УДК 004.4′242

МЕТОД ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ НА ОСНОВЕ ТЕСТОВЫХ ПРИМЕРОВ С ПОМОЩЬЮ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

***Ф. Н. Царев*[1],**
аспирант
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Предлагается метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования.

Приводятся описания представления автоматов в виде особей алгоритма генетического программирования, операций мутации и скрещивания, а также генетического алгоритма. Применение метода иллюстрируется на примере построения автомата управления часами с будильником.

Ключевые слова *— генетическое программирование, автоматное программирование, машинное обучение.*

Введение

Автоматное программирование — парадигма программирования, в рамках которой программные системы предлагается строить в виде набора взаимодействующих автоматизированных объектов управления [1]. Автоматизированный объект управления состоит из управляющего конечного автомата и объекта управления. Таким образом, поведение каждого автоматизированного объекта управления во многом описывается детерминированным конечным автоматом.

Для большинства задач автоматы удается строить эвристически вручную. Однако в ряде случаев такое построение слишком трудоемко или приводит к неоптимальным результатам. К таким задачам относятся, например, задачи «Умный муравей» [2, 3], «Умный муравей-3» [4] и задача об управлении моделью беспилотного летательного аппарата [5]. Для построения автоматов в таких задачах можно применять генетические алгоритмы (ГА) [6–8].

Традиционный метод построения конечных автоматов с помощью ГА [3, 9–11] использует вычисление функции приспособленности на основе моделирования работы системы со сложным поведением в некоторой внешней среде. Главным недостатком этого метода является то, что при его применении функцию приспособленности необходимо «с нуля» реализовывать для каждой задачи. Кроме того, такой подход к вычислению функции приспособленности связан с большими затратами вычислительных ресурсов.

Целью настоящей работы является разработка метода построения конечных автоматов на основе генетического программирования, в котором устранены указанные недостатки. Для достижения этой цели предлагается осуществлять построение конечных автоматов на основе тестовых примеров.

Постановка задачи

При применении парадигмы автоматного программирования для реализации сущности со сложным поведением выделяется система управления и объект управления. На начальном этапе проектирования программы выделяются события (e1, e2, ...), входные переменные (x1, x2, ...) и выходные воздействия (z1, z2, ...). После этого проектирование программы может идти разными путями. Один из них состоит в написании сценария работы программы, по которому далее эвристически строится автомат. Пример построения автомата таким способом приведен в работе [12].

[1] Научный руководитель — доктор технических наук, профессор, заведующий кафедрой технологий программирования Санкт-Петербургского государственного университета информационных технологий, механики и оптики *А. А. Шалыто*.

Другой подход, который практически не применяется для построения автоматных программ, но достаточно широко распространен при традиционной разработке программ, называется «разработкой на основе тестов» (*test-driven development*) [13]. При его использовании процесс написания кода на языке программирования идет параллельно с написанием тестов для программы, а добавление функциональности в программу осуществляется только после того, как создан тест для проверки этой функциональности. Таким образом, функциональность программы описывается набором тестов для нее.

В случае применения автоматного программирования в качестве тестов для управляющего конечного автомата естественно рассматривать пары последовательностей, одна из которых описывает события и входные переменные, поступающие на вход автомату, а вторая — выходные воздействия, которые должен вырабатывать автомат при обработке входных воздействий. Таким образом, задача построения управляющего конечного автомата становится похожей на задачу построения конечного преобразователя, для решения которой успешно используются ГА [14].

Описание предлагаемого метода

Исходными данными для построения конечного автомата управления системой со сложным поведением являются:

— список событий;

— список входных переменных;

— список выходных воздействий;

— набор тестов Tests, каждый из которых содержит последовательность Input[*i*] событий, поступающих на вход конечному автомату, и соответствующую ей эталонную последовательность Answer[*i*] выходных воздействий.

Отметим, что при использовании описываемого метода входные переменные явным образом не задаются. Для учета входных переменных необходимо добавить в список событий новые события, объединяющие исходные события и логические формулы, содержащие входные переменные. Например, если в списке событий присутствует событие e1, а в списке входных переменных — x1, то новым событием может быть e1 [x1] (произошло событие e1 и переменная x1 истинна).

Отметим также, что для тестов, которые задаются для построения конечного автомата, справедливо свойство, которое можно сформулировать следующим образом: «префиксы тестов являются тестами» — если из входной последовательности событий удалить часть событий, находящихся в ее конце, то результат обработки автоматом этой последовательности будет префиксом исходной выходной последовательности.

