Министерство образования и науки Российской Федерации
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет Информационных Технологий и Программирования
Направление (специальность) Прикладная математика и информатика
Квалификация (степень) Бакалавр прикладной математики и информатики
Специализация ----------
Кафедра Компьютерных технологий **Группа** 4538

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Применение генетических алгоритмов для построения клеточных автоматов

Автор квалификационной работы Бедный Ю.Д. (подпись)

Руководитель Корнеев Г.А., Шалыто А.А. (подпись)

Санкт-Петербург
2006

Оглавление

ВВЕДЕНИЕ

Генетические алгоритмы являются одним из бурно развивающихся и перспективных направлений в искуственном интеллекте и программировании. С их помощью могут быть найдены решения многих сложных и интересных задач в различных областях. Например, автоматическое создание программ, задачи оптимального управления, регрессионный анализ и эмпирическое прогнозирование, теория игр, символьное интегрирование и дифференцирование, проектирование мультиагентных систем и *клеточных автоматов*, моделирование эволюционных процессов, предсказание структуры сложных биологических объектов и создание нейронных сетей. К сожалению, в нашей стране этой области исследований уделяется мало внимания. Одна из *задач* данной работы – хотя бы частично восполнить указанный пробел.

Часто автоматы обладают сложным поведением, как, например, в задаче классификации плотности, рассматриваемой в настоящей работе. В таком случае их проектирование является нетривиальной и трудоемкой задачей. Кроме того, статическое определение автоматов (**полное** их задание на этапе написания программы) не обладает достаточной гибкостью. Зачастую, предпочтительней создавать автоматы “на лету”, не накладывая ограничений сверху на количество их состояний и сложность поведения. Возникает естественное желание – *автоматизировать процесс построения автоматов* и сделать его *динамическим*, поручив основную работу компьютеру.

Однако возможно ли автоматизированное построение автоматов? Каким способом оно может быть осуществлено? В данной работе *генетические алгоритмы* выбраны *как метод проектирования автоматов*. Вначале работы кратко излагаются ключевые концепции генетических алгоритмов, приводится перечень задач, в которых генетическое программирование успешно применяется для построения автоматов, а затем подробно рассматривается, изучается и *формализуется* наиболее интересная из них – задача классификации плотности для одномерных детерминированных бинарных клеточных автоматов. При этом

цель исследований – установить применимость генетических алгоритмов к построению автоматов рассматриваемого класса, и на примере выбранной задачи рассмотреть различные подходы к ее решению, использующие генетические алгоритмы. Анализ достоинств и недостатков существующих подходов к решению задачи классификации плотности приводит к созданию *нового метода решения этой задачи, который назван* **композитное генетическое программирование**.

Задача классификации плотности для клеточных автоматов выбрана не случайно. Дело в том, что клеточные автоматы интересны для изучения, так как являются наглядным примером системы, обладающей очень простой структурой и в то же время сложным поведением. Кроме того, клеточные автоматы тесно связаны с понятием *мультиагентных систем*, являясь их разновидностью.

В заключение работы анализируются ее результаты, и приводится ряд открытых вопрос и направлений для дальнейших исследований в этой области.

Ряд терминов (например, “coevolution”, “resource sharing”, “gene expression programming”), используемых в данной работе, не имеет точного и устоявшегося перевода на русский язык в виду малого количества публикаций в нашей стране, посвященных генетическому программированию. Употребление этих терминов в данной работе, как правило, сопровождается их оригинальным написанием на английском языке, приводимым в скобках.

ГЛАВА 1. ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ КЛЕТОЧНЫХ АВТОМАТОВ

1.1. Генетические алгоритмы

Природа удивляет оптимальностью и простотой структуры различных биологических объектов, а также согласованностью и эффективностью их работы. Многих исследователей давно интересовал “философский” вопрос – возможно ли эффективное построение важных и полезных для человека систем на основе принципов биологической эволюции?

Подобные идеи возникали у ряда исследователей. В 1966 г. Л. Фогель, А. Оуэнс и М. Уолш предложили схему эволюции логических автоматов, решающих задачи прогноза. В 1975 г. вышла книга Дж. Холланда “Адаптация в естественных и искусственных системах”, в которой был предложен генетический алгоритм, исследованный в дальнейшем учениками и коллегами Дж. Холланда в Мичиганском университете. Эти работы заложили основы эволюционного программирования.

Особенно эффективно применяется эволюционное программирование для моделирования биологических процессов и прогнозирования структуры биологических объектов. Из известных задач такого типа интересны следующие: определение функций генов, и предсказание третичной структуры белка по его первичной структуре (определение последовательности нуклеотидов) [1, 2].

В общем виде эволюционный алгоритм – это оптимизационный метод, базирующийся на эволюции популяции “особей”. Каждая особь характеризуется приспособленностью – функцией ее генов. Задача оптимизации состоит в максимизации функции приспособленности. В процессе эволюции – в результате отбора, рекомбинаций и мутаций геномов особей происходит поиск особей с высокими приспособленностями.

По сравнению с обычными оптимизационными методами эволюционные алгоритмы имеют особенности: параллельный поиск, случайные мутации и рекомбинации уже найденных хороших решений. Они хорошо подходят для оптимизации плохо определенных нелинейных функций.

Существуют различные модификации эволюционных алгоритмов:

- *традиционные генетические алгоритмы (Conventional Genetic Algorithms)* [1].
- *генетическое программирование (Genetic Programming*), основателем которого является *J.R.Koza* [3].
- *программирование с экспрессией генов (Gene Expression Programming)***,** предложенное *C.Ferreira* [4].

В разделе, посвященном применению различных методов генетического программирования к решению *задачи классификации плотности*, будут рассмотрены их отличия, а сейчас будут рассмотрены принципы работы эволюционных алгоритмов на примере наиболее простого из них – *традиционного генетического алгоритма*.

1.1.1. Традиционный генетический алгоритм

Генетический алгоритм (ГА) [1] – это компьютерная модель эволюции *популяции* искусственных *особей*. Каждая особь k полностью характеризуется своей *хромосомой* x_k. В генетическом алгоритме (в отличии, например, от программирования с экспрессией генов) геном особи состоит из единственной хромосомы. Хромосома определяет *приспособленность* особи $f(x_k)$. Хромосомы всех особей генетического алгоритма (в отличии от генетического программирования) имеют одинаковую структуру – представляются строками фиксированной длины.

Задача генетического алгоритма – максимизация функции приспособленности. Данная задача решается постепенно – по мере эволюции. Эволюция состоит из последовательности поколений – популяций.

Для каждого поколения *отбираются* особи с большими значениями приспособленности. Хромосомы отобранных особей с некоторой вероятностью *рекомбинируются,* затем, с другой, как правило, малой вероятностью, подвергаются *мутациям*, а вслед за этим переносятся в следующее поколение. Отбор (селекция), рекомбинация и мутация являются генетическими операциями. Размер популяции и функция приспособленности не изменяются в процессе эволюции от поколения к поколению. Ожидается, что каждое последующее поколение будет содержать "более приспособленные" особи, нежели предыдущее. Здесь можно проследить непосредственную аналогию с эволюцией особей в биологии.

Пусть размер популяции *r*. Хромосомы представлены битовыми строками фиксированной длины *l*. **Схема** генетического алгоритма с одноточечной рекомбинацией и мутацией может быть представлена следующим образом.

Шаг 1. Создадим случайную начальную популяцию $p_0 = \{x_k^0\}$, $t = 0$. Способ случайного создания такой популяции зависит от конкретной задачи. Например, каждая *k-ая* битовая строка x_k^0 длины *l* может быть сгенерирована случайно с равномерным распределением над каждым битом. Или, что тоже самое, равновероятно выбрано целое число из диапазона $0..2^l - 1$ и его двоичная запись рассматривается как генерируемая строка.

Шаг 2. Вычислим приспособленность $f(x_k^t)$ каждой особи x_k^t популяции $p_t = \{x_k^t\}$. Обычно это наиболее трудоемкий по времени шаг генетического алгоритма.

Шаг 3. Производя отбор особей x_k^t в соответствии с их приспособленностями $f(x_k^t)$ и применяя генетические операторы рекомбинации и мутации к отобранным особям, сформируем популяцию следующего поколения $p_{t+1} = \{x_k^{t+1}\}$. Обычно данный шаг разбивается на *r* итераций, на каждой из которых в новую популяцию добавляется *одна* особь следующим образом:

- пропорционально их приспособленности из текущей популяции выбираются *две* особи-"родители": $\boldsymbol{x}, \boldsymbol{y}$;

- с некоторой вероятностью происходит рекомбинация родительских особей – “скрещивание” ***x*** с ***y***. В результате образуются их “потомки” z_1, z_2. Это происходит следующем образом: вероятностно выбирается с равномерным распределением над 0..*l-1* целое число $k \in [0, l-1]$. Первые *k* символов z_1 будут совпадать с символами *x* на соответствующих позициях, а последние *l-k* – с символами *y*. Для z_2 – наоборот. Пример "скрещивания" строк фиксированной длины: $l = 6$, $x =$ **000100**, y = **111000**, $k = 2$, тогда $z_1 =$ **001000**, $z_2 =$ **110100**. “Забудем” о родителях, обозначив $x = z_1$, y = z_2.
- равновероятно выберем одну из особей (*x* либо *y)* и отбросим ее – она не будет перенесена в следующую популяцию. Оставшуюся особь обозначим как *z*;
- с некоторой малой вероятностью произведем мутацию особи *z,* вероятностно выбрав целое число $k \in [0, l-1]$ – позицию мутации. Например, $k = 4$. До мутации $z =$ **001000**, после мутации $z =$ **001010**.
- перенесем *z* в новую популяцию.

Шаг 4. Повторим шаги 2, 3 для $t = 0, 1, 2, \ldots$ до тех пор, пока не выполнится условие окончания эволюционного поиска (прекращается рост максимальной приспособленности в популяции, число поколений *t* достигает заданного предела и т.д.).

1.1.2. Математический аппарат традиционного генетического алгоритма

Почему генетический алгоритм работает – почему в вероятностном смысле происходит постепенная оптимизация целевой функции приспособленности? Обоснование функционирования генетических алгоритмов приводится в работах [1, 3] (по сути, неформальное) и формализованное (правда с существенными ограничениями на функцию приспособленности) приводится в работе [5].

Математическая модель традиционного генетического алгоритма описана в работах [6, 7].

1.1.3. Несколько задач, в которых генетическое программирование используется для построения автоматов

Перечислим эти задачи:

- итерированная дилемма узников (*Iterated prisoner's dilemma*) [1, 8];
- задача про "умного" муравья (*Artificial ant problem*) [3, 9];
- задача классификации плотности для клеточных автоматов (*Density classification task for cellular automata*), рассматриваемая в настоящей работе;
- задача синхронизации для клеточных автоматов (*Synchronization task for cellular automata*) [10];
- задача упорядочивания для клеточных автоматов (*Ordering task for cellular automata*) [11].

