11.2 ПРОСТЫЕ ТЕСТОВЫЕ ЗАДАЧИ[1]

Представленные ниже задачи являются широко распространенными и известными, некоторые из них тривиальные, соответствующие функции приспособленности/качества подходят для небольших экспериментов и проектов. Задачи представлены для поиска булевского или вещественного вектора фиксированной длины, многокритериальных сценариев и генетического программирования (и эволюции грамматик).

Многие из этих задач используются уже давно и несколько устарели, поэтому если вы работаете над научной исследовательской статьей, вам необходимо потратить некоторое время на поиск и изучение тестовых задач, применяемых в рассматриваемой области. Кроме этого, если вы используете тестовые задачи для сравнения качества результатов работы различных алгоритмов, то помните о соблазне необъективных сравнений, когда формируется такой набор тестовых задач, при котором ваш алгоритм выглядит лучше. Конечно, этот набор задач всегда можно найти, но какая с этого выгода? Будет намного полезнее попытаться понять, как ваш алгоритм работает на широком спектре хорошо понимаемых задач, описанных в литературе, либо на задачах, представляющих существенный интерес для научного сообщества или его части[2].

11.2.1 ЗАДАЧИ ДЛЯ БУЛЕВСКИХ ВЕКТОРОВ

Задача «Max Ones» («Максимизация единичных разрядов»). Задача «Max Ones», иногда называемая «OneMax», представляет тривиальную булевскую задачу: необходимо максимизировать количество единичных разрядов в имеющемся векторе. Данная задача – классический пример линейной проблемы, в которой отсутствует какая-либо связь между значениями вектора. Простейшие методы локального спуска легко решают эту задачу. Ее формулировка (по Дэвиду Экли)[3]:

$$f(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} x_i$$

Задача «Leading Ones» («Начальные единичные разряды»). Данная задача также весьма проста: необходимо максимизировать количество подряд идущих единичных разрядов, если считать с начала вектора. Другими словами наибольшую роль играет позиция первого нулевого разряда в векторе. Приведенное ниже уравнение представляет математическое

---

[1] Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 0.2. September, 2009 (http://cs.gmu.edu/~sean/book/metaheuristics/). Перевел – Юрий Цой, 2009 г.
Любые замечания, касающиеся перевода, просьба присылать по адресу yurytsoy@gmail.com
Данный текст доступен по адресу: http://qai.narod.ru/GA/meta-heuristics_11_2.pdf

[2] Здесь необходимо вспомнить знаменитую теорему об отсутствии бесплатных обедов (No Free Lunch Theorem, NFL), доказанную Дэвидом Уолпертом и Вильямом Макреди. Теорема гласит, что при определенных условиях, при рассмотрении всех возможных задач, любой алгоритм оптимизации в среднем не превосходит любой другой алгоритм. Другими словами, если существует набор задач P, для которого алгоритм A будет лучше алгоритма B, то существует также набор задач P', для которого верно обратное. Эта теорема представляет больше теоретический интерес, но, на мой взгляд, имеет ограниченную практическую ценность, потому что пространство всех возможных задач включает в том числе и самые «необычные» и «причудливые» задачи, которые вряд ли можно встретить в жизни. По моему мнению, больший интерес представляет то, какие алгоритмы работают лучше на типичных задачах для той или иной проблемной области и почему. Более подробную информацию о NFL-теореме можно найти в статье Wolpert David., Macready W., No free lunch theorems for optimization //
IEEE Transactions on Evolutionary Computation, 1997, vol. 1, no. 1, pp. 67–82.

[3] Ackley D. A Connectionist Machine for Genetic Hillclimbing. – Kluwer Academic Publishers, 1987.

описание этой задач, однако оно плохо подходит для программной реализации в силу вычислительной сложности. Нужно просто посчитать количеств единичных разрядов в векторе пока не будет встречен нулевой. Задача «Leading Ones» – нелинейная: вклад («полезность») компонента x_i существенно зависит значений компонент x_1, ..., x_{i-1}. Тем не менее, эта задача легко решается.

$$f(\langle x_1, ..., x_n \rangle) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

Задача «Leading Ones Blocks» («Начальные единичные блоки»). Представляет собой усложненную модификацию предыдущей задачи. Для данного числа b необходимо подсчитать количество единичных строк, длиной b каждая, до тех пор, пока не встретится нулевой разряд. Например, при b = 3, имеем f (<1, 1, 0, 0, 0, 1, 1, 0, 1>)= 0 поскольку в векторе в начале не содержится строка из 3 единиц. Однако f (<1, 1, 1, 0, 0, 0, 1, 0, 1>) = 1. Далее f (<1, 1, 1, 1, 0, 1, 1, 0, 1>)= 1, но f (<1, 1, 1, 1, 1, 1, 0, 1, 0>) = 2 и, наконец, f (<1, 1, 1, 1, 1, 1, 1, 1, 1>) = 3. Простой способ математической записи целевой функции для данной задачи: использовать функцию для «Leading Ones», поделить результат на b и округлить до ближайшего целого снизу:

$$f(\langle x_1, ..., x_n \rangle) = \left\lfloor \frac{1}{b} \sum_{i=1}^{n} \prod_{j=1}^{i} x_j \right\rfloor$$

