	91	0,018
29	6 000 000	27	62	0,021
30	7 100000	26	78	0,036
31	3 260 000	27	87	0,027
32	2 670 000	25	75	0,014
33	620 000	27	82	0,038
34	20 000 000	24	91	0,019
35	10000000	24	90	0,026
36	35 000 000	22	89	0,022
37	5 100000	29	69	0,036
38	865000	30	74	0,021
39	675000	27	63	0,017
40	4 650 000	29	69	0,026
41	135 000	28	70	0,03
42	400 000	27	76	0,041
43	1 640 000	26	87	0,017
44	1 380 000	29	88	0,021
45	1 950 000	26	71	0,013
46	1 000 000	27	85	0,014
47	6 900 000	27	82	0,016
48	9 000 000	27	86	0,024
49	22 000 000	29	91	0,016
50	350 000	27	69	0,026
	256 695 000			
	188 500 000 - Свободные ресурсы			

Оптимизация производилась с использованием алгоритмов VEGA, SPEA, NPGA, FFGA.

Решение представлялось битовым вектором длиной 50 (согласно количеству заявок). Полученный битовый вектор решения интерпретировался следующим образом: если первый бит равен 0, то первая заявка на кредит удовлетворяется, иначе – не удовлетворяется, то же самое для второго бита и т.д. Каждым алгоритмом получен некоторый спектр решений с вариациями риска и доходности.

Задача принятия решений при управлении инновационными процессами реструктурированного предприятия ОПК

Решались практические задачи принятия решений при управлении инновационными процессами реструктурированного предприятия ОПК,

взятых с реального предприятия (Химзавода – филиала ФГУП «Красмаш»), в том числе:

– формирования инновационной программы предприятия;

– распределения общих ресурсов при управлении инновациями предприятия ОПК;

– планирования программы выпуска инновационного товара (после реализации инновационного проекта).

При реструктуризации предприятия его руководство (центральная компания) создает систему, которая позволит всем подразделениям стать материально заинтересованными в выборе новой продукции, развитии и увеличении объемов производства. Это так называемые Центры финансовой ответственности (ЦФО), которые непосредственно связаны с процессом производства.

Центральный офис предприятия реализует инвестиционную политику совместной деятельности ЦФО и других участников объединения. С этой целью он концентрирует ресурсы и вкладывает их в новые инновационные проекты, реконструкцию и модернизацию ЦФО и обслуживающих подразделений.

Модель формирования инновационной программы предприятия позволяет определить возможность включения (или невключения) той или иной инновации в общий портфель в соответствии с ее прибыльностью, обеспеченностью финансами и рискованностью внедрения. Меру риска определяют на основании экспертной оценки рискованности каждого отдельного проекта.

Для формализованной записи рассматриваемой модели вводятся следующие обозначения:

$П_{ij}$ – плановый годовой объем прибыли, получаемый i-м ЦФО от внедрения j-го нововведения;

R_{ij} – экспертная оценка рискованности соответствующего инновационного проекта;

c_{ij} — плановые годовые затраты финансовых средств i-го ЦФО на j-е нововведение, способствующее увеличению мощности ЦФО;

C_i — плановые годовые объемы финансовых средств, выделяемые ЦФО в план нововведений, $C = \sum_{i=1}^{m} C_i$;

M —плановый годовой объем финансовых средств, выделяемый центральной компанией в планы нововведений ЦФО;

r — допустимая средняя прибыль на 1 руб. затрат (норма прибыли на капитал);

x_{ij} — искомый параметр, показывающий, планируется ли к внедрению на i-м ЦФО j-е нововведение (если $x_{ij} = 1$, то планируется; если $x_{ij} = 0$, не планируется).

Таким образом, для повышения обоснованности принятия решений при управлении инновационными процессами реструктурированного предприятия ОПК необходимо в общем случае решить задачу условной многокритериальной оптимизации, которая формально выглядит следующим образом:

$$f_1(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij} x_{ij} \to \max; \quad (1)$$

$$f_2(X) = \frac{1}{\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}} \sum_{i=1}^{m}\sum_{j=1}^{n} R_{ij} x_{ij} \to \min; \quad (2)$$

$$g_1(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij} \le C + M; \quad (3)$$

$$g_2(X) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij} x_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} x_{ij}} \le r; \quad (4)$$

$$x_{ij} = \{1,0\}.$$

Представленная модель включает оптимизацию двух целевых функций (критериев) (1) – (2) с булевыми переменными и двумя линейными ограничениями (3) – (4).

Рассматриваемая практическая задача является довольно сложной, решением в данном случае будет не одна точка условного оптимума, а множество точек Парето, принадлежащих допустимой области.