1.2. Клеточные автоматы

1.2.1. Ведение в клеточные автоматы

Клеточные автоматы – одни из наиболее интересных объектов дискретной математики. Идея клеточных автоматов, принадлежащая Джону фон Нейману, появилась в конце 40-х годов XX века. В нашей стране первые работы анализирующих структуру, поведение и свойства клеточных автоматов появились в середине 70-х годов XX века в Московском государственном университете (А.Н. Колмогоров, Г.Л. Курдюмов, А.Л. Тоом, Л.А. Левин, П. Гач) [12–17]. Пожалуй, одним из самых известных клеточных автоматов является клеточный автомат под названием игра “Жизнь”, созданный Джоном Хортоном Конвеем в 1970 г. [18, 19]. Данный автомат эмулирует процессы, происходящие при зарождении, развитии и гибели колонии живых организмов. Стивен Вольфрам (Stephen Wolfram) и вовсе считает теорию клеточных автоматов “новым видом

науки” и в книге “A New Kind of Science” [20] приводит обоснование этого мнения, что, правда, вызвало неоднозначную реакцию в научном мире.

Большой интерес к клеточным автоматам вызван следующими фактами. Во-первых, клеточные автоматы эквивалентны как модель вычислимости машине Тьюринга. Следовательно, они могут быть использованы для исследования алгоритмической разрешимости различных задач как средство распознавания языков. Данная идея, освещена в работах [14, 17].

Во-вторых, они представляют собой мультиагентную систему с рефлексивными агентами, частично наблюдающими дискретный мир (как вероятностный, так и детерминированный) [21], а мультиагентные системы такого вида широко используются для изучения искусственного интеллекта. Примерами мультиагентных систем в жизни и биологии являются колонии насекомых, экономические системы, иммунная система организма, мозг человека.

В-третьих, клеточные автоматы обладают интересной способностью – осуществлять распределенные вычисления. Локальность взаимодействий и слабая восприимчивость клеточных автоматов к помехам (“шуму”) являются существенными их достоинствами. В частности, бесконечный одномерный бинарный автомат, рассмотренный в работах [12, 16], устраняет любое конечное возмущение за конечное время. Автомат сохраняет данное свойство при малых возмущениях и в случае конечной длины, что доказано в статье [22]. Это свойство клеточных автоматов позволяет надеяться, что на их основе возможно создание как программных, так и аппаратных, эффективных и надежных систем, осуществляющих распределенные вычисления. К сожалению, на данный момент не известен общий метод проектирования таких систем на основе клеточных автоматов.

В данной работе рассматриваются одномерные бинарные детерминированные клеточные автоматы. На основе работ [12–18] приведем достаточно формальное, но в то же время, простое их определение. В целом, оно совпадает с определением, данным в работе [14]. Введем также несколько сопутствующих определений, полезных в дальнейшем. Для краткости изложения

будем использовать термины “клеточный автомат” или просто “автомат” вместо термина “одномерный бинарный детерминированный клеточный автомат” там, где это не вызовет путаницы.

1.2.2. Математический аппарат клеточных автоматов

Одномерный детерминированный клеточный автомат *(однородная одномерная детерминированная среда)* – это специального вида оператор $P: \mathrm{X}_n \to \mathrm{X}_n$. Этот оператор действует на пространстве $X_n = S_n^{\mathrm{Z}} = \{x\},\ x = (x_i),\ x_i \in S_n = \{0, \ldots, n\},\ i \in \mathrm{Z}$, которое состоит из всех бесконечных в обе стороны последовательностей, члены которых – целые числа от 0 до *n*: Автомат *P*, действующий на множестве X_n, задается радиусом *r*, $r \in \mathrm{Z}^+$ и функцией $f: S_n^{2r+1} \to S_n$ и имеет следующий вид. Для любого $x \in \mathrm{X}_n$ *i*-я компонента $(Px)_i = f(x_{i-r}, \ldots, x_{i+r})$.

Назовем $x \in \mathrm{X}_n$ ***островом***, если лишь конечное число его компонент x_i отлично от нуля. Назовем *x* ***пустым островом***, если все его компоненты – нули. Обозначим за X_n' – ***совокупность островов*** в X_n . Говорят, что автомат *P* ***размывает остров*** *x*, если существует натуральное число *t*, такое, что $\forall n \geq t : P^n x$ – пустой остров. Автомат *P* ***размывающий***, если он размывает всякий остров $x \in \mathrm{X}_n$.

Назовем автомат ***бинарным,*** если он действует на $\mathrm{X}_1 = \mathrm{B} = \{0, 1\}$. Говорят, что автомат *P* является ***монотонным***, если задающая его функция f монотонна – $x_{i-r} \leq y_{i-r}, \ldots, x_{i+r} \leq y_{i+r} \Rightarrow f(x_{i-r}, \ldots, x_{i+r}) \leq f(y_{i-r}, \ldots, y_{i+r})$.

Зафиксируем некоторый автомат *P*. Состояния (конфигурации) *x, y* $(x, y \in \mathrm{X}_n)$ называются ***эквивалентными*** для данного автомата, если существует натуральное число *t*, для которого $P^t x = P^t y$. Назовем состояние $x \in \mathrm{X}_n$ ***притягивающим***, если ему эквивалентны все $y \in \mathrm{X}_n$, для которых множество $\{i \mid y_i \neq x_i\}$ конечно. Автомат *P* называется ***консервативным***, если он имеет

неэквивалентные притягивающие состояния. Состояние $x \in \mathrm{X}_n$ называется ***стационарным,*** если $Px = x$ – если состояние сохраняется во времени. Состояние x называется ***периодическим***, если $\exists k \in \mathrm{N}$, такое что $\forall t : P^t x = P^{t+k} x$. Любое стационарное состояние является периодическим.

Функцию f, порождающую оператор P будем также называть ***правилом***.

1.2.3. Иллюстрация к определению клеточного автомата

На рис.1 приведен пример одномерного бинарного детерминированного клеточного автомата единичного радиуса, и проиллюстрирован один шаг его работы.

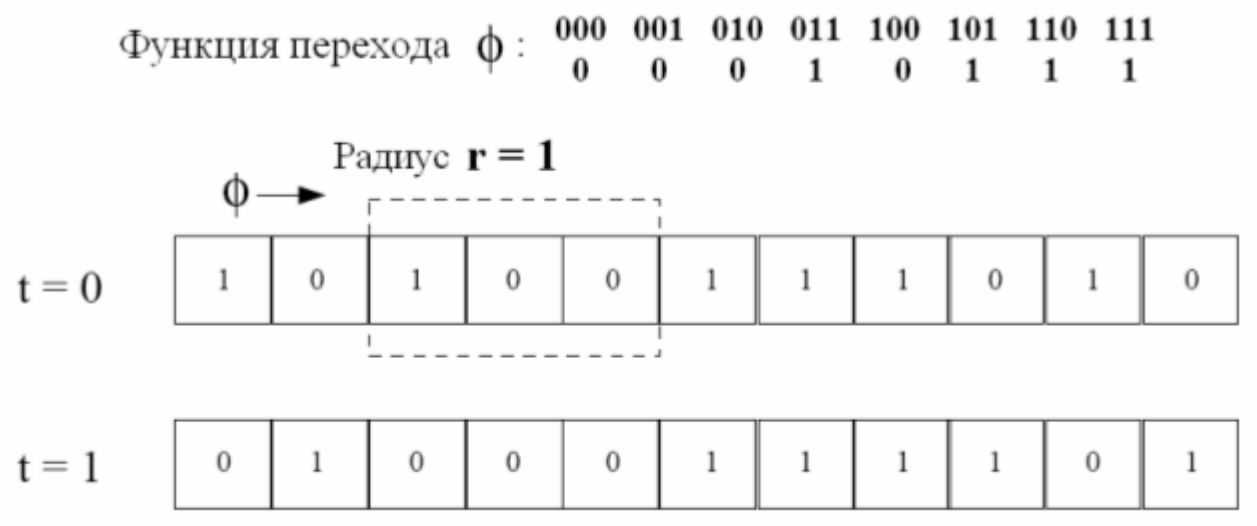

Рис. 1. Одномерный бимнарный детерминированный клеточный автомат

ГЛАВА 2. ЗАДАЧА КЛАСИФИКАЦИИ ПЛОТНОСТИ

2.1. Постановка задачи

Задача классификации плотности (*density classification task – DCT*), рассматриваемая в данной работе, является интересным и наглядным примером того, что клеточные автоматы с малым радиусом (ограниченным локальным взаимодействием) могут обладать достаточно сложным глобальным поведением, позволяющим осуществлять вычисления. Сформулируем задачу классификации плотности для одномерных бинарных клеточных автоматов.

Рассмотрим некоторый детерминированный одномерный бинарный клеточный автомат P' радиуса r', заданный функцией f'. Согласно определению, данному выше, такой автомат действует на множестве 2^Z. Для некоторого натурального числа L, где $L > r'$, существует сужение P оператора P' с множества 2^Z на множество 2^L, задаваемое радиусом $r = r'$, функцией $f \equiv f'$, такое что $(Px)_i = f(x_{(i-r+L)\div L}, x_{(i-r+1+L)\div L}, \ldots, x_{(i-r+k+L)\div L}, \ldots, x_i, \ldots, x_{(i+r-k)\div L}, \ldots, x_{(i+r-1)\div L}, x_{(i+r)\div L})$.

Менее формально: возьмем отрезок длины L, “свернем” его в окружность и зададим на этой окружности оператор P.

Зададим вещественное число $\rho^* \in [0,1]$. Определим вещественную функцию $\rho(x): 2^L \to [0,1]$ как “плотность конфигурации” $x \in 2^L$ — отношение количества единиц в ней к L. Будем говорить, что клеточный автомат P ***распознает (классифицирует)*** конфигурацию x, если $\exists n: P^{(n)}x = s$, где s — стационарное состояние:

a) “все нули” (0^L), если $\rho(x) < \rho^*$;

b) “все единицы” (1^L), в противном случае: $\rho(x) \geq \rho^*$;

Рис. 2 иллюстрирует данное определение.

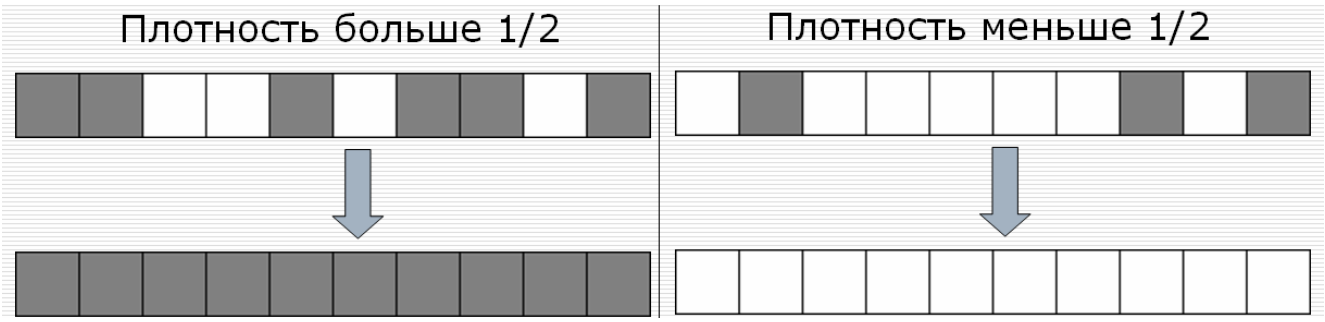

Рис. 2. Клеточный автомат распознает конфигурацию, где: $\rho > 1/2$ **(слева) и** $\rho < 1/2$ **(справа)**

Зададим индикаторную функцию $I_{\rho^*}(P,x): 2^{2^{2r+1}} \times 2^L \to \{0,1\}$, принимающую единичное значение, если ли оператор P (заданный радиусом r и функцией $f: 2^{2r+1} \to \{0,1\}$) распознает конфигурацию x, и значение ноль в противном случае. Зафиксировав L и r, определим функцию $g_{L,r,\rho^*}: 2^{2r+1} \to [0,1] \subset \Re;\ g_{L,r,\rho^*}(f \in 2^{2r+1}) = \dfrac{\sum_{x \in 2^L} I_{\rho^*}(f,x)}{2^L}$.