Задача «Trap» («Ловушка»). Так называемые задачи-ловушки представляют классический пример обманчивых функций (*deceptive functions*). Вот простой пример: приспособленность вектора равна количеству нулевых разрядов, если в векторе присутствует хотя бы один единичный разряд, однако если все разряды – единичные, то приспособленность внезапно повышается до значения n+1 (n – длина вектора). Таким образом, в данной задаче задается направление поиска, уводящее от оптимального решения («нет нулевых разрядов»), прямо в «ловушку». К примеру, f (<0, 0, 0, 0>)= 4, f (<0, 0, 1, 0>)= 3, f (<1, 0, 1, 0>)= 2, f (<1, 0, 1, 1>)= 1, но, бац, f (<1, 1, 1, 1>) = 5. Математическая формулировка задачи включает два слагаемых: первое равно количеству нулевых разрядов, а второе появляется только когда все разряды – единичные. Различные функции ловушки были изначально придуманы Дэвидом Экли[4].

$$f(\langle x_1, ..., x_n \rangle) = \left(n - \sum_{i=1}^{n} x_i \right) + (n+1) \prod_{i=1}^{n} x_i$$

11.2.2 ЗАДАЧИ ДЛЯ ВЕЩЕСТВЕННЫХ ВЕКТОРОВ

Многие классические задачи для вещественных векторов являются задачами минимизации, а не максимизации. Чтобы преобразовать их к задаче максимизации в простейшем случае нужно изменить знак результата. Однако если используется пропорциональная селекция или стохастическая однородная селекция (*Stochastic Universal Selection*, *SUS*), необходимо дополнительно прибавить достаточно большое число, чтобы избежать появления отрицательных значений. Хотя я вместо этого просто использовал бы турнирную селекцию.

Для большинства задач на рис. 66 показаны примеры для простейшего двумерного случая.

[4] См. сноску на предыдущей странице для задачи «Max Ones».

Задача «Sum» («Сумма»). Данная задача представляет тривиальную версию задачи «Max Ones» только для вещественных чисел. Рассматривается просто сумма компонентов вектора. Как и следует ожидать, задача «Сумма» – линейная и в ней нет связанности (*linkage*).

$$f(\langle x_1, ..., x_n \rangle) = \sum_{i=1}^{n} x_i \qquad x_i \in [0.0, 1.0]$$

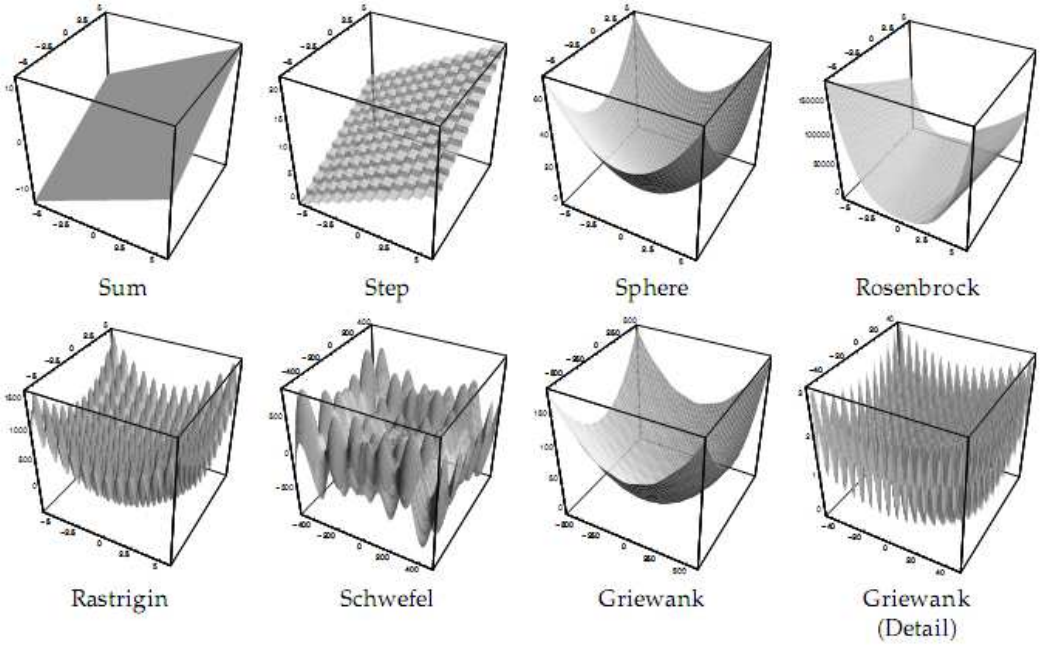


Рис. 66. Задачи для вещественных чисел в двумерном пространстве (<x_1, x_2>)

Задача «Linear» («Линейная сумма»). Линейные функции представляют обобщение задачи «Сумма» и в них также отсутствует связанность. Данные функции описываются взвешенной суммой компонентов вектора, где вес каждой компоненты задан в виде константы a_i. Имея вектор таких констант <a_0, …, a_n>, каждый элемент умножается на соответствующий вес и прибавляется к сумме:

$$f(\langle x_1, ..., x_n \rangle) = a_0 + \sum_{i=1}^{n} a_i x_i \qquad x_i \in [0.0, 1.0]$$

Задача «Step» («Шаговая»). Еще одна функция без связанности, но на этот раз негладкая из-за использования округления. Благодаря этому появляются небольшие области, незначительные мутации в которых не приводят к изменению значения функции приспособленности. Данная функция входит в популярный набор тестовых задач Де Йонга[5], отсюда и традиционные ограничения для значений x_i (от -5,12 до +5,12 включая концы).