Представление конечного автомата в виде хромосомы ГА. Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого состояния и номер начального состояния. Для каждого состояния хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется, и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [15].

Выбор представления графа переходов автомата с помощью списков ребер (в отличие от работы [15], в которой применялись полные таблицы переходов) обоснован тем, что, как правило, в автоматах управления системами со сложным поведением не в каждом состоянии определена реакция на каждое событие.

Алгоритм расстановки пометок. Опишем алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Как было сказано выше, для каждого перехода в особи ГА записано, сколько выходных воздействий должно вырабатываться при его выборе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом переходе, можно определить, какие выходные воздействия должны вырабатываться на переходах, использовавшихся при обработке входной последовательности.

Для каждого перехода T и каждой последовательности выходных воздействий zs вычисляется величина C[T][zs] — число случаев, в которых при обработке входной последовательности, соответствующей одному из тестов, на переходе T должны быть выработаны выходные воздействия, образующие последовательность **zs. Далее**, каждый переход помечается той последовательностью zs_0, для которой величина $C[T][zs_0]$ максимальна.

Функция приспособленности. Функция приспособленности основана на редакционном расстоянии (расстоянии Левенштейна) [16]. Редакционным расстоянием между двумя последовательностями символов называется минимальное число операций замены символа, вставки симво-

ла и удаления символа, которые необходимо выполнить над первой последовательностью для того, чтобы она совпала со второй.

Для вычисления функции приспособленности выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i]. Обозначим последовательность выходных воздействий, которую сгенерировал автомат при входе Input[i], как Output[i]. После этого вычисляется величина

$$FF_1 = \frac{\sum_{i=1}^{n}\left(1 - \frac{ED(\text{Output}[i], \text{Answer}[i])}{\max(|\text{Output}[i]|, |\text{Answer}[i]|)}\right)}{n},$$

где $ED(A, B)$ — редакционное расстояние между строками A и B. Отметим, что значения функции FF_1 лежат в пределах от 0 до 1. При этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Эта функция вычисляется следующим образом:

$$FF_2 = \begin{cases} 0{,}5 \cdot T \cdot FF_1 + \frac{1}{M}(\text{M} - cnt), FF_1 < 1 \\ T + \frac{1}{M}(M - cnt), FF_1 = 1 \end{cases},$$

где T — «стоимость» прохождения всех тестов; M — произвольное целое число, большее максимального числа переходов в автомате; cnt — число переходов в автомате. При проведении вычислительных экспериментов были выбраны следующие значения: $T = 20$, $M = 100$.

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньшее число переходов. Кроме этого, автомат, который «идеально» проходит все тесты, оценивается выше, чем автомат, проходящий тесты не идеально.

Операция мутации. При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0,05) выполняется каждое из действий:

— изменение начального состояния;

— изменение описания каждого из переходов;

— удаление или добавление перехода для каждого из состояний.

После выполнения операции мутации может возникнуть ситуация, когда в автомате из одного состояния присутствуют два перехода по одному и тому же событию. Для устранения таких переходов применяется операция удаления дублирующихся переходов.

Операция удаления дублирующихся переходов. В целях удаления дублирующихся переходов для каждого состояния выполняются следующие операции: последовательно просматривается список переходов из этого состояния, при этом запоминаются события, переходы по которым определены для этого состояния. Если очередной переход Tr происходит по событию, для которого в списке уже есть переход, то переход Tr удаляется из списка.

Операция скрещивания. Скрещивание описаний автоматов производится следующим образом. Обозначим как P1 и P2 — «родительские» особи, а **S1 и S2 — особи-«потомки». Для начальных** состояний S1.is и S2.is автоматов S1 и S2 будет верно одно из двух соотношений:

S1.is = P1.is и S2.is = P2.is;

S1.is = P2.is и S2.is = P1.is.

Опишем, как устроены переходы автоматов S1 и **S2. Скрещивание описаний автоматов производ**ится отдельно для каждого состояния. Обозначим список переходов из состояния номер i автомата P1 как P1.T[i], а список переходов из состояния номер i автомата P2 как P2.T[i]. Для выполнения «скрещивания переходов» с равной вероятностью может быть выбран один из двух методов.

При использовании *традиционного метода скрещивания* списки переходов S1.T[i] и S2.T[i] строятся следующим образом.

1. Строится общий список переходов, в который помещаются переходы, входящие как в P1.T[i], так и в P2.T[i].

2. К полученному списку применяется случайная перестановка.