Постановка задачи классификации плотности (постановка *DCT*): при заданных r, L и ρ^* требуется найти $f_{\max} \in 2^{2^{2r+1}}$, на которой достигается максимум функции g_{L,r,ρ^*}, то есть $\max\limits_{f \in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f) = g_{L,r,\rho^*}(f_{\max})$.

Другими словами, при заданном радиусе автомата и длине окружности требуется найти правило, задающее клеточный автомат, распознающий как можно большее количество конфигураций. Будем говорить, что функция (правило) $f' \in 2^{2^{2r+1}}$ ***полностью (совершенно) решает задачу DCT***, если $g_{L,r,\rho^*}(f') = 1$.

В неформальной постановке задача может быть сформулирована так: *требуется найти правило заданного радиуса, классифицирующее наибольшее количество начальных конфигураций заданной длины.* Ниже приведен пример автомата, классифицирующего начальную конфигурацию, содержащую больше нулей, чем единиц.

На рис. 3 показан процесс распознавания автоматом начальной конфигурации, плотности меньше 0.5.

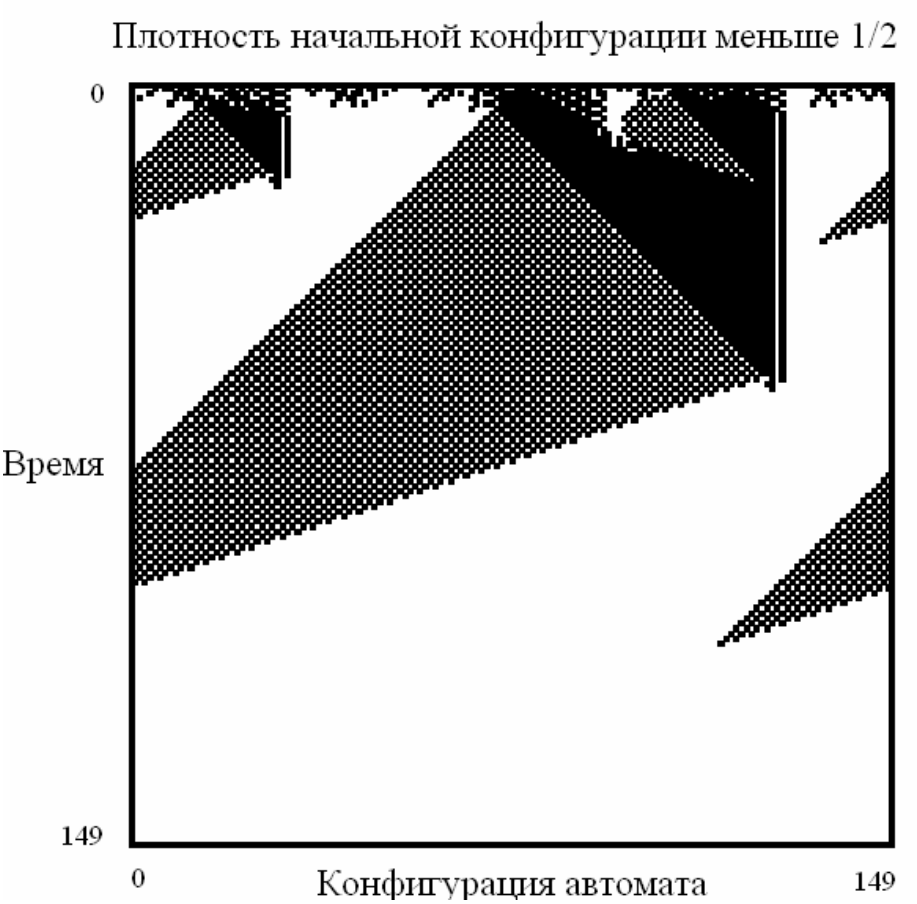

Рис. 3. Распознавания автоматом начальной конфигурации, $\rho < 1/2$

2.2. Обзор теоретических результатов в данной области

Первые публикации [23, 24], в которых поставлена (хотя и менее формально, чем в настоящей работе) и рассмотрена данная задача, появились в 1993 г. Исследователи положили $\rho^* = 1/2$.

Затем выбрали L достаточно больш*и*м (равным 149), чтобы вычисление индикаторной функции $I_{1/2}$ для заданного правила f на всевозможных конфигурациях c (а всего их 2^L) не являлось тривиальной задачей.

Радиус автомата r был выбран достаточно большим (равным трем), чтобы решение задачи невозможно было найти простым перебором всех правил (всего их $2^{2^{2r+1}}$), но в то же время достаточно малым, чтобы она не потеряла смысл и интерес — чтобы автоматы обладали лишь ограниченным, локальным взаимодействием. Действительно, при $r = L$ существует тривиальное правило

$f(x) = \begin{cases} 0, \rho(x) < \rho^* \\ 1, \rho(x) \geq \rho^* \end{cases}$, на котором $g_{L,r,½}$ принимает единичное значение. Получили задачу: найти клеточный автомат (вернее правило его задающее) радиуса три, классифицирующий как можно больше начальных конфигураций длины 149. Автомат классифицирует конфигурацию, если переводит ее в стационарное состояние "все нули" ("все единицы"), когда она содержит больше нулей (единиц), чем единиц (нулей).

Данная группа исследователей использовала ***генетические алгоритмы, как средство для построения клеточных автоматов***, решающих *DCT*. При решении задачи с помошью генетических алгоритмов возникают трудности с определением функции приспособленности – *apriori* непросто найти количество конфигураций, классифицируемых конкретным правилом, так как всего таких конфигураций 2^{149}. Поэтому авторами указанных работ была введена следующая функция приспособленности – процент конфигураций из репрезентативного тестового набора, классифицируемых правилом. Тестовый набор может быть получен генерацией большого количества конфигураций случайным образом – каждый из 149 битов конфигураций равновероятно принимает значение 0 либо 1.

Интересно отметить тот факт, что исследователи пытались построить правило, на котором $g_{L,r,½}$ принимает значение равное единице. В 1995 г. *M.Land* и *R.K.Belew* [25] (после неудачных попыток создать аналогичное правило), задались вопросом о самом его существовании. Результатом размышлений стала теорема о том, что такого правила при достаточно малом соотношении $\frac{r}{L}$ (пока клеточные автоматы сохраняют свойство локальности) не существует. Приведем формулировку этой теоремы формализованную согласно данному выше определению *DCT*. За доказательством можно обратиться к работе [25].

Теорема *(M.Land, R.K.Belew)***.** При заданных r, ρ^*, L, удовлетворяющих условию $L > \max(4r/\rho, 4r/1-\rho)$, не существует функции $f_{\max} \in 2^{2^{2r+1}}$, такой что $g_{L,r,\rho^*}(f_{\max}) = 1$.

В этой же статье авторы ставят ряд вопросов, наиболее интересный из которых: какого же значение $\max_{f\in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f)$, при *L=149, r = 3,* $\rho^* = ½$. Данный вопрос остается открытым. Сами авторы (на основе правила *GKL,* которое будет рассмотрего в настоящей работе), предположили, что $\max_{f\in 2^{128}} g_{149,3,½}(f) \approx 0.81$. Как показали дальнейшие исследования, это предположения не вполне верно. Забегая вперед, заметим, что $\max_{f\in 2^{128}} g_{149,3,½}(f) > 0.86$.

И так, не существует клеточного автомата полностью решающего задачу *DCT*. Следовательно, автоматы описываемого типа не достаточно "выразительны" (эффективны) для решения этой задачи. Поэтому крайне интересен результат работы [26], в которой показано, что пара простейших (единичного радиуса) клеточных автоматов, действующих последовательно, решают *DCT совершенно*. Приведем формулировку, доказанной им теоремы. Предварительно пронумеруем все автоматы радиуса один в порядке возрастания числа, двоичной записью которого служит таблица, задающая функцию $f(x,y,z)$ клеточного автомата. Автомат с номер *k* обозначим P_k. Так, правила, приведенные ниже, имеют номера 184 и 232:

$000 \to 0,\ 001 \to 0,\ 010 \to 0,\ 011 \to 1,$ $100 \to 1,\ 101 \to 1,\ 110 \to 0,\ 111 \to 1$	$000 \to 0,\ 001 \to 0,\ 010 \to 0,\ 011 \to 1,$ $100 \to 0,\ 101 \to 1,\ 110 \to 1,\ 111 \to 1$
Правило номер 184	Правило номер 232

Клеточные автоматы, порождаемые этими правилами, обозначаются P_{184} и P_{232} соответственно. Правило номер 184 (*if rule*), задается следующим булевым выражением $f_{184}(x,y,z) = yx \vee \neg yz$. Для правила номер 232 (*majority rule*), булево выражение выглядит следующим образом: $f_{232}(x,y,z) = xy \vee xz \vee yz$.

Теорема *(H. Fuks).* Пусть $\rho^* = ½$, *r*, *L* заданы. Положим $n = \left\lfloor \frac{L-2}{2} \right\rfloor$, $m = \left\lfloor \frac{L-1}{2} \right\rfloor$.

Тогда $\forall x \in 2^L : P_{232}^m(P_{184}^n(x)) = s$, где *s* — стационарное состояние:

a) "все нули" (0^L), если $\rho(x) < ½$.

b) "все единицы" (1^L), если $\rho(x) > ½$.

c) конфигурация, состоящая из чередующихся нулей и единиц ..0101.., если $\rho(x) = ½$ — нулей и единиц в ней поровну.

2.3. На сколько сложна *DCT*?

Возникает вопрос – насколько сложна *DCT,* и почему именно методы генетического программирования используются для ее решения. Не существует ли простого детерминированного и эффективного (например, полиноминального от (*L + r)*) алгоритма решения данной задачи? Этот вопрос **не был** обозначен в работах [4, 23–31]. Надо признать, что и в данной работе на него не дается ответа, но на основе работы [14] высказывается приводимая ниже гипотеза.

Гипотеза. При заданном значении $\rho^* \in [0, 1]$ функция $\max\limits_{f \in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f)$ невычислима за полиноминальное от *(L + r)* время. Соображениями в поддержку этой гипотезы являются утверждения следующих теорем.

Теорема 1 (*А.Л. Тоом, Л.Г. Митюшин*). Существует такой остров $x^0 \in X_1'$, например, остров с двумя единицами вида …001100…, что *невычислимо* по любому монотонному бинарному автомату *P* определить, размывает ли *P* остров x^0.

Теорема 2 (*А.Л. Тоом, Л.Г. Митюшин*). Существует такой монотонный бинарный автомат P_0, что *невычислимо* по любому острову $x \in X_1'$ определить размывает ли P_0 остров *x*.

Доказательства данных теорем приведены в работе [14]. Укажем лишь его идею, приведя формулировку основной леммы, на которую они опираются.