[5] Возможно, слишком популярный набор. Кен Де Йонг в свое время развернул активную компанию, чтобы прекратить его использование! Тестовый набор был предложен в диссертации Де Йонга: De Jong K. An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. Ph.D. thesis. University of Michigan, 1975. Текст диссертации доступен для скачивания по адресу: http://cs.gmu.edu/!eclab/kdj thesis.html

Обычно требуется минимизировать значение функции, однако это не обязательно: можно также искать и максимум, свойства функции от этого не изменятся.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor \qquad x_i \in [-5.12, 5.12]$$

Задача «Sphere» («Сферическая»). Наша последняя задача без связанности придумана Инго Рехенбергом[6]. В ней производится суммирование квадратов компонентов вектора. Это снова задача минимизации и она входит в тестовый набор Де Йонга (обратите внимание на границы значений переменных). Задача максимизации также представляет интерес, т.к. глобальные максимумы расположены по углам области определения.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = \sum_{i=1}^{n} x_i^2 \qquad x_i \in [-5.12, 5.12]$$

Функция Розенброка. Классическая задача оптимизации с интересными особенностями, разработанная Говардом Розенброком[7]. В случае двух измерений функция создает небольшое плато вокруг низкого холма, с большими крыльями по обеим сторонам. Точка минимума <1, 1, …, 1> находится на одной стороне холма, в то время как особи часто «застревают» на другой стороне. Также как и предыдущие задачи входит в набор Де Йонга, с соответствующими ограничениями на значения переменных. Является задачей минимизации.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = \sum_{i=1}^{n-1} (1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \qquad x_i \in [-5.12, 5.12]$$

Функция Растригина. Предложена Леонардом Андреевичем Растригиным[8] в 1974 как двухмерная функция, но позднее была расширена Хайнцем Мюленбайном (*Heinz Muhlenbein*), М. Шомишем (*M. Schomisch*) и Иоахимом Борном (*Joachim Born*) для случая большего количества переменных[9]. График функции имеет вид картонной коробки для яиц, в которую положили баскетбольный мяч: функция является комбинацией сферической функции и синусоиды, которая создает множество локальных оптимумов. Является частью набора Де Йонга, отсюда и традиционные границы для переменных. Представляет задачу минимизации.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = 10n \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) \qquad x_i \in [-5.12, 5.12]$$

Функция Швефеля. Разработана Ханс-Паулем Швефелем[10] и также, как и функция Растригина, имеет множество локальных оптимумов. Однако в ней оптимумы располагаются ближе друг к другу (поэтому между ними легко «прыгать») по мере удаления от глобального оптимума. И снова является задачей минимизации. Отметим большие границы значений переменных, чем встреченные ранее.

---

[6] Rechenberg I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart, Germany: Fromman-Holzbook, 1973.

[7] Rosenbrock H. An automatic method for finding the greatest or least value of a function // The Computer Journal, 1960, vol. 3, no. 3, pp. 174–184.

[8] Я полагаю, что функция была представлена в Растригин Л.А. Системы экстремального управления. М.: Наука, 1974.Однако в силу того, что книгу практически невозможно достать и проверить, не забивайте голову.

[9] Muhlenbein H., Schomisch D., Born J. The parallel genetic algorithm as function optimizer // Belew R., Booker L. (eds.): Proceedings of the Fourth International Conference on Genetic Algorithms, 1991, pp. 271–278.

[10] Schwefel H.-P. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkhauser, 1977.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right) \qquad x_i \in [-500, 500]$$

Функция Гривонка. Схожая с функцией Растригина функция Андреаса Гривонка имеет триллион локальных оптимумов[11]. Функцию необходимо минимизировать и традиционно значения переменных изменяются от -600 до +600, что создает огромное число локальных оптимумов.

$$\text{(Minimize)} \qquad f(\langle x_1, ..., x_n \rangle) = 1 + \frac{1}{4000}\left(\sum_{i=1}^{n} x_i^2\right) + \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \qquad x_i \in [-600, 600]$$

11.2.3 МНОГОКРИТЕРИАЛЬНЫЕ ЗАДАЧИ

Описываемые ниже задачи взяты из классического сравнения многокритериальных алгоритмов оптимизации, описанного в статье Эккарта Цитцлера (*Eckart Zitzler*), Кальянмоя Деба (*Kalyanmoy Deb*) и Лотара Тиеле (*Lothar Thiele*)[12]. Как и во многих многокритериальных тестовых задачах в этих задачах рассматривается минимизация целевой функции. Можете изменить их на задачи максимизации, например, изменением знака. Во всех четырех задачах имеется два критерия O_1 и O_2. Все задачи сконструированы таким образом, что O_2 представляет собой функцию от двух дополнительных функций g и h. Глобальные границы Парето для всех четырех задач и сильный локальный фронт Парето для одного случая приведены на рис. 67.