Решения в генетическом алгоритме представляются в виде бинарной строки, а рассматриваемая задача является задачей оптимизации с булевыми переменными, не требуется перевод генотипа в фенотип. В итоге результатом решения задачи будут наборы нулей и единиц – хромосомы, биты которых представляют собой номера соответствующих инновационных проектов, предлагаемых для внедрения на предприятии в плановом году: 1 – инновационный проект принят для внедрения, 0 – проект не входит в итоговый портфель предприятия.

После перевода ограничений задачи оптимизации в критерии, исходная задача свелась к безусловной задаче многокритериальной оптимизации, модель преобразуется в следующий вид:

$$f_1(X) = \sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij} x_{ij} \to \max; \quad (5)$$

$$f_2(X) = \frac{1}{\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}} \sum_{i=1}^{m}\sum_{j=1}^{n} R_{ij} x_{ij} \to \min; \quad (6)$$

$$f_3(X)=\begin{cases}0, \textit{ если } \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \le C+M,\\ \left(\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}-(C+M)\right)^2, \textit{ в противном случае.}\end{cases} \quad (7)$$

$$f_4(X)=\begin{cases}0, \textit{ если } \dfrac{\sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij}x_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}} \le r,\\ \left(\sum_{i=1}^{m}\sum_{j=1}^{n} \Pi_{ij}x_{ij}-r\cdot\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}\right)^2, \textit{ в противном случае.}\end{cases} \quad (8)$$

$$x_{ij}=\{1,0\}.$$

Таким образом, теперь необходимо решить задачу безусловной оптимизации с булевыми переменными четырех представленных критериев (5) – (8).

Согласно постановке задачи допустимая область находится на пересечении ограничений (3) и (4). В связи с тем, что область поиска по сравнению с тестовыми задачами расширяется, а возможных вариантов решения становится больше, размер внешнего множества в методе SPEA был увеличен до 40. Остальные настройки алгоритма не менялись.

Были проверены различные варианты значений параметров задачи (плановый годовой объем финансовых средств, выделяемый центральной компанией в планы нововведений ЦФО (M) и норма прибыли на капитал (r)) и на всех рассмотренных вариантах алгоритм показал свою эффективность. Далее приводятся результаты решения практической задачи при M=40 и r=0.5.

В таблице 8 приведены недоминируемые допустимые решения (колонка 2), получающиеся после 800-го поколения при решении задачи с помощью метода SPEA. Соответствующие им значения критериев (5) – (8) представлены в колонках 3-6.

Таблица 8

№ п/п	Решение условной задачи	Значение f_1, млн. руб.	Значение f_2	Значение g_1	Значение g_2
1	*2*	*3*	*4*	*5*	*6*
1	1011111111111111111011000 1	117.5	1.98	0	0
2	0011101101111111100000000	68.9	1.66923	0	0

3	1 00111000011111101010000 1	65.3	1.63846	0	0
4	00011001011111101010000 1	69.6	1.67692	0	0
5	00111111111111111011000 1	114	1.96842	0	0
6	00011000011110110000000 1	42	1.43333	0	0
7	00111111011111111011000 1	104.65	1.90556	0	0
8	00111000011111110000000 1	54.7	1.54545	0	0
9	00111000011111100010000 1	60.7	1.59167	0	0
10	00011001011111100000000 1	59	1.59091	0	0
11	00111101011111101010000 1	84.3	1.77333	0	0
12	00011000011111100000000 1	51.5	1.52	0	0
13	00011001011111100010000 1	65	1.63333	0	0
14	00111111011111111010000 1	94.65	1.83529	0	0
15	00011000011110110010000 1	48	1.5	0	0
16	00111101011111100010000 1	79.7	1.74286	0	0
17	00011011011111111010000 1	79.95	1.76	0	0
18	00111111011111101010000 1	91	1.8125	0	0
19	00111011011111101010000 1	79.5	1.73333	0	0
20	10111111111111111110000 1	110.45	1.935	0	0
21	10111111011111111010000 1	98.15	1.85556	0	0
22	00111001011111101010000 1	72.8	1.68571	0	0
23	00111001011111100010000 1	68.2	1.64615	0	0
24	00111000011110110000000 1	45.2	1.47	0	0

25	00011000011111110001000001	57.5	1.57273	0	0
26	00111011011111110001000001	74.9	1.7	0	0
27	00111001011111110000000001	62.2	1.60833	0	0
28	10111111011111111011000001	108.15	1.92105	0	0
29	00111111011111110001000001	86.4	1.78667	0	0
30	00111111011111101011000001	101	1.88824	0	0
31	00111101011111111010000001	87.95	1.8	0	0