Лемма (*А.Л. Тоом, Л.Г. Митюшин*). Существует такое вычислимое отображение $\varphi(T)$, переводящее всякую машину *Тьюринга T* в монотонный бинарный автомат, и такое вычислимое отображение $\Psi_T(\pi)$, переводящее всякую программу π для *T* в остров $x \in X_1'$ (и зависящее от *T* как от параметра), что автомат $\varphi(T)$ размывает остров $\Psi_T(\pi)$ в том и только том случае, если машина *T* применима к программе π.

При этом пустая программа при всех T переводится в остров …001100… Вычислимость обоих отображений понимается, как рекурсивность функций в выбранной нумерации автоматов и островов и стандартной геделевской нумерации машин *Тьюринга* и их программ.

Всякому острову $x=(x_{-r},...,x_r)\in X_1'$ авторы приписывают номер, равный $x_0+\sum_{k=1}^{\infty}(2^{2k-1}x_k+2^{2k}x_{-k})$. Все бинарные автоматы нумеруют в порядке возрастания радиуса r, а при равных r – в порядке возрастания числа, двоичной записью которого служит таблица, задающая функцию $f(x_{-r},\ldots,x_r)$.

2.4. Правило Гача, Левина, Курдюмова (GKL) и обоснование его эффективности, как решения *DCT*

Авторы практически всех рассмотренных статей [4, 23–31], которые посвящены задаче классификации плотности, отмечают, что очень хорошее (классифицирующее примерно 81.6% из 10^6 случайно генерированных начальных конфигураций) правило было предложено *П. Гачем*, *Л. Левиным* и *Г. Курдюмовым* еще в 1976 г. (*GKL*-правило).

На рис.4 изображена зависимость процента распознаваемых *GKL* конфигураций $g_{149,3,½}(f_{GKL})$ от плотности конфигураций $\rho(x), x\in 2^{149}$.

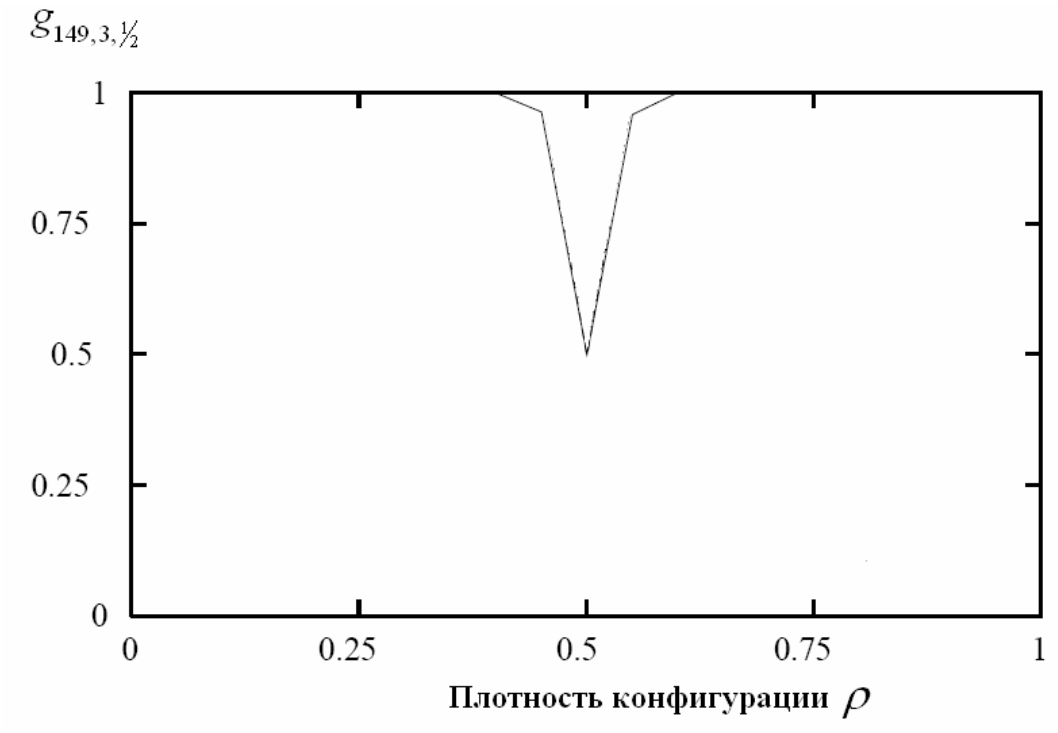

Рис. 4. Зависимоть процента распознаваемых GKL конфигураций от плотности конфигураций

В тоже время авторы указанного правила не анализируют, как оно работает, почему оно работает хорошо, и не уточняют, что в действительности оно было построено для решения другой задачи *apriori,* не имеющей отношение к *DCT*.

В статье [12] правило *GKL* было предложено в свете решения задачи о существовании *неэргодичной* одномерной однородной случайной среды с положительными вероятностями переходов. Этой же задаче посвящены публикации [15, 17]. Вслед за авторами работы [17] приведем ее формулировку, введя предварительно определение однородной одномерной случайной среды (однородного одномерного *недетерминированного* клеточного автомата).

Однородная по пространству одномерная случайная среда S_φ (где φ – функция переходов состояний элементов) – это бесконечная система автоматов $\{s_i\}$, где $i \in Z$, работающая в дискретном времени $t = 0, 1, 2, \ldots$ Состояние автомата s_i в момент времени t – обозначим его через s_i^t – есть элемент конечного множества X. Оно вероятностным образом определяется состояниями в момент $t-1$ автоматов $s_{i-r},\ldots,s_{i+r}$, где r – ***радиус среды*** есть некоторое натуральное число, а

именно, если $s_{i-r}^{t-1} = x_{-r}, \ldots, s_{i+r}^{t+r} = x_r$, то $s_i^t = y \in X$ с вероятностью $\varphi_y^t(x_{i-r}, \ldots, x_{i+r})$. При этом, если значения s_i^u при всех *i*, $u \le t-1$ заданы, то все s_i^t условно независимы.

Однородной одномерной случайной средой называется однородная по пространству одномерная случайная среда S_φ, вероятности переходов состояний элементов которой не зависят от времени, то есть $\forall t, u \in \mathrm{N}; y, x_i \in X : \varphi_y^t(x_{-r}, \ldots, x_r) \equiv \varphi_y^u(x_{-r}, \ldots, x_r)$.

Вероятностная мера μ на множестве состояний однородной случайной среды X называется ***инвариантной***, если она не изменяется со временем. Однородная одномерная случайная среда называется ***эргодичной,*** если у нее есть только одна инвариантная мера, иначе ***неэргодичной***.

Постановка задачи*:* существует ли неэргодичная одномерная однородная случайная среда с положительными вероятностями переходов $\varphi_y(x_{-r}, \ldots, x_r)$?

Долгое время считалось, что такой среды не существует, пока *Г.Л. Курдюмов* не доказал ее существования, приведя пример такой среды. **Идея доказательства**: сначала строится (весьма сложная) *детерминированная* среда (клеточный автомат), с двумя неэквивалентными притягивающими состояниями x_1, x_2 (также непростыми, не стационарными и даже не периодическими). Затем, на полученную среду накладывают малый (но не нулевой) шум, получая случайную среду с положительными вероятностями переходов. Потом, исходя из x_1 и x_2, строятся инвариантые меры μ_1, μ_2.

Однако приведенная среда оказалась очень сложной и не наглядной. Возникла идея построения значительно более простой неэргодичной среды. Для ее построения в работе [12] и была создана *детерминированная* среда, называемая в данной работе *GKL*. Данная среда является бинарным клеточным автоматом радиуса три, с простым правилом $f(x_{-3}, x_{-2}, x_{-1}, x_0, x_1, x_2, x_3) = x_0(x_{-3} + x_{-1}) + \overline{x_0} x_1 x_3$. Другими словами, когда $x_0 = 1$, значение x_0 в следующий момент времени определяется “голосованием”: это будет значение, преобладающее среди x_0, x_{-1},x_{-3}. Аналогично, при $x_0 = 0$, “голосуют” x_0, x_1, x_3. Введя булевы функции

$IF(x,y,z) = xy + \bar{x}z$, $M(x,y,z) = xy + xz + yz$, можно записать *GKL* следующим образом $f(x_{-3}, x_{-2}, x_{-1}, x_0, x_1, x_2, x_3) = IF(\ x_0,\ M(x_0, x_{-1}, x_{-3}),\ M(x_0, x_1, x_3)\ )$.

Почему *GKL* – хорошее решение *DCT*? Потому, что *GKL* является размывающим автоматом. Он любую конфигурацию с конечным числом единиц (остров) за конечное время переводит в конфигурацию “все нули”. Кроме того, он симметричен в том смысле, что и любая конфигурация с конечным числом нулей переводится за конечное время в состояние “все единицы”. Это свойство доказывается в работе [12], а в более популярном изложении в работе [16]. Состояния “все нули” и “все единицы” являются стационарными притягивающими, других притягивающих состояний у *GKL* не существует. Сохраняется ли свойство размывания, когда мы определяем *GKL* **не** на бесконечной одномерной решетке 2^Z, а на окружности 2^L как в *DCT*?

Теорема *(Л.А. Левин)* [22]. Пусть $\alpha = 1/y$, где y – корень уравнения $5^y = 2^y + 1$. Пусть автомат *P*, задаваемый правилом *GKL,* определен на окружности длиной $L = 3k^\alpha$. Рассмотрим произвольную начальную конфигурацию $x = (x_1, ..., x_L) \in 2^L$, для которой $\rho(x) \le \frac{k}{L}$ $(\rho(x) \ge 1 - \frac{k}{L})$, то есть единиц (нулей) в ней не более *k. Тогда* $P^L x = s$, где s – стационарное состояние “все нули” (“все единицы”).

2.5. Существующие подходы к решению *DCT*

Перечислим (табл.1) в порядке возрастания результативности методы генетического программирования (и не только), применявшиеся для решения *DCT*. Укажем их основные достоинства и недостатки.

Таблиаца 1. Подходы к решению *DCT*

Алгоритм и авторы	**Год**	**Таблица истинности правила**	**Результат на 10^6 тестах**
Conventional GA: Michell, Das, Crutchfiled	1994	00000101000001000000010110000111000001010000 00000000111101110111000000110111011101010101 10000011011110111111111111011011101111111	**76.9**

Алгоритм и авторы	Год	Таблица истинности правила	Результат на 10^6 тестах
GKL (без ГП)	1978	00000000010111110000000001011111000000000101111100000000010111110000000001011111111111110101111100000000010111111111111101011111	**81.6**
Davis (без ГП)	1995		**81.8**
Das (без ГП)	1995	00000111000000000000011111111111000011110000000000000011111111111100001111000000000000011111111111000111001100010000111111111111	**82.1**
GP: Koza, Andre, Bennet	1996	00000101000000000101010100000101000001010000000001010101000001010101010111111111010101011111111101010101111111110101010111111111	**82.3**
GEP: Ferreira	2002	00000000010101010000000001110111000000000101010000000011101110000111101010101000011110111011111111111101010101111111110111011	**82.55**
Conventional CA c коэволюцией и разделением ресурсов: *Juille, Pollack (JP)*	1998	00010100010100010011000001011100000000000010100001100111001011111000101110001000111111111010111110000111101010011110011110101111	**86.0**

2.5.1. Традиционные генетические алгоритмы (*Conventional GA*)

Подход предложен *M. Mitchell, P. T. Hraber, and J. P. Crutchfield* в работе [23]. За счет представления решений в виде таблиц истинности искомых правил не накладывает ограничений сверху на сложность выводимых решений, а, следовательно, *теоретически* позволяет найти оптимальное правило. Однако подход оказался крайне неэффективным – исследователями так и не было найдено правило лучшее, чем *GKL*. **Причины** малой эффективности:

a) слишком сложное (в виде таблиц истинности) представление решений, что значительно увеличивает время поиска хороших начальных приближений;

b) не используются такие эффективные техники генетического программирования как коэволюции (*coevolution*), разделение ресурсов

(*resource sharing*), распределенное генетическое программирование (*parallel genetic algorithms, island model genetic algorithms*).