Задача ZDT1. Представляет базовую многокритериальную задачу с выпуклой границей Парето для вещественного вектора особи с n = 30 генами. Не имеет локальных оптимумов.

$$\begin{aligned}
\text{(Minimize)} \quad & O_1(\langle x_1, ..., x_n \rangle) = x_1 \qquad\qquad x_i \in [-1, 1] \\
& O_2(\langle x_1, ..., x_n \rangle) = g(\langle x_1, ..., x_n \rangle) \times h(\langle x_1, ..., x_n \rangle) \\
& g(\langle x_1, ..., x_n \rangle) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
& h(\langle x_1, ..., x_n \rangle) = 1 - \sqrt{\frac{x_1}{g(\langle x_1, ..., x_n \rangle)}}
\end{aligned}$$

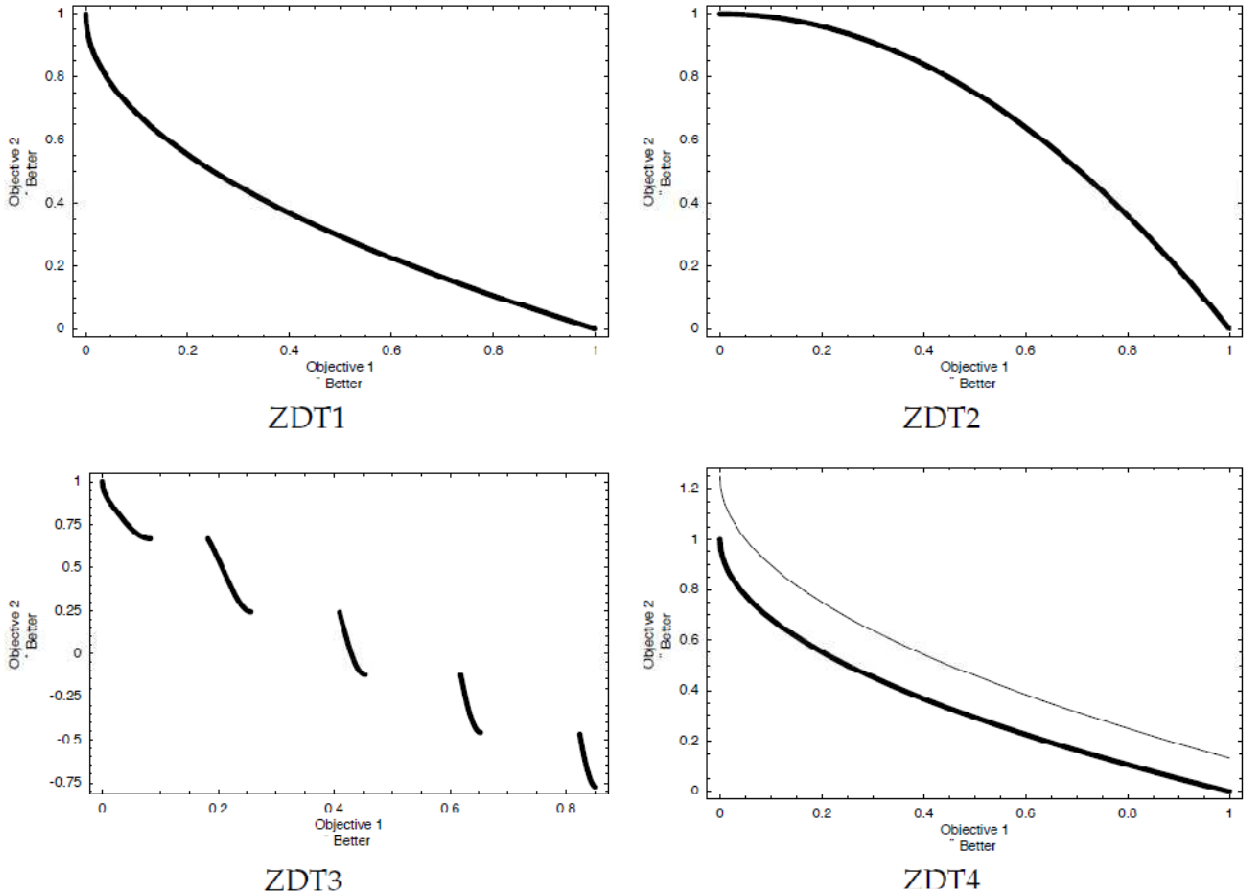
Задача ZDT2. Функция похожа на ZDT1, только «вогнутая». Снова n = 30. Не имеет локальных оптимумов.

$$\begin{aligned}
\text{(Minimize)} \quad & O_1(\langle x_1, ..., x_n \rangle) = x_1 \qquad\qquad x_i \in [-1, 1] \\
& O_2(\langle x_1, ..., x_n \rangle) = g(\langle x_1, ..., x_n \rangle) \times h(\langle x_1, ..., x_n \rangle) \\
& g(\langle x_1, ..., x_n \rangle) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i \\
& h(\langle x_1, ..., x_n \rangle) = 1 - \left(\frac{x_1}{g(\langle x_1, ..., x_n \rangle)}\right)^2
\end{aligned}$$

---

[11] Griewank A. Generalized descent for global optimization // Journal of Optimization Theory and Applications, 1991, no. 34, pp. 11–39.

[12] Zitzler E., Deb K., Thiele L. Comparison of multiobjective evolutionary algorithms: Empirical results // Evolutionary Computation, 2000, vol. 8, no. 2, pp. 125–148.



Граница Парето имеет разрывы

Тонкая линия обозначает наилучшую локальную субоптимальную границу Парето. Другие субоптимальные границы не показаны.

Рис. 67. Границы Парето для четырех многокритериальных задач (ZDT1, ZDT2, ZDT3 и ZDT4), описанных в разделе 11.2.3. Все задачи представляют собой задачи минимизации, поэтому меньшие значения критериев являются более предпочтительными.