3. Далее возможны два равновероятных варианта:

• либо в S1.T[i] помещаются первые $|$P1.T[i]$|$ переходов из полученного списка, а в S2.T[i] — оставшиеся переходы;

• либо в S1.T[i] помещаются первые $|$P2.T[i]$|$ переходов из полученного списка, а в S2.T[i] — оставшиеся переходы.

При использовании *метода скрещивания с учетом тестов* списки переходов S1.T[i] и S2.T[i] строятся следующим образом.

1. Составляется список всех используемых тестов, упорядоченный по возрастанию нормированного редакционного расстояния между «правильным ответом» Answer и последовательностью Output выходных воздействий, генерируемой автоматом, — значения выражения $\frac{\text{ED}(\text{Output}[i], \text{Answer}[i])}{\max(|\text{Output}[i]|, |\text{Answer}[i]|)}$. В автоматах P1 и P2 помечаются те переходы, которые используются при обработке первых 10 % тестов из полученного упорядоченного списка.

2. *Помеченные* переходы копируются в S1.*T*[*i*] и S2.*T*[*i*] напрямую.

3. Строится общий список переходов, в который помещаются *непомеченные* переходы, входящие как в P1.*T*[*i*], так и в P2.*T*[*i*].

4. К полученному списку *L* применяется случайная перестановка.

5. Список S1.*T*[*i*] дополняется первыми переходами из списка *L* до размера |P1.*T*[*i*]|, а список S2.*T*[*i*] дополняется оставшимися переходами.

В обоих случаях к получившимся в результате скрещивания автоматам S1 и S2 применяется операция удаления дублирующихся переходов.

Пример применения предлагаемого метода

Применение предлагаемого метода иллюстрируется на примере построения автомата управления часами с будильником [1]. Эти часы имеют три кнопки (помеченные буквами «А», «Н», «М»), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника. Если будильник выключен, то кнопки «Н» и «М» служат для установки текущего времени, а кнопка «А» переводит часы в режим «Настройка будильника», в котором кнопки **«Н» и «М» устанавливают** не текущее время, а время срабатывания будильника. Повторное нажатие кнопки **«А» включает** будильник. После этого если текущее время совпадает со временем срабатывания будильника, то включается звонок, который отключается либо нажатием кнопки **«А», либо само**произвольно через минуту. Кроме этого, нажатие кнопки «А» приводит к выключению будильника.

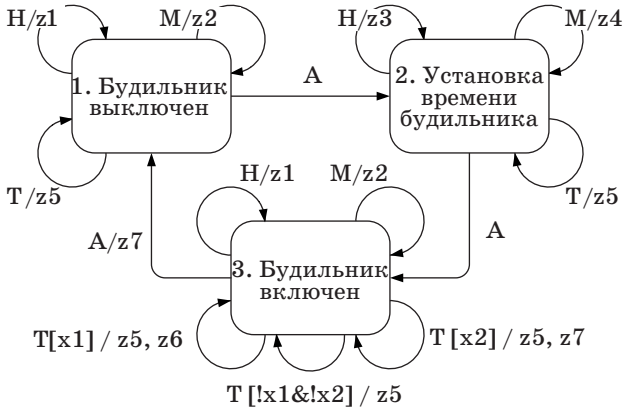
Рассматриваемые часы с будильником являются системой со сложным поведением, так как в ответ на одни и те же входные события (нажатия кнопок) в зависимости от режима работы генерируются различные выходные воздействия. Поведение этих часов может быть описано с помощью конечного автомата [1], который содержит три состояния (рис. 1).

Система управления часами с будильником имеет четыре события:

- H — нажата кнопка «Н»;
- M — нажата кнопка «М»;
- A — нажата кнопка «А»;
- T — генерируется таймером каждую секунду.

Кроме этого, она содержит две входные переменные:

- x1 — верно ли, что текущее время совпадает со временем срабатывания будильника;
- x2 — верно ли, что текущее время на минуту больше времени срабатывания будильника?



■ ***Рис. 1.*** *Граф переходов автомата управления часами с будильником*

Число выходных воздействий равно семи:

- z1 — увеличить на единицу число часов в текущем времени;
- z2 — увеличить на единицу число минут в текущем времени;
- z3 — увеличить на единицу число часов во времени срабатывания будильника;
- z4 — увеличить на единицу число минут во времени срабатывания будильника;
- z5 — прибавить минуту к текущему времени;
- z6 — включить звонок будильника;
- z7 — выключить звонок будильника.