2.5.2. Генетическое программирование (*GP*)

Данный подход, предложенный D. *Andre, F.H. Bennet, J.R. Koza* 1996 г. [31], позволил найти лучшее решение, чем какое-либо ранее известное (в том числе *GKL*). Хромосомы представляются в виде деревьев, узлами которых могут быть функции и терминалы. Его основными **преимуществами** при решении задачи *DCT* являются:

a) представление решений в виде деревьев булевых формул. Узлом дерева может быть одна из 26 определенных булевых функций или один из 26 определенных терминалов. Такое представление решений более выразительно, нежели таблицы истинности правил. Деревья булевых формул могут иметь произвольную структуру (и высоту), что позволяет за счет маленьких деревьев быстро, по сравнению с методом conventional GA, находить хорошие приближения и в дальнейшем их улучшать (уже за счет больших деревьев). Таким образом, эволюция сама “определяет” наиболее эффективный способ представления решений;

b) используются эффективные методы генетического программирования, такие как распределенные генетические алгоритмы [32], автоматически определяемые функции – поддерево узла–функции, присутствующее в большинстве хороших решениях, а, следовательно, обладающих некоторыми хорошими свойствами. В дальнейшем, такие поддеревья выделяются в функции соответствующей арности и участвуют в построении деревьев наряду с предопределенными (на этапе компиляции) функциями.

К основным **недостаткам** *GP* относятся:

a) по-сравнению с *GEP* (где высота деревьев ограничена) – медленный поиск хороших решений, ввиду сложности их представления;

b) **общая проблема для *GP*** (безотносительно к *DCT*) – необходимость проверки корректности генетических операций, осуществляемых над хромосомами – деревьями. Поэтому имеет место большое расходование вычислительного времени “впустую”. Так, не каждая рекомбинация и мутация родительских особей (деревьев) приводит к появлению корректных потомков правильно составленных деревьев, в которых у каждого узла-функции количество узлов-потомков совпадает с арностью функции. Например, узел-функция может в результате мутации заменяться только на функцию такой же арности;

c) не используются коэволюции и разделение ресурсов.

2.5.3. Программирование с экспрессией генов (*GEP*)

Создатель данной концепции генетического программирования – *Candida Ferreira* [4]. Хромосомы в этом методе представляются в виде так называемых *Karva*-деревьев, которые могут быть легко линеаризованы – записаны в массив (некоторый аналог – структура данных *heap*).

Название метода отражает идею его работы, заимствованную из биологии: существует первичная структура хромосом – линеаризованные *Karva*-деревья, которые в последствии экспрессируются в обычные (не линеаризованные) деревья – вторичную структуру. Интересно также, что некоторые участки первичной структуры не экспрессируются. Таким образом, одной вторичной структуре (дереву) может соответствовать несколько первичных (массивов). Аналог в биологии: последовательность нуклеотидов – первичная структура белка. В дальнейшем она экспрессируется (транслируется на рибосомах) в последовательность аминокислот – вторичную структуру белка. При этом последовательность нуклеотидов содержит как транслируемые участки – *экзоны*, так и не транслируемые – *интроны*, удаляемые в процессе *сплайсинга*.

Преимущества данного подхода:

a) **главное достоинство** *GEP – Karva*-деревья обладают существенным преимуществом перед обычными деревьями (используемыми в *GP)* – все генетические операции с ними корректны по определению. Таким образом, *GEP* алгоритм не тратит время на проверку корректности деревьев, получаемых в результате рекомбинаций и мутаций. При этом отметим, что выполнять генетические операции над линеаризованными деревьями не сложнее и не дольше, чем над битовыми строками;
b) мультигенная структура хромосом – хромосома состоит из нескольких генов;
c) наличие незначащих мутаций в интронах (областях, не экспрессирующихся в деревья).

2.5.4. Применение традиционных генетических алгоритмов наряду с поддержкой коэволюций и разделяемых ресурсов

Идея коэволюций, реализованная *H.Juillie и J. B. Pollack* применительно к *DCT* в 1998 г. в работе [28], оказалась крайне эффективной и позволила найти наилучшее известное на текущий момент решение задачи классификации плотности. **В чем заключается** эта идея?

Когда нет возможности точно вычислить функцию приспособленности (так в *DCT* всего 2^{149} возможных конфигураций), приходится создавать некоторый набор репрезентативных тестов – таких, чтобы значение *приближенной* функции приспособленности, вычисляемой на этих тестах, как можно более точно соответствовало ее *реальному* значению. Решения задачи в процессе эволюции постепенно адаптируются к выбранным тестам, и они перестают быть репрезентативными – значение приближенной функция становится бо́льшим, чем значение реальной. Чтобы избежать этого неприятного эффекта автор идеи коэволюций *W. Daniel Hillis* в 80-х годах предложил наряду с популяцией решений создать эволюционирующую популяцию тестов. Функция

приспособленности определенная *на множестве решений* прямо пропорциональна количеству тестов, которые “проходит” это решение, а функция приспособленности *на множестве тестов* прямо пропорционально *количеству решений*, которые не проходят данный тест. Это позволит тестам постоянно совершенствоваться, становясь все сложнее для решений.

Заметим, что эту идею *Hillis* предложил при поиске решения задачи построения минимальной (по количеству логических элементов) сортирующей сети, рассмотренной *Д. Кнутом*.

Однако представление решений в виде таблицы истинности правил обладает указанными выше недостатками, что “тормозит” процесс эволюции. Так на нахождение решения исследователям потребовалась неделя при размере популяции 1000.

ГЛАВА 3. КОМПОЗИТНОЕ ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

3.1. Постановка проблемы

При анализе работы программ реализующих *GA*, *GP* и *GEP* для решения *DCT,* была выявлена **главная проблема:**

- при “слишком сложной” структуре хромосом (*GA M.Mitchell* – в виде таблиц истинности, представленных битовыми строками длины 128), а также *GP J.Koza* (деревья, в которых 26 функций и 26 терминалов) б*о*льшая часть вычислений проходит впустую, и крайне медленно появляются хорошие решения. В то же время наличие хороших начальных приближений крайне важно для эффективной работы эволюционных алгоритмов;
- при “слишком простой” структуре (*GEP C.Ferreira* – хромосомы длины 13 всего с двумя функциями *IF, M* и семью терминалами) очень быстро находятся хорошие начальные приближения, но они не могут быть улучшены в силу недостаточной выразительности хромосом. Так, по-видимому (автором данной работы были проведены эксперименты), не существует правила, классифицирующего более *83%* конфигураций, которое имеет короткую булеву формулу.

Для решения поставленной проблемы автором настоящей работы предложен *простой* и в то же время весьма *эффективный* (как показала реализация применительно к *DCT*) подход, названный “Композитное генетическое программирование”. Помимо решения этой проблемы, предлагаемый подход обладает рядом других достоинств. Обозначим сначала основную его идею, а затем приведем описание.

3.2. Основная идея

Основная идея предлагаемого метода – позволить различным генетическим алгоритмам (*GEP, GP, GP + Island Model, GA, GA + Coevulution + Resource Sharing,…*), решающим одну и ту же оптимизационную задачу (например, *DCT*) взаимодействовать между собой. Предлагается создать многокомпонентную систему с *репозиторием,* в котором хранятся решения задачи в *универсальной форме,* оцениваемые *универсальной функцией приспособленности.* Применительно к *DCT:*

- универсальная форма решений – таблицы истинности правил, представленные битовыми строками фиксированной длины;
- универсальная функция – вероятность распознавания данным правилом случайной тестовой конфигурации, вычисляемая статистически на большом наборе тестовых конфигураций.

Клиентские программы, реализующие различные методы решения задачи, представлены отдельными *модулями*. Они периодически отдают серверу наилучшие из своих решений и запрашивают решения из репозитория по определенному *критерию* (например, чуть лучшие, чем их собственные). Форма представления решений на клиентских *модулях* может существенно отличаться от *универсальной формы*, но должна быть к ней приводима – существует некоторая *функция преобразования решений*, отображающая решения различных *модулей* в *универсальную форму*. Желательно, чтобы она была обратима и “быстро” вычислима (например, линейно по длине представления решения). *Универсальная функция приспособленности* – целевая функция решаемой оптимизационной задачи. Она также может существенно отличаться от функций приспособленности модулей.

На рис. 5 показана примерная схема композитного генетического программирования.

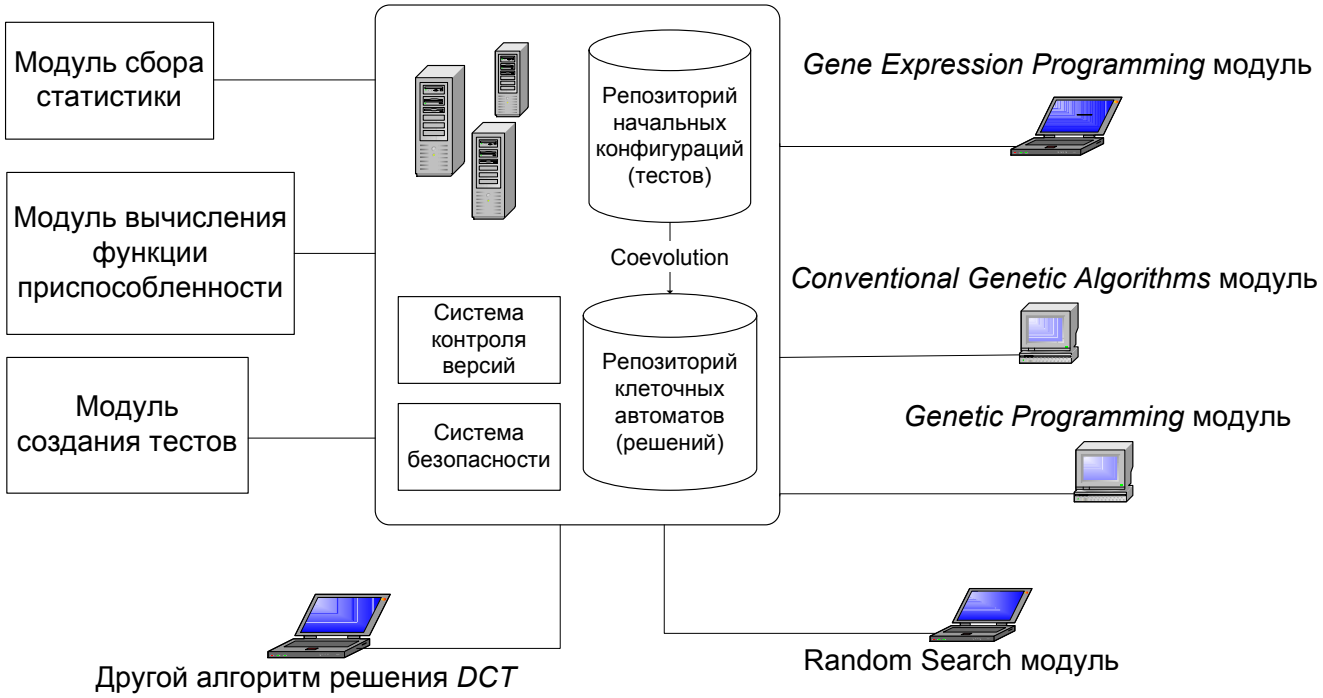

Рис. 5. Схема композитного генетического программирования

3.3. Преимущества предлагаемого подхода и его описание

Перечислим достоинства предлагаемого подхода.