Задача ZDT3. Данная функция имеет границу Парето с разрывами. Снова n = 30. Не имеет локальных оптимумов.

$$
\begin{aligned}
&\text{(Minimize)} \quad O_1(\langle x_1, \ldots, x_n \rangle) = x_1 \qquad\qquad x_i \in [-1, 1] \\
&\qquad O_2(\langle x_1, \ldots, x_n \rangle) = g(\langle x_1, \ldots, x_n \rangle) \times h(\langle x_1, \ldots, x_n \rangle) \\
&\qquad g(\langle x_1, \ldots, x_n \rangle) = 1 + \frac{9}{n-1} \sum_{i=2}^{n} x_i \\
&\qquad h(\langle x_1, \ldots, x_n \rangle) = 1 - \sqrt{\frac{x_1}{g(\langle x_1, \ldots, x_n \rangle)}} - \frac{x_1}{g(\langle x_1, \ldots, x_n \rangle)} \sin(10\pi x_1)
\end{aligned}
$$

Задача ZDT3. Функция имеет выпуклую границу Парето, но со многими локальными субоптимальными границами, в которых могут «застревать» особи, что делает задачу средней по сложности. Задача определена для меньшего значения n, чем предыдущие: n = 10. Значения переменной x_1 изменяются в интервале [0, 1], а значения других переменных x_i – в интервале [-5, 5].

$$\text{(Minimize)} \quad O_1(\langle x_1, ..., x_n \rangle) = x_1 \qquad x_1 \in [0,1], x_{i>1} \in [-5,5]$$
$$O_2(\langle x_1, ..., x_n \rangle) = g(\langle x_1, ..., x_n \rangle) \times h(\langle x_1, ..., x_n \rangle)$$
$$g(\langle x_1, ..., x_n \rangle) = 1 + 10(n-1) + \sum_{i=2}^{n} x_i^2 - 10\cos(4\pi x_i)$$
$$h(\langle x_1, ..., x_n \rangle) = 1 - \sqrt{\frac{x_1}{g(\langle x_1, ..., x_n \rangle)}}$$

11.2.4 ЗАДАЧИ ДЛЯ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Поскольку задачи генетического программирования направлены на оптимизацию небольших компьютерных программ, они в некотором смысле более яркие и детальные, чем описанные до этого математические функции. Приведенные здесь задачи не очень сложные: их часто удается решить с популяцией из 1000 или около того особей за 51 поколение (включая начальное). Приведенные проблемы разработаны Джоном Козой[13].

Символьная регрессия. Представляет каноническую задачу генетического программирования и, возможно, используется слишком часто. Целью является поиск математического выражения, которое наилучшим образом аппроксимирует набор точек данных вида <x, $f(x)$> некоторой неизвестной (для алгоритма оптимизации) функции f. Традиционно в качестве аппроксимируемой используется функция $f(x) = x^4 + x^3 + x^2 + x$, хотя Коза также предлагал функции $g(x) = x^5 - 2x^3 + x$ и $h(x) = x^6 - 2x^4 + x^2$. Графики этих функций приведены на рис. 68.

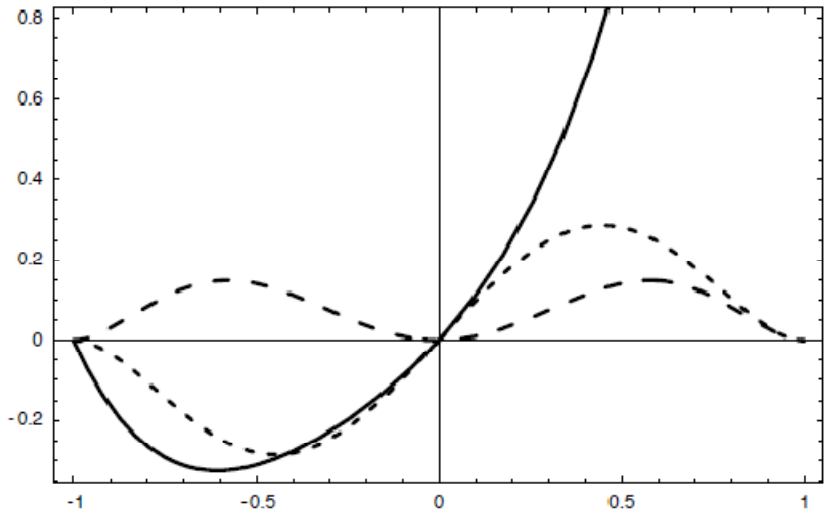


Рис. 68. Графики функций $f(x)$: сплошная линия, $g(x)$: линия с крупным пунктиром, $h(x)$: линия с мелким пунктиром описанных в разделе Символьная регрессия. $f(1) = 4$.

Сначала создаются 20 случайных значений $x_1,\ldots, x_{20}$ из интервала от -1 до 1 каждое, которые затем используются на протяжении всего запуска. Оценка особи производится следующим образом: для каждого из 20 значений x_i в листьях функция x возвращает значение x_i, после чего производится оценка дерева особи. Предположим, что в результате вычислений в дереве получили y_i. Приспособленность особи показывает насколько полученные 20

[13] Приведено с изменениями по Koza J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992 и по Koza J.R. Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press, 1994.

значений y_i соответствуют ожидаемым $f(x_i)$, как правило, с использованием простых мер близости. Таким образом, приспособленность равна $\sum_{i=1}^{20} | y_i - f(x_i) |$.

Очевидно, что имеем дело с задачей минимизации. Она легко может быть преобразована к задаче максимизации: $\frac{1}{1+ fitness}$. Пример идеального решения `(+ (* x (* (+ x (* x x)) x)) (* (+ x (cos (- x x))) x))`.