Система тестов для построения автомата управления часами с будильником состояла из 38 тестов, описывающих его работу во всех трех режимах. В качестве примера приведем тесты для состояния «Будильник выключен» (таблица).

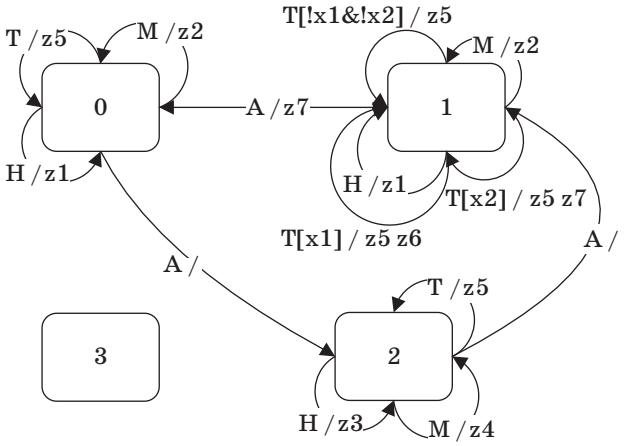
Построение конечного автомата управления часами с будильником проводилось при следующих параметрах алгоритма генетического программирования:

- размер поколения — 2000 особей;
- доля «элиты» — наиболее приспособленных особей, напрямую переходящих в следующее поколение, — 10 %;
- число поколений до малой «мутации поколения» — 100 поколений;
- число поколений до большой «мутации поколения» — 150 поколений;
- размер автоматов в начальном поколении — четыре состояния.

Было проведено 1000 запусков алгоритма с указанными параметрами. Цель в каждом из них состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности, соответствующее такому автомату, — 20.86). На каждом из запусков алгоритма генетического программирования был построен автомат (рис. 2), в котором из начального (нулевого) состояния до-

■ *Тесты для состояния «Будильник выключен»*

	Входная последовательность	Выходная последовательность	Комментарий
1	*T, T, T, T*	z5, z5, z5, z5	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту
2	*H, H, H, H*	z1, z1, z1, z1	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено на единицу
3	*M, M, M, M*	z2, z2, z2, z2	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу
4	*T, M, H, T, T, T, M, T, H, H, T, M*	z5, z2, z1, z5, z5, z5, z2, z5, z1, z1, z5, z2	Описывает обработку событий H, M и T в состоянии «Будильник выключен»
5	*A, A, A, T, T, T, T*	z7, z5, z5, z5, z5	После трех нажатий кнопки **«A» часы должны находиться в состоянии** «Будильник выключен». Аналог первого теста
6	*A, A, A, H, H, H, H*	z7, z1, z1, z1, z1	После трех нажатий кнопки **«A» часы должны находиться в состоянии** «Будильник выключен». Аналог второго теста
7	*A, A, A, M, M, M, M*	z7, z2, z2, z2, z2	После трех нажатий кнопки **«A» часы должны находиться в состоянии** «Будильник выключен». Аналог третьего теста

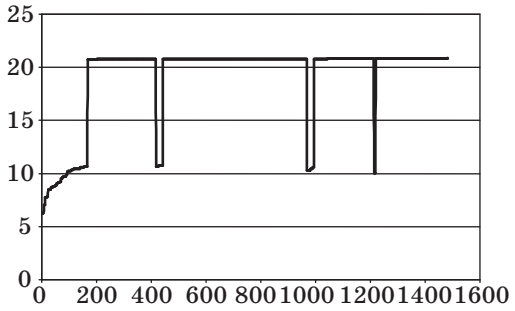


■ ***Рис. 2.*** *Граф переходов автомата, построенного с помощью алгоритма генетического программирования*

стижимы только три состояния из четырех. Если удалить недостижимое состояние, то этот граф переходов будет изоморфен графу переходов, построенному вручную.

График зависимости максимального значения функции приспособленности от номера поколения представлен на рис. 3 при одном из запусков алгоритма.