1. *Модуль* с простым представлением решений через *репозиторий* передает быстро найденные хорошие решения другим *модулям*, обладающим более выразительным представлением решений, которые способны улучшить любые хорошие решения, но не могут быстро их найти. Так, в *DCT* метод *GEP* с простым представлением хромосом позволяет быстро находить хорошие решения и передавать их методам *GP* или *GA* для дальнейшего улучшения. Самостоятельно методы *GP* и *GA* не способны быстро находить хорошие решения.
2. Если при работе генетического алгоритма в популяции появляется особь со значением приспособленности много превосходящим приспособленность остальных особей популяции, то *разнородность* (*diversity*) такой популяции (определяемая, например, как среднеарифметическое расстояний Хэмминга

между всеми парами хромосом популяции) быстро убывает, что приводит к резкому уменьшению производительности генетического алгоритма. Обоснование этого факта приводится в работе [33]. Поэтому, *модулю* целесообразно периодически запрашивать решения, которые “чуть лучше” наилучших из его собственных решений, а дальше пытаться самостоятельно эволюционировать. Кроме того, *модуль* с целью повышения разнородности своей популяции может, предоставить *репозиторию* свою популяцию, запросить решение, наиболее сильно отличающееся от его собственных решений, но обладающее примерно такой же приспособленностью. Существуют и другие варианты: *модуль* может эволюционировать полностью самостоятельно (без общения с *репозиторием*) до тех пор, пока идет оптимизационный процесс – значение функции приспособленности постепенно увеличивается. Как только модуль не способен улучшить решения самостоятельно – он обращается за помощью к *репозиторию*, заодно предоставляя ему наилучшие из своих решений.

3. Предлагаемая схема позволяет “объединить” для решения одной задачи различные, как по способу представления хромосом, так и по способу задания функции приспособленности алгоритмы – *модули*. Применительно к *DCT:*
   - разный способ задания хромосом в *DCT* имеют все три алгоритма: *GA* – строки фиксированной длины*, GP* – деревья*, GEP* – *Karva*-деревья (линеаризованные деревья)*;*
   - функции приспособленности также различны – при использовании коэволюции функция приспособленности меняется (вместе с популяцией начальных конфигураций) во времени. Без коэволюции также существует несколько способов ее задания: процент распознанных тестовых конфигураций, сгенерированных один раз в начале работы алгоритма или процент распознанных тестовых конфигураций, генерируемых на каждом шаге эволюции. Кроме того, тестовые конфигурации можно генерировать различными “случайными”

способами – равномерно над каждым битом или равномерно над плотностью конфигурации.

Поскольку решения в *репозитории* могут представляться в другом виде, нежели в *модуле,* необходимо существование *функции преобразования решений*. Эта функция может вычисляться как на клиенте *(модулем)*, так и на сервере *(репозиторием)*. Так в *DCT* решения, полученные методами *GP* и *GEP* (деревья), преобразуются в таблицы истинности правил (битовые строки фиксированной длины). Преобразование здесь выполняется вычислением значение булевого выражения, представимого деревом, на всевозможных значениях терминалов $x_{-r},...,x_r; x_i \in \{0, 1\}$. Так *GKL* правило, полученное *GEP* в виде $IF(\ x_0,\ M(x_0, x_{-1}, x_{-3}),\ M(x_0, x_1, x_3)\ )$, преобразуется в битовую строку:

```
00000000010111110000000001011111000000000101111100000000010111110000000001011
11111111111010111110000000001011111111111101011111
```

4. Естественным образом получаем *распределенную систему вычислений*, к которой в любое время могут быть подключены как совершенно разные *модули*, так и по нескольку экземпляров одинаковых модулей. Как показано в работе [32] распределенные генетические алгоритмы (решение задачи ищется сразу в нескольких независимых небольших популяциях, между которыми иногда осуществляется *миграция* хороших решений) зачастую имеют сверхлинейную производительность по той причине, что разные популяции часто идут в направлении различных локальных экстремумов, а миграции между ними позволяет улучшать независимо найденные решения. При этом общая разнородность, считаемая на объединении всех популяции, остается на более высоком уровне, чем в случае нераспределенного подхода – когда поиск решения выполняется в одной популяции большого размера.
5. Простая и удобная поддержка *коэволюций*, преимущества которых описаны в работах [27–29]. На сервере может одновременно находиться несколько *репозиториев*: решений, тестов для решений, тестов для тестов и т. д. Так в *DCT* имеются два *репозитория*: решений (правил клеточных автоматов) и

тестов (начальных конфигураций, на которых тестируются правила). Существует возможность разделения ролей по *модулям*. Одни *модули* могут специализироваться только на поиске решений, другие на составлении тестов к решениям, третьи (что *важно*, так как это наиболее трудоемкая по времени операция) – *только* на вычислении *универсальной функции приспособленности* на основе решений и тестов, найденных другими *модулями*. В подходе, предложенном в работе [28], важную роль в вычислении функции приспособленности тестовых конфигураций играет статистика $E_{i,j} = (R_i, \rho(IC_j)) = \ln(2) + p\ln(p) + q\ln(q);\ 0 \le p,q \le 1$ (доопределенная $\ln(2)$ при $p = 0$ и $q = 0$). В ней R_i – некоторое правило, IC_j – некоторая начальная конфигурация, q – вероятность (вычисляемая статистически) того, что правило R_i распознает конфигурацию с плотностью $\rho(IC_j)$, p=1–q. Данная статистика отражает "покрытие решений тестами" – показывает, для каких решений существуют хорошие тесты, а для каких нет, тем самым помогая оценивать "качество тестов". Напомним, что на множестве тестовых конфигураций задана *функция приспособленности*, статистика E – участвует в задании этой функции. Такого типа статистики, могут собираться отдельными *модулями* на основе репозиториев решений и тестов и сохраняться в репозитории. Затем модули, запрашивая из репозитория решения и тесты, смогут получать также и статистики, к ним относящиеся, и не тратить время на самостоятельное их вычисление. При наличии таких статистик отдельный *модуль*, предоставив серверу информацию о своих решениях, имеет возможность запросить у него "хорошие" тесты для этих решений.

6. *Репозиторий* выступает и как надежное централизованное хранилище найденных решений для большой и сложной распределенной задачи и как основа системы контроля версий. К нему в любой момент времени может подключиться *модуль* и "сохранить", найденные им решения (по своему

идентификатору), которые позже могут быть им (по этому же идентификатору) полностью или только частично загружены.

7. К решениям, добавляемым в *репозиторий,* целесообразно добавлять метаданные (атрибуты): каким *модулем* данное решение было добавлено в репозиторий, дата и время этого добавления. Тогда получаем сразу несколько полезных возможностей:
   - возможность собирать статистику: *показатель эволюции*, определенный, например, как скорость роста функции приспособленности на лучших решениях *репозитория*, разнородность решений *репозитория* и т.д. В *DCT* важную роль показатели: средняя плотность правил – хорошие правила обладают плотностью близкой к 0.5; средняя плотность тестовых конфигурации – чем больше этот показатель, тем лучше набор тестовых конфигураций; число беспорядков правила – чем меньше, тем “монотоннее” правило и обычно, тем проще булева формула, его задающая. На основе статистик можно делать заключения о том происходит ли постепенная максимизация универсальной функции приспособленности, как быстро она идет, какие *модули* вносят наибольший вклад, какие *модули* не предлагают хороших решений и могут быть отключены с оповещением о нерезультативности. Статистика может собираться как непосредственно на сервере, так и отдельными клиентскими *модулями,* специализирующимися именно на ее сборе, а не реализующими генетические алгоритмы*;*
   - основываясь на дате добавления решения, можно организовать *систему контроля версий:* возможность сравнивать репозитории на различные даты, “*откатывать*” *(rollback)* на конкретную дату, просматривать историю *“добавлений” (commit)* различными модулями. Возможность “отката” требуется, например, если изначально *универсальная функция приспособленности* была определена не точно, и можно потерять хорошие решения (размер *репозитория*, как хранилища решений ограничен). Либо до определенного момента происходила эволюция, как

в смысле начальной (неточной) функции приспособленности, так и в смысле уточненной, а с некоторой даты только в смысле начальной – тогда требуется откатить репозиторий на эту дату и, уточнив функцию, продолжить работу генетического алгоритма. Еще один пример, при котором необходим откат – сбой в работе системы в результате нарушения системы безопасности одним из модулей;

- на основе метаданных решений, хранимых сервером, легко организовать соревнование (турнир) между различными *модулями* (как алгоритмами, представляемыми различными людьми или организациями) в решении сложной распределенной задачи, что стимулирует участников и зачастую значительно ускоряет решение самой задачи.

8. В предлагаемой системе легко реализуется обеспечение безопасности *(security)* и разделение доступа. Каждый *модуль* может осуществлять как *анонимный* доступ к системе, так и *авторизованный* (по уникальному идентификатору и паролю). Первый подход применим в случае, если модуль предоставляет *репозиторию* решения, оцениваемые *универсальной функцией приспособленности* (вычисленной на сервере) как “очень хорошие”. В таком случае неважно, кто предоставил решения, главное, что *репозиторий* самостоятельно убедился в их “качестве”. Другое дело – если модуль просит *репозиторий*, добавить решения, на которых значение *универсальной функции приспособленности* уже вычислены самим *модулем*. Перепроверка этих значений на сервере – затратная по времени операция, в то же время, если *модулем* была допущена ошибка при вычислении функции приспособленности, то в репозиторий попадут плохие решения, замещая, в силу ограниченных размеров репозитория, хорошие, что приведет (в лучшем случае) к замедлению процесса эволюции. Еще более наглядные примеры – *модули*, специализирующиеся на вычислении *универсальной функции приспособленности* и на сборе различных статистик. На основе работы таких *модулей*, *репозиторий* отдает решения, запрашиваемые другими *модулям*, и принимает решения от них.

Предлагается определить *несколько ролей* в системе и привязать к каждой *роли* набор *прав*. Каждый *модуль* может иметь несколько ролей. Возможные *роли*: “администратор системы”, “сборщик статистики по решениям (тестам)”, “вычислитель *универсальной функции приспособленности* для решений (тестов)”.

3.4. Реализация метода

Для реализации предложенного метода был выбран язык *Java* и технология *Remote Method Invocation (RMI)*. На рис. 6, 7 приведена упрощенная диаграмма классов. На ней отображены только основные интерфейсы и классы, а также ключевые их методы. Не показаны классы, отвечающие за поддержку коэволюции, между классами отображено только отношение наследования. После диаграммы следует краткое описание назначений интерфейсов.