Табл. 5. Набор функций для задачи символьной регрессии

Function	Arity	Description
$(+\ i\ j)$	2	Returns $i + j$
$(-\ i\ j)$	2	Returns $i - j$
$(*\ i\ j)$	2	Returns ij
$(\%\ i\ j)$	2	If j is 0, returns 1, else returns i/j
$(\sin\ i)$	1	Returns $\sin(i)$
$(\cos\ i)$	1	Returns $\cos(i)$
$(\exp\ i)$	1	Returns e^i
$(\mathrm{rlog}\ i)$	1	If j is 0, returns 0, else returns $\log(\lvert i \rvert)$
x	0	Returns the value of the independent variable (x).
ERCs	0	(*Optional*) Ephemeral random constants chosen from floating-point values from -1 to 1 inclusive.

11-битный булевский мультиплексор. Целью данной задачи является поиск булевской функции, которая производит мультиплексирование с использованием 3-битного адреса. Есть три булевские адресные переменные (A0, A1, и A2) и восемь соответствующих булевских переменных для данных (D0, D1, D2, D3, D4, D5, D6, D7). Искомый мультиплексор возвращает значение переменной данных по адресу, представленному значениями переменных A0, A1, и A2. Например, если A2 ложно, а A1 и A0 истинны, то адрес равен 3 (011), и оптимальная особь должна вернуть значение, записанное в D3. Поскольку все 11 переменных рассматриваются вместе, то всего существует 2048 комбинаций этих переменных и следовательно 2048 тестовых случая. Более простой 6-битный булевский мультиплексор имеет две адресные переменные (A0 и A1), четыре переменных данных (D0, D1, D2, D3) и 64 тестовых случая.

Особь, представляющая мультиплексор, включает одно дерево. Для оценки приспособленности для всех тестовых случаев в адресные переменные и переменные данных записывают значения, соответствующие рассматриваемому случаю, и затем оценивается дерево. Приспособленность определяется как число верных ответов особи для различных наборов значений переменных адреса и данных.

Пример идеального решения для задачи 11-битного булевского мультиплексора:

```
(if (not a0) (if (not a0) (if (not a1) (if a2 (if a2 d4 d6) d0)
(if a2 d6 (if a2 d4 d2))) (if (or a2 a2) (if a1 (or (if (not (if
a2 d5 d0)) (and (and d4 d0) (and a2 d5)) (or (and d7 d0) (not
a1))) (if (not a1) (if (if d4 d1 d5) d0 d5) (or d6 (or (and (and
d4 d0) (or (and d5 d1) (and d6 d6))) (and d7 (or (if a0 (or a2 a2)
```

```
d4) (and d1 (and d5 a2)))))))) d5) (if a1 (or d3 (and d7 d0)) (if
a0 d1 d0)))) (if (or a2 a2) (if a1 (if (not a1) (if (and d7 d0)
(if a2 d5 d0) (if a2 d6 d3)) (and d7 (or (if a0 a2 (or d1 a1))
(not a1)))) d5) (if a1 (or (if (not a0) (if a2 d6 (if a2 d4 d2))
(if a1 d3 (or (or d3 (if a1 d3 d1)) (not a2)))) (not a1)) (if a0
d1 d0))))
```

Табл. 6. Набор функций для 11-битного булевского мультиплексора

Function	Arity	Description
(and *i j*)	2	Returns $i \cap j$
(or *i j*)	2	Returns $i \cup j$
(not *i*)	1	Returns $\neg i$
(if *test then else*)	3	If *test* is true, then *then* is returned, else *else* is returned.
a0, a1, *and* a2	0	Return the values of variables A0, A1, and A2 respectively.
d0, d1, d2, d3, d4, d5, d6, *and* d7	0	Return the values of variables D0, D1, D2, D3, D4, D5, D6, and D7 respectively.

N-четность. Задачи N-четности, также как и 11-битный булевский мультиплексор, являются булевскими задачами для *n* переменных. В этих задачах необходимо вернуть «Истина», если текущая булевская строка содержит четное количество переменных со значением «Истина». Таким образом всего может быть 2^n тестовых случаев. Вычисление приспособленности производится аналогично задаче 11-битного булевского мультиплексора.

В зависимости от значения N меняется сложность задачи N-четности, в силу изменения количества тестовых случаев. Билл Лэнгдон отмечает, что при решении задачи на четность строительные блоки отсутствуют[14]. Решение для задачи 4-четности:

```
(nand (or (or (nor d3 d0) (nand (or d3 d1) (nor d2 d3))) d3) (nor
(nor (and (or (and (or (or (nor d1 d2) (and d3 d0)) (and d1 d2))
(nand (and d0 d3) (nand (or d0 d1) (or d2 d1)))) (and (or d0 d2)
(and d1 d1))) (nand (and (nor d3 d0) (and (and (nand (nand (nor d3
d3) (or (or d0 d0) (nor (and d3 d0) (nor d1 (nand d3 d2))))) d2)
(nor d1 d1)) (or (or d0 d1) (nor d3 d2)))) (nand (or d0 d1) (nor
d3 d3)))) (or (and (nand d1 d1) (and d1 d3)) (nor (nand (or d1 d2)
(nor d3 d0)) d0))) (and (or (or (or (and (nand d1 d1) (and d1 d3))
(nor (nand (or d1 d2) (nor d3 d0)) (and (nand d1 d3) (and d3
d0)))) (and d3 d0)) (and d3 d2)) (and (and d1 d2) (or (or d0 (nor
(or d0 d0) (and d2 d3))) d0)))))
```

[14] Langdon W. Scaling of program tree fitness spaces, 1999, Evolutionary Computation, vol. 7, no. 4, pp. 399–428.