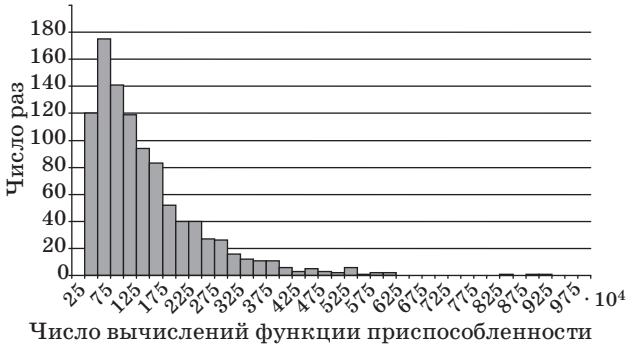
Как видно из графика, автоматы, входящие в начальное поколение, проходят тесты примерно наполовину. Примерно к двухсотому поколению был построен автомат, полностью проходящий все тесты, однако содержащий достаточно большое число переходов. Далее шел процесс уменьшения числа переходов, во время которого три раза (в районе 400-, 1000- и 1200-го поколений) к популяции применялась операция «большой мутации», в результате чего значение функции



■ ***Рис. 3.*** *Зависимость максимального значения функции приспособленности от номера поколения*

приспособленности уменьшалось примерно до десяти. В итоге в 1482-м поколении был построен автомат, полностью проходящий все тесты и содержащий 14 переходов.

Для каждого из запусков запоминалось число вычислений функции приспособленности (которое равно числу просмотренных во время работы авто-



■ ***Рис. 4.*** *Распределение числа вычислений функции приспособленности в проведенных вычислительных экспериментах*

матов) в процессе построения автомата. На рис. 4 показано распределение этой величины, полученное в результате вычислительных экспериментов.

Минимальное значение числа вычислений функции приспособленности составило 256 063, максимальное — 9 239 523. В предположении, что эта величина распределена по логнормальному распределению, ее среднее значение составляет 1 443 351.20 (стандартное отклонение — 1 103 401.82).

Заключение

В работе предложен метод построения автоматов управления системами со сложным поведением с учетом тестов с помощью генетического программирования. При использовании разработанного метода для построения управляющих конечных автоматов необходимо задать только тестовые примеры, а вычисление функции приспособленности требует существенно меньше вычислительных ресурсов, чем при использовании моделирования для вычисления функции приспособленности. Рассмотрен пример применения разработанного метода.

Исследования проводятся по государственному контракту, выполняемому в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы».

Литература

1. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. — СПб.: Питер, 2010. — 176 с.
2. **Jefferson D. et al.** The Genesys System: Evolution as a Theme in Artificial Life // Proc. of Second Conf. on Artificial Life. MA: Addison-Wesley, 1992. P. 549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html (дата обращения: 19.03.2010).
3. **Царев Ф. Н., Шалыто А. А.** Применение генетического программирования для генерации автомата в задаче об «Умном муравье» // Интегрированные модели и мягкие вычисления в искусственном интеллекте: Сб. тр. IV Междунар. науч.-практ. конф. Т. 2. М.: Физматлит. 2007. С. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf (дата обращения: 19.03.2010).
4. **Бедный Ю. Д., Шалыто А. А.** Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/ant (дата обращения: 19.03.2010).
5. **Параценко Д. А., Царев Ф. Н., Шалыто А. А.** Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок»: Проектная документация / СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates (дата обращения: 19.03.2010).
6. **Гладков Л. А., Курейчик В. В., Курейчик В. М.** Генетические алгоритмы. — М.: Физматлит, 2006. — 320 с.
7. **Рассел С., Норвиг П.** Искусственный интеллект: современный подход. — М.: Вильямс, 2006. — 1407 с.
8. **Koza J. R.** Genetic programming: on the programming of computers by means of natural selection. — MIT Press, 1992. — 835 с.
9. **Поликарпова Н. И., Точилин В. Н., Шалыто А. А.** Применение генетического программирования для генерации автоматов с большим числом входных переменных // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 24–42.
10. **Данилов В. Р.** Метод представления автоматов деревьями решений для использования в генетическом программировании // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 103–108.
11. **Давыдов А. А., Соколов Д. О., Царев Ф. Н.** Применение генетических алгоритмов для построения автоматов Мура и систем взаимодействующих автоматов Мили на примере задачи об «Умном муравье» // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 108–114.
12. **Мазин М. А., Парфенов В. Г., Шалыто А. А.** Разработка интерактивных приложений Macromedia Flash на базе автоматной технологии: Проектная документация / СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/flash/ (дата обращения: 19.03.2010).
13. **Бек К.** Экстремальное программирование: разработка через тестирование. — СПб.: Питер, 2003. — 224 с.
14. **Lucas S., Reynolds T.** Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. 2007. Vol. 11. Is. 3. P. 308–325.
15. **Lucas S., Reynolds T.** Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // IEEE Transactions on Evolutionary Computation. 2005. Vol. 27. Is. 7. P. 1063 –1074.
16. **Левенштейн В. И.** Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. Академии наук СССР. 1963. № 4. С. 845–848.