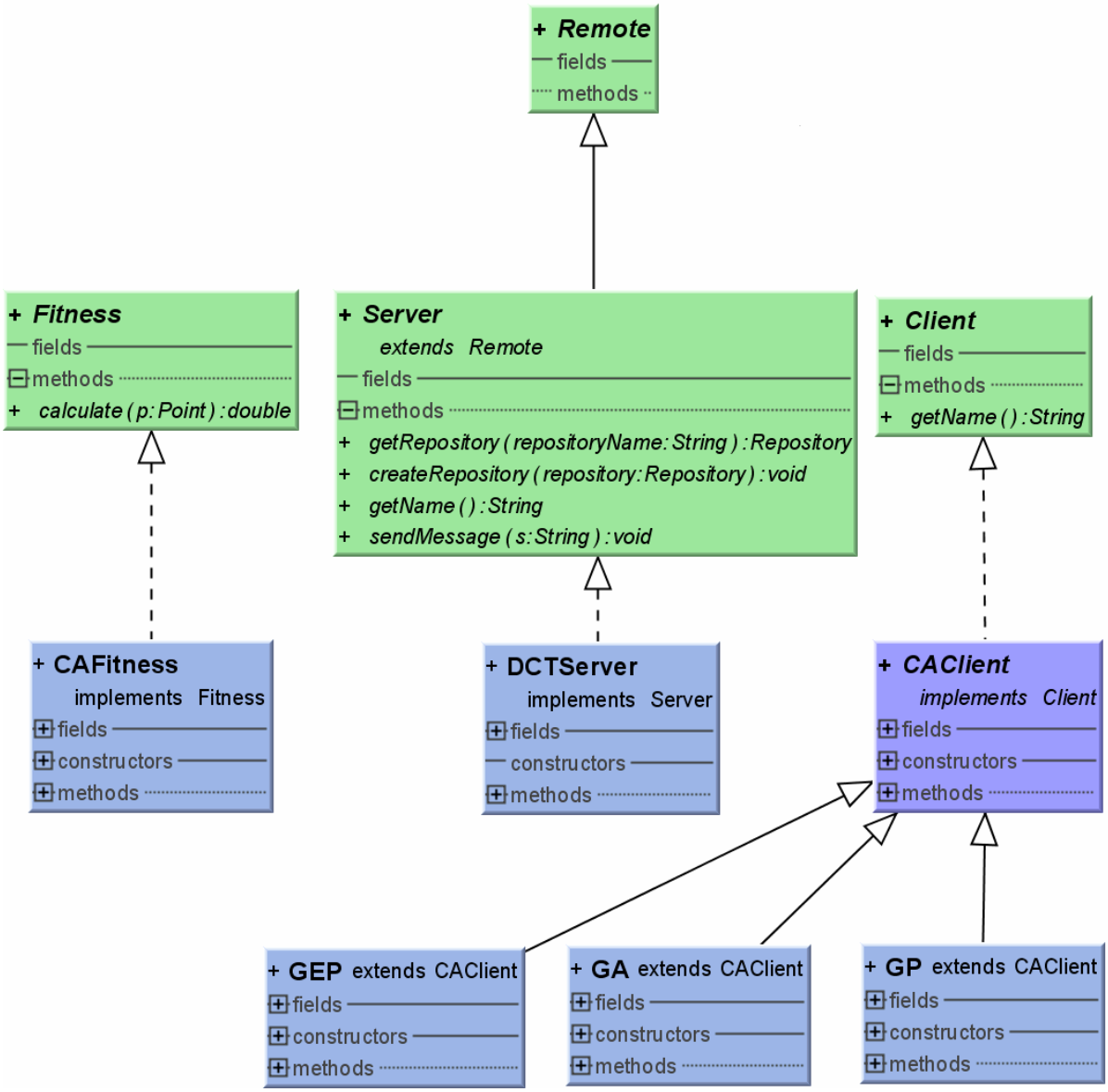

Рис. 6. Диаграмма классов

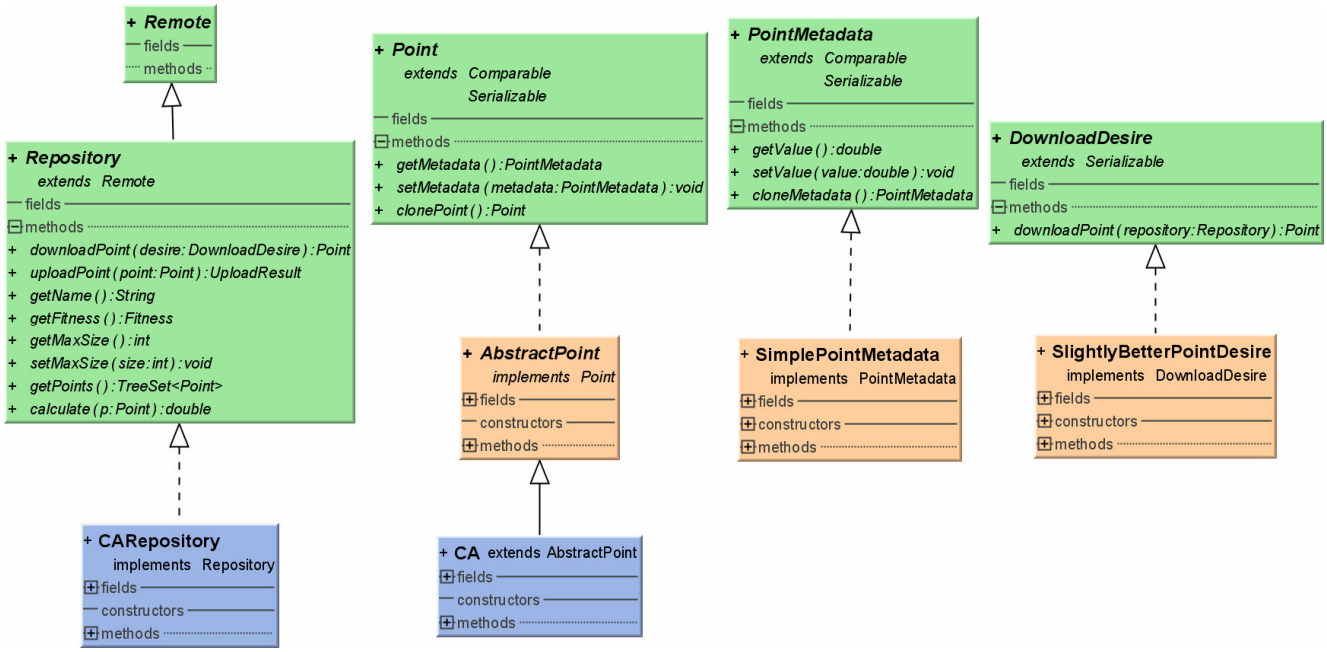

Рис. 7. Диаграмма классов (продолжение)

Зеленым цветом отмечены интерфейсы. Розовым – абстрактные классы и конкретные имплементации интерфейсов. Они не привязаны к *DCT* и могут быть использованы при имплементации решения любой другой задачи методом *композитного генетического программирования*. Синим цветом обозначены классы, реализующие интерфейсы для *DCT*. Приведем краткое описание назначений интерфейсов и некоторых классов:

`Server` – сервер, хранящий репозиторий;

`Client` – модуль, содержащий алгоритм решения задачи. Его конкретизации для задачи *DCT* – классы **`GEP`**, **`GA`**, **`GP`** реализуют соответствующие их названиям алгоритмы;

`Fitness` – функция приспособленности, вычисляемая на решениях задачи;

`Point` – решение задачи, элемент популяции;

`Repository` – репозиторий решений;

`PointMetadata` – метаданные решений: значение функции приспособленности решения, дата добавления в репозиторий, имя модуля, добавившего данное решение и т. д.;

`DownloadDesire` – запрос (“пожелание”) модуля при обращении к репозиторию для получения решений.

3.5. Результаты работы метода композитного генетического программирования в *DCT*

Атором настоящей работы был проведен ряд экспериментов с **целью**:

- *проверки эффективности* метода композитного генетического программирования;
- поиска правила, решающего *DCT лучше, чем любое из известных* на сегодня правил. То есть лучше *JP*–правила, котрое распознает около 86% конфигураций.

Характеристики компьютера, на котором выполнялись эксперименты – ноутбук с процессором *Intel Pentium M* 1.86 *GHz*.

Результаты экспериментов и их анализ: за несколько часов работы программы, реализующей композитное генетическое программирование, было получено много хороших правил. Приведем некоторые из них. Значение поля “Результат” табл.2, получено тестированием правила на 10^5 начальных конфигураций.

Таблица 2. Таблицы истинности

Таблица истинности правила	**Результат**
`00000001000101000001000011010111000000010000111100111001010101110000110101110101111111110001011111010001001111011111100101010111`	**0.837**
`00000001000101000011000011011111000000010000111100111001001000101110000110101110101111111110001011101110001001111011111100101010111`	**0.833**
`00000001000101000011000011010111000000010000111100111001010101110000111011101001111111110001011101110001001111011111100101010111`	**0.831**
`00000000000001010011001100100111000000000101010100110011011101110000111100001010000111100100111111101010101111111101110111`	**0.826**

Таблица истинности правила	Результат
0000000100010100000000001101011100000001000011110011100101010111 0000110110110100111111110001011111110001001111111111100101010111	**0.826**

Наилучшего правила, как следует из данной таблицы, найдено **пока не было. Однако** ни в работе [31], ни в работе [4] не было получено аналогичных результатов, не говоря уже о том, что в работе [31] популяция состояла из 51200 правил (против 100 в экспериментах данной работы), а вычисления осуществлял 64 процессорный компьютер, занимавшийся этой задачей более месяца. В работе [28], поиск наилучшего из известных на сегодня правила с результатом 86% выполнялся в течение недели, а популяция решений содержала 1000 правил. На основе этих данных заключаем, что **метод композитного генетического программирования является весьма эффективным в** *DCT*. Есть основания полагать, что он является эффективным и для решения других задач, где применимо генетическое программирование. Поиск примеров, подтверждающих данное предположение – предмет дальнейших исследований.

Сравнительная эффективность методов решения *DCT* приведена в табл.3.

Таблица 3. Эффективность методов решения *DCT*

Метод	Время	Размер популяции	Результат, %
Стандартный *ГА* с коэволюцией и разделением ресурсов (Juille, Pollack)	около недели	1000	86.0
Композитное генетическое программирование	несколько часов	100	83.7
Программирование с экспрессией генов (Ferreira)	несколько часов	50	82.55
Генетическое программирование (Koza, Andre, Bennet)	около месяца	51 200	82.33

Почему не было получено правила лучшего, чем *JP?* Возможны следующие причины:

1. Лучшего решения задачи *DCT*, чем *JP* не существует.
2. Программа не запускалась на достаточно долгое время, так как главной задачей являлась проверка эффективности предлагаемого метода. В случае запуска на более мощном компьютере на более длительное время *возможно* получение лучших результатов.
3. Неверно реализован метод коэволюции и разделения ресурсов, который используется во всех трех модулях (*GA, GP, GEP*).
4. Ошибка в реализации предлагаемого метода композитного генетического программирования.
5. Предлагаемый метод недостаточно эффективен для получения наилучшего решения данной задачи.