Библиографический список

1. Holland J. H. Adaptation in natural and artificial systems / J. H. Holland. – Ann Arbor. MI: University of Michigan Press, 1975.
2. Goldberg D. E. Genetic algorithms in search, optimization, and machine learning / D. E. Goldberg. – Reading, MA: Addison-Wesley, 1989.
3. Семенкин Е.С., Семенкина О.Э., Коробейников С.П. Оптимизация технических систем: Учебное пособие / Е.С. Семенкин, О.Э. Семенкина, С.П. Коробейников. – Красноярск: СИБУП, 1996.
4. Michalewicz Z. Genetic algorithms, numerical optimization and constraints // Proc. of the Sixth Int. Conf. on Genetic Algorithms and their Applications, Pittsburgh, PA, 1995.
5. Parmee I. (Ed.) Adaptive computing in engineering design and control. Proceedings of the International Conference / I. Parmee. – Plymouth, 1996. – 325 pp.
6. Goodman E. et al (Eds). Evolutionary computation and its applications. Proceedings of the International Conference / E. Goodman. – Moscow: IHPCS of RAS, 1996. –350 pp.
7. Семенкин Е.С., Лебедев В.А. Метод обобщенного адаптивного поиска для синтеза систем управления сложными объектами. – М.: МАКС Пресс, 2002. – 320 с.
8. Семенкин Е.С., Семенкина О.Э., Коробейников С.П. Адаптивные поисковые методы оптимизации сложных систем–Красноярск: СИБУП, 1997.–355 с.
9. Семенкин Е.С., Об эволюционных алгоритмах решения сложных задач оптимизации / Гуменникова А.В., Емельянова М.Н., Семенкин Е.С., Сопов Е.А // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева: Сб. научн. трудов.– Красноярск: СибГАУ, 2003.– С. 14–23.
10. Жукова М.Н. Коэволюционный алгоритм решения сложных задач оптимизации – диссертация на соискание ученой степени кандидата наук, Красноярск: СибГАУ, 2004. – 124 с.
11. Goldberg, D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, Massachusetts: Addison-Wesley, 1989.
12. Branke J. Memory Enchanced Evolutionary Algorithms for Changing Optimization Problems, Institute AIFB, University of Karlsruhe, 1999.
13. Юдин Д.Б. Вычислительные методы теории принятия решений. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 320 с. (Теория и методы системного анализа.)
14. Baker J. Adaptive selection methods for genetic algorithms. Proc. International Conf. on Genetic Algorithms and Their Applications. J. Grefenstette, ed. Lawrence Erlbaum, 1985.
15. Baker J. Reducing Bias and Inefficieny in the Selection Algorithm. Genetic Algorithms and Their Applications: Proc. Second International Conf. J. Grefenstette, ed. Lawrence Erlbaum, 1987.

16. Bentley P.J., Wakefield J.P. Finding Acceptable Solutions in the Pareto-Optimal Range using Multiobjective Genetic Algorithms. In Proceedings of the 2nd On-Line World Conference on Soft Computing in Engineering Design and Manufacturing, 1997.

17. Cieniawski S. E. An investigation of the ability of genetic algorithms to generate the tradeoff curse of a multi-objective groundwater monitoring problem. Master's thesis. University of Illinois at Urbana-Champaign. 1993.

18. Coello Coello Carlos A. An empirical study of evolutionary techniques for multiobjective optimization in engineering design. PhD thesis. Department of computer science, Tulane university. New Orleans, LA, apr 1996.

19. Coello Coello C.A. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Laboratorio Nacional de Informatica Avanzada, Veracruz, Mexico, 1998.

20. Deb K. Multi-objective Optimization using Evolutionary Algorithms. Chichester, UK: Wiley, 2001.

21. Deb K., Pratap A., Agarwal S., Meyarivan T. A Fast and Elitist Multi-objective Genetic Algorithm: NSGA II. KanGAL Report No. 200001. Indian Institute of Technology, Kanpur, India, 2000.

22. Fonseca C.M., Fleming P.J. Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: A unified formulation. Technical report 564, University of Sheffield, Sheffield, UK, January 1995.

23. Horn J., Nafpliotis N., Goldberg D. E. A niched Pareto genetic algorithm for multiobjective optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation, Vol. 1, Piscataway, 1994. – P. 82-87.

24. Knowles J., Corne D. The Pareto archived evolution strategy: A new baseline algorithm for multiobjective optimization. Proceedings of the 1999 Congress on Evolutionary Computation, Piscataway, New Jersey: IEEE Service Center, 1999, pp. 98-105.