Табл. 7. Набор функций для N-четности

Function	Arity	Description
(and i j)	2	Returns $i \cap j$
(or i j)	2	Returns $i \cup j$
(nand i j)	2	Returns $\neg(i \cap j)$
(nor i j)	2	Returns $\neg(i \cup j)$
d0, d1, d2, *etc.*	0	Return the values of variables D0, D1, D2, ... respectively. The number of d*x* nodes in the function set is the number of bits in the particular Parity problem being run.

Искусственный муравей (Artificial Ant). Представляет чрезвычайно сложную задачу[15] для генетического программирования. В данной задаче необходимо найти простой алгоритм управления роботом-муравьем, который находит и съедает наибольшее количество пищи за 400 тактов модельного времени[16]. Муравей может двигаться вперед, поворачивать налево и направо. Если во время движения он проходит через ячейку с пищей, то он съедает ее. Муравей также может «почуять», есть ли в клетке перед ним пища. Решетка с миром, в котором живет искусственный муравей показана на рис. 69. След ячеек с пищей известен как «След Санта Фе». Мир тороидальный: выход за границы приводит к появлению муравья с противоположной стороны.

Начальная позиция. Муравей повернут направо.

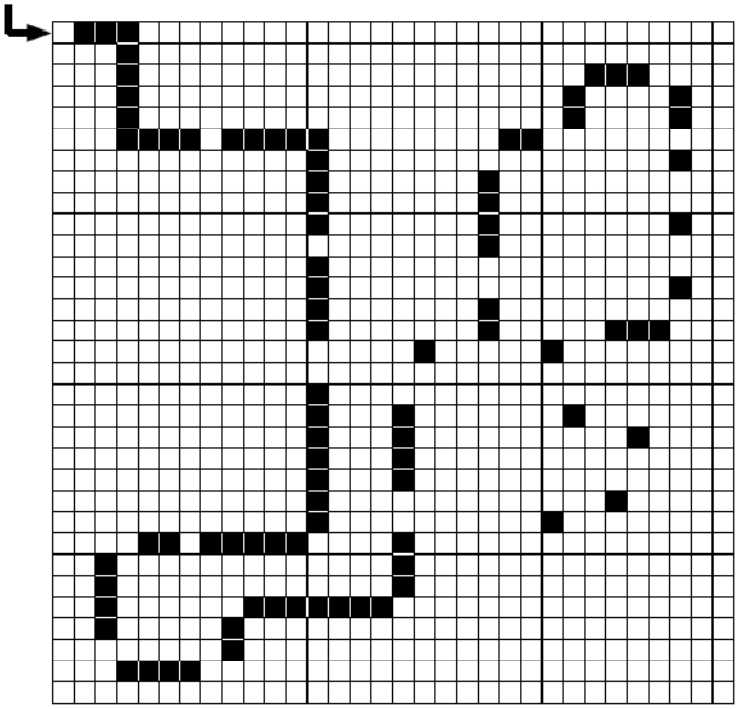


Рис. 69. След Санта Фе, тороидальный клеточный мир. Черные квадраты обозначают ячейки с пищей.

[15] Одна из моих самых любимых статей, в основном благодаря кнутовскому вниманию к деталям, как раз посвящена этому вопросу: Langdon W. B., Poli R. Why ants are hard // In Genetic Programming 1998: Proceedings of the Third Annual Conference, Morgan Kaufmann, 1998, pp. 193–201.

[16] Возможно 400 – это опечатка, которая, тем не менее, закрепилась. Джон Коза считает, что должно быть 600.

Особь искусственного муравья состоит из одного дерева. Оценка приспособленности производится следующим образом. Муравей начинает путь с левого верхнего угла и повернут направо. Затем начинается исполнение дерева: при активации узла движения или сенсора, муравей соответственно двигается или чувствует. Когда исполнение дерева заканчивается, оно начинает исполняться с самого начала. Каждое движение занимает 1 такт времени. Оценка заканчивается, когда муравей съел всю пищу в мире, либо когда закончилось 400 шагов. Приспособленность равна количеству съеденной пищи.

Задача искусственного муравья отличается от задачи символьной регрессии и булевских задач тем, что сами значения на выходах узлов дерева игнорируются. Единственно важным является то, как влияет действие узла на сам мир, т.е. побочный эффект от активации узла: движение муравья, поворот и т.д. Это означает, что для искусственного муравья порядок исполнения узлов определяет действия особи, в то время как для предыдущих задач очередность исполнения узлов дерева не играла особой роли. Пример решения (очень компактного) для задачи искусственного муравья: `(progn3 (if-food-ahead move (progn2 left (progn2 (progn3 right right right) (if-food-ahead move right)))) move right)`.

Табл. 8. Набор функций для задачи искусственного муравья

Function	Arity	Description
(progn3 *a b c*)	3	*a*, *b*, then *c* are executed.
(progn2 *a b*)	2	*a*, then *b* are executed.
(if-food-ahead *then else*)	2	If food is immediately in front of the ant, *then* is executed, else *else* is executed.
move	0	Moves the ant forward one square, eating food if it is there.
left	0	Rotates the ant ninety degrees to the left.
right	0	Rotates the ant ninety degrees to the right.