25. Koski J., Oscyczka A. Multi-criteria Design Optimization. Springer-Verlag, 1990.

26. Srinivas N., Deb K. Multiple-Objective function optimization using non-dominated sorting genetic algorithms. Evolutionary Computation, Vol. 2, pp. 221-248, 1995.

27. Сопов Е.А. Вероятностный генетический алгоритм и его исследование // VII Королевские чтения. Том 5. - Самара: Изд-во Самарского научного центра РАН, 2003. – С. 38-39.

28. Горбань, А. Н. Нейронные сети на персональном компьютере [Текст] / А.Н. Горбань, Д. А. Россиев. – Новосибирск : Наука, 1996. – 276 с.

29. Круглов, В. В. Искусственные нейронные сети. Теория и практика [Текст] / В. В. Круглов, В. В. Борисов. – М. : Горячая линия – Телеком, 2002. – 139 с.

30. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы [Текст] : Пер. с польск. / Д. Рутковская, М. Пилиньский, Л. Рутковский. М. : Горячая линия – Телеком, 2004. – 452 с.

31. Киселев В.В. Обучение в системах нечеткого управления. / Математические структуры и моделирование, вып.6, 2000. – с.78-90.

32. Тихонов А.Н., Цветков В.Я. Методы и системы поддержки принятия решений. / М.: МАКС Пресс, 2001. - 312 с.

33. Асаи К, Прикладные нечеткие системы: Пер. с япон. / К. Асаи, Д. Ватада, С. Иваи // М.: Мир, 1993. – 368 с., ил.

34. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. «Питер», 2001. 384 с.

35. Бежитский, С.С. Гибридный эволюционный алгоритм для задач выбора эффективных вариантов систем управления / С.С. Бежитский, Е.С. Семенкин, О.Э. Семенкина // Автоматизация и современные технологии. – № 11. – 2005. – С. 24-31.

36. Усков В.В., Сравнительный анализ эффективности методов комбинаторной оптимизации на примере задачи коммивояжера – диссертация на соискание степени магистра, Красноярск, СибГАУ, 2003. – 35 с.

37. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация: Алгоритмы и сложность. М.: Мир, 1985. 512 с.

38. Жукова, М.Н. Коэволюционный алгоритм решения сложных задач оптимизации : дис. канд. тех. наук: 05.13.01: защищена 16.12.04 / Жукова Марина Николаевна. – Красноярск, 2004.

39. Жуков, В.Г. Моделирование сложных систем коэволюционным алгоритмом генетического программирования : дис. канд. тех. наук: 05.13.01: защищена 26.12.06 / Жуков Вадим Геннадьевич. – Красноярск, 2006.

40. Семенкин Е.С., Об эволюционных алгоритмах решения сложных задач оптимизации / Гуменникова А.В., Емельянова М.Н., Семенкин Е.С., Сопов Е.А // Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева: Сб. научн. трудов.– Красноярск: СибГАУ, 2003.– С. 14–23.

41. Eberhart, R. C, and Kennedy, J.. (1995). A new optimizer using particle swarm theory. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, 39-43. Piscataway, NJ: IEEE Service Center.

42. Kennedy, J., and Eberhart, R. C. (1997). A discrete binary version of the particle swarm algorithm. Proc. 1997 Conf. on Systems, Man, and Cybernetics, 4104-4109. Piscataway, NJ: IEEE Service Center.

43. Антамошкин А.Н. Оптимизация функционалов с булевыми переменными. – Томск: Изд-во Том. ун-та, 1987, - 104 с.

44. Панфилов И.А. Модели и алгоритмы выбора эффективной конфигурации многопроцессорных систем обработки информации и управления - диссертация на соискание ученой степени к.т.н., СибГАУ, Красноярск, 2006 г.

45. Сопов Е.А. Эволюционные алгоритмы моделирования и оптимизации сложных систем - диссертация на соискание ученой степени к.т.н.., СибГАУ, Красноярск, 2004 г.

46. Липинский Л.В. Алгоритмы генетического программирования для формирования интеллектуальных информационных технологий - диссертация на соискание ученой степени к.т.н., СибГАУ, Красноярск, 2006 г.

47. Бочарников А.В. Разработка и исследование модифицированного ГА для решения задач оптимизации – диссертация на соискание степени магистра, СибГАУ, Красноярск, 2004 г.

48. Гуменникова А.В. Адаптивные поисковые алгоритмы для решения сложных задач много-критериальной оптимизации - диссертация на соискание ученой степени к.т.н., СибГАУ, Красноярск, 2006 г.