Газонокосилка (Lawnmower). В данной задаче особь направляет газонокосилку в тороидальном мире, аналогичном миру муравья в задаче искусственный муравей. В задаче о газонокосилке особь может повернуть налево, скосить траву перед собой или перепрыгнуть через <*x*, *y*> клеток. У газонокосилки нет сенсорной информации: она должна быть запрограммирована заранее косить вслепую. Стандартный размер лужайки 8x8 клеток.

Коза предложил данную задачу. чтобы показать преимущества автоматически определяемых функций (*automatically defined functions*, *ADFs*). Управлять газонокосилкой без ADF сложно, однако с использованием ADF задача становится довольно простой. Без ADF особь представляет одно дерево, набор функций для которого показан в табл. 9. С использованием ADF газонокосилка состоит из трех деревьев: главного, дерева ADF1 и дерева ADF2, – а набор функций расширен в соответствии с табл. 10.

Табл. 9. Набор функций для газонокосилки

Function	Arity	Description
(progn2 *a b*)	2	*a*, then *b* are executed. Returns the return value of *b*.
(v8a *i j*)	2	Evaluates *i* and *j*, adds the vectors they return, modulo 8, and returns the result.
(frog *i*)	1	Evaluates *i*. Let $\langle x, y \rangle$ be *i*'s return value. Then frog moves $\langle x, y \rangle$ squares relative to its present rotation, where the positive X axis points in the present "forward" direction of the lawnmower, and the positive Y axis points in the present "heading left" direction. Returns $\langle x, y \rangle$.
mow	0	Moves the lawnmower forward one square, mowing that square of lawn if it is not already mown. Returns $\langle 0, 0 \rangle$.
left	0	Rotates the lawnmower ninety degrees to the left. Returns $\langle 0, 0 \rangle$.
ERCs	0	Ephemeral random constants of the form $\langle x, y \rangle$, where x is an integer chosen from the range $(0, ..., x_{max} - 1)$ and y is an integer chosen from the range $(0, ..., y_{max} - 1)$, where x_{max} and y_{max} are the width and height of the lawn in squares, respectively.

Табл. 10. Расширения к набору инструкций газонокосилки для двух дополнительных ADF деревьев (ADF1 и ADF2). Все деревья имеют одинаковый набор функций за следующим исключением: в дереве ADF2 нет функции (frog i).

Дополнительные ADF функции для главного дерева

Function	Arity	Description
(adf1 *arg1*)	1	Automatically defined function which calls the ADF2 tree.
adf2	0	Automatically defined function which calls the ADF2 tree.

Дополнительные ADF функции для дерева ADF2

Function	Arity	Description
adf2	0	Automatically defined function which calls the ADF2 tree.
arg1	0	The value of argument *arg1* passed when the ADF2 tree is called.

Для вычисления приспособленности газонокосилка помещается в произвольную точку на лужайке и дерево особи исполняется один раз. Каждая команда mow и frog перемещает газонокосилку и косит траву в новой позиции. После завершения исполнения дерева

вычисляется количество скошенных квадратов, которое и определяет значение приспособленности. Пример оптимальной особи с ADF:

Главное дерево: (progn2 (progn2 (adf1 (progn2 (adf1 left) (v8a <7,0> <0,4>))) (progn2 left <3,4>)) (v8a (progn2 (adf1 (v8a left left)) (progn2 (frog mow) (adf1 adf2))) (adf1 (progn2 (v8a <6,7> adf2) (progn2 <1,1> mow)))))

Дерево ADF1: (v8a (v8a (v8a (progn2 (v8a adf2 mow) (v8a adf2 mow)) (frog (v8a mow arg1))) (v8a (v8a (frog arg1) (progn2 <1,4> <2,6>)) (progn2 (v8a <1,5> adf2) (frog mow)))) (v8a (v8a (v8a (progn2 adf2 adf2) (v8a adf2 mow)) (v8a (progn2 arg1 adf2) (frog left))) (frog (v8a (v8a arg1 left) (v8a <7,0> mow)))))

Дерево ADF2: (progn2 (v8a (progn2 (v8a (v8a mow mow) (v8a mow <5,1>)) (v8a (v8a mow left) (progn2 left mow))) (v8a (progn2 (v8a mow mow) (progn2 <1,3> <2,1>)) (v8a (progn2 <3,6> mow) (progn2 left <3,4>)))) (v8a (progn2 (progn2 (v8a mow left) (progn2 <6,6> <1,4>)) (progn2 (v8a mow left) (v8a mow <7,7>))) (progn2 (v8a (progn2 left left) (v8a mow left)) (v8a (progn2 left <2,1>) (v8a <1,7> mow)))))

Хотя данная особь выглядит достаточно впечатляюще, но, на самом деле, задача газонокосилки решается достаточно легко с использованием генетического программирования с ADF. Большая часть кода этой особи просто не нужна. Причина, по которой ADF в рассматриваемой задаче работает хорошо проста и даже «нечестна»: особь для газонокосилки исполняется всего один раз, и в ее дереве нет итераций или рекурсий, поэтому данное дерево должно содержать достаточно команд, чтобы поместить газонокосилку на каждую клетку на лужайке. Чтобы реализовать такой алгоритм в одном дереве нужно большое дерево. Но с использованием ADF деревьев, главное дерево может постоянно вызывать ADF (а дерево ADF1 вызывать функции из дерева ADF2), поэтому общий размер особи получается существенно меньше, и при этом сохраняется преимущество от использования большого числа шагов.

Как и в задаче искусственного муравья, газонокосилка работает через «побочные эффекты» исполнения дерева и порядок следования команд играет значимую роль.