Параметры эксперимента: радиус правила – три; длина одномерного бинарного клеточного автомата – 149. Сервер содержал *репозиторий* правил клеточных автоматов – решений *DCT*. Размер репозитория – 500. Репозиторий хранил правила в виде их таблиц истинности – битовые строки длиной 128. *Универсальная функция приспособленности* – процент распознаваемых правилом тестовых конфигураций (битовых строк длиной 149) из 10^5 сгенерированных случайно (каждый бит равновероятно принимает значение 0 или 1). С репозиторием работали три *модуля*, каждый из которых поддерживал коэволюцию с разделением ресурсов, и имел следующие параметры.

Для правил:

- количество правил – 100;
- двухточечная мутация с вероятностью 0.2;
- вероятность рекомбинации – 0.7;
- использовался *элитизм (elitism)*: 20% лучших правил переносились в следующую популяцию без изменений. Еще 5% генерировались случайным (специфичным для конкретного *модуля*) способом. Оставшиеся 75%

отбирались пропорционально их приспособленности (*roulette wheel selection*).

Для тестов:

- количество тестов –100;
- мутация отсутствовала;
- рекомбинация отсутствовала;
- отбор осуществлялся пропорционально функции приспособленности – 95% тестовых конфигураций переносилось в следующую популяцию без изменений. Оставшиеся 5% генерировались случайно – равновероятно над плотностью конфигурации.

Описание модулей:

1. *GA-модуль,* решающий задачу традиционными генетическими алгоритмами. Представление решений – битовые строки длиной 128.
2. *GP-модуль*, реализующий метод генетического программирования на деревьях с единственной функцией *IF* и количеством терминалов равным семи.
3. *GEP-модуль,* реализующий метод генетического программирования с экспрессией генов. Набор функций: *IF*, *M*. Длина *головы* гена – четыре, общая длина – 13. Хромосомы имеют мультигенную структуру: состоят из трех генов, соединенных функцией *IF*. Длина хромосомы – 39.

ЗАКЛЮЧЕНИЕ

1. Проанализирована возможность применения генетического программирования для создания автоматов. Установлено, что в ряде интересных задач, где решения представимы в виде автоматов, метод генетического программирования дает лучшие результаты, чем другие подходы. Это следующие задачи:
   - итерированная дилемма узников (Iterated prisoner's dilemma);
   - задача про "умного" муравья (Artificial and problem);
   - задача классификации плотности для клеточных автоматов (Density classification task for cellular automata);
   - задача синхронизации для клеточных автоматов (Synchronization task for cellular automata).
2. Выбрана, формализована, подробно изучена наиболее трудная, интересная и интенсивно изучаемая в последнее десятилетие проблема – задача классификации плотности для клеточных автоматов. Данной проблеме посвящен ряд публикаций [4, 23–31]. В основном задача классификации плотности для клеточных автоматов рассматривается, как *benchmark* для различных направлений, идей, подходов и концепций эволюционного программирования, но она представляет также и интерес для теоретических исследований в области клеточных автоматов и мультиагентных систем.
3. Выяснено и обосновано, почему правило, предложенное *Гачем, Курдюмовым, Левиным в 1976 г. (GKL правило)* [12] "хорошо" ($g_{149,3,½} \approx 0.816$) решает *DCT*.
4. Проанализирована работа *А.Л. Тоома* [14], в которой доказываются два результата о невычислимости для одномерных бинарных автоматов – операторов $P: X_n \to X_n$, действующих на пространстве X_n, состоящем из всех бесконечных в обе стороны последовательностей, члены которых – целые числа от 0 до n . $X_n = S_n^Z = \{x\}$, $x = (x_i)$, $x_i \in S_n = \{0,\ldots,n\}$, $i \in Z$. Данные результаты имеют отношение к задаче классификации плотности. На их основе высказана

гипотеза о том, что даже в случае $X_n,\ X_n = S_n^Y,\ Y = [0, r],\ r \in Z^+$ задача "трудна" – не существует полиноминального (от $L + r$) алгоритма ее решения и, следовательно, применение генетического программирования для ее решения *является обоснованным*.

5. Проанализированы свойства автоматов, "достаточно хорошо" (не хуже *GKL* автомата, предложенного *Гачем, Левиным, Курдюмовым* [12]) решающих задачу классификации плотности. Установлены, главным образом эмпирически, свойства их таблиц истинности. Установленные свойства использовались для задания хорошей функции приспособленности (обеспечивающей быструю эволюцию) модулей при реализации метода композитного генетического программирования. Перечислим их:
   a) плотность (отношение количества единиц к длине таблице истинности) близка к значению ½. Правила с плотностью близкой к этому значению, получали (при прочих равных) большее значение функции приспособленности;
   b) число беспорядков мало по сравнению с числом беспорядков таблицы истинности случайного автомата (каждый бит таблицы истинности которого равновероятно принимает значение 0 либо 1). Этот параметр отражает "сложность правила". Функция *GEP-модуля* имела большее (при прочих равных) значение на сложных правилах, в то время как *GP* и *GA* модули стимулировались к поиску простых функций. Их функции приспособленности имели на сложных правилах меньшее значение, чем на простых;
   c) обладают свойством консервативности. Более интересно, что количество неэквивалентных притягивающих состояний у "хороших автоматов" мало. В частности, у *GKL* их всего два. Правила с меньшим количеством (но не меньшим двух) консервативных состояний "премировались" дополнительным неотрицательным слагаемым функции приспособленности;

d) почти все, за исключением самых лучших правил, которые получены *H. Juillie и J. B. Pollack* [28] (их булевы формулы содержат 15–20 дизъюнктов), выразимы короткими булевыми формулами – менее пяти дизъюнктов в *DNF*. На основе экспериментов можно предположить, что не существует правил, классифицирующих более 83% начальных конфигураций, обладающих короткой булевой формулой;

6. Рассмотрены различные методы генетического программирования для решения задачи классификации плотности (*GA*, *GP*, *GEP*). Составлены компьютерные программы, их реализующие. Проверены результаты решения исследуемой задачи данными алгоритмами, анонсированные в публикациях. При реализации *GEP* была применена идея коэволюций, что не было сделано в предыдущих работах.
7. Проанализированы основные достоинства и недостатки каждого из существующих подходов к *DCT*, определена их сравнительная эффективность, предложен *новый* метод генетического программирования для ее решения, названный ***композитное генетическое программирование***. Основные идеи предлагаемого подхода сформулированы в общем виде, что позволяет применять его не только к *DCT*, но и к любым другим задачам, решаемым методами генетического программирования.
8. Метод композитного генетического программирования реализован на языке *Java* с использованием технологии *RMI*. Эксперименты показали, что он достаточно эффективен – были получены хорошие решения *DCT* значительно быстрее, чем предлагавшимися ранее подходами.
9. В результате работы метода было получено около 150 правил распознающих более 80% начальных конфигураций. Во всех публикациях, посвященных данной задаче, приводятся менее 10 правил, обладающих такой результативностью. Кроме того, полученное множество правил обладает большой разнородностью – в среднем два правила из набора отличаются в каждом пятом бите (из 128). На основе полученных данных, был проведен анализ на экстремумы пространства решений *DCT*. Найденное множество

“хороших” правил было разбито на восемь подмножеств, в каждом из которых любые два правила отличаются менее чем в 32 битах (каждый четвертый бит из 128).

Открытые вопросы и направления дальнейшей работы

1. Получить клеточный автомат, решающий *DCT* лучше, чем все известные на текущий момент.
2. Реализовать предложенный в данной работе метод композитного генетического программирования для решения задач:
   - итерированная дилемма узников;
   - задача про “умного” муравья;
   - задача синхронизации для клеточных автоматов.
3. Выявить нескольких достаточно трудных (например, принадлежащих классу сложности *NPC*) задач, элементами множества решений которых являются автоматы, а методы генетического программирования ранее не использовались для их решения. Провести теоретический анализ применимости предложенного в данной работе подхода к решению таких задач. Реализовать метод для конкретной задачи.
4. Установить сложность (от L) следующей задачи. **Дан** одномерный бинарный клеточный автомат P радиуса $r \in \mathrm{N}$, замкнутый в окружность: $P : 2^{2r+1} \to \{0, 1\}$. Автомат обладает свойством локальности – то есть r “много меньше” L (положим $\frac{N}{r} > 10$). Дана начальная конфигурация x, $x \in 2^L$. **Необходимо определить** распознает ли P конфигурацию x. Напомним, что P распознает x тогда и только тогда, когда $\exists n : \forall m \geq n \ (m, n \in \mathrm{N}) \Rightarrow P^{(m)}x = s$, где $s \in 2^L, s = \begin{matrix} 0^L, \ \rho(x) < \frac{1}{2} \\ 1^L, \ \rho(x) \geq \frac{1}{2} \end{matrix}$, а $\rho(x)$ есть “плотность конфигурации” x – отношение количества единиц в ней к L.
5. Установить сложность (от L и r) следующей задачи. Дан одномерный бинарный клеточный автомат P радиуса r и длины L, замкнутый в

окружность. Определить количество распознаваемых автоматом P начальных конфигураций x.

6. Установить сложность (от L и r) задачи вычисления $\max_{f \in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f)$.
7. Исследовать возможность создания общего подхода (в том числе и с использованием генетического программирования) к проектированию эффективных, надежных и помехоустойчивых систем, на основе клеточных автоматов, осуществляющих распределенные вычисления

ЛИТЕРАТУРА

1. *Mitchell M.* An Introduction to Genetic Algorithms. The MIT Press, MA, 1996.
2. *Тарантул В. З.* Геном человека. М.: Языки славянской культуры, 2003.
3. *Koza J. R.* Genetic programming. On the Programming of Computers by Means of Natural Selection. The MIT Press, MA, 1998.
4. *Ferreira C.* Gene Expression Programming: A New Adaptive Algorithm for Solving Problems /Complex Systems. 2001. Vol. 13, issue 2, pp. 87–129. www.gene-expression-programming.com/webpapers/GEP.pdf
5. *Vose M. D., Wright A.H.* Simple genetic algorithms with linear fitness //Evolutionary Computation. 1994. Vol. 2, number 4. http://citeseer.ist.psu.edu/vose94simple.html
6. *Vose M. D.* A Critical Examination Of The Schema Theorem. Technical Report UT-CS-93212. University of Tennessee Computer Science Department. Knoxville. TN. USA, 1993. http://citeseer.ist.psu.edu/129900.html
7. *Vose M. D., Wright A. H.* The Simple Genetic Algorithm and the Walsh Transform. Part I. Theory //Evolutionary Computation. 1998. Vol. 6, number 3. http://citeseer.ist.psu.edu/vose98simple.html
8. *Fogel D. B.* Applying Fogel and Burgin's 'Competitive Goal-Seeking through Evolutionary Programming' to Coordination, Trust, and Bargaining Games. IEEE Press Piscataway. NJ, 2000. http://citeseer.ist.psu.edu/fogel00applying.html
9. *Langdon W.B., Poli R.* Better Trained Ants for Genetic Programming. 1998. http://citeseer.ist.psu.edu/105696.html
10. *Das R., Crutchfield J. P., Mitchell M., Hanson J. E.* Evolving Globally Synchronized Cellular Automata /In Proceedings of the Sixth International Conference on Genetic Algorithms. 1995. pp. 336–343. http://citeseer.ist.psu.edu/das95evolving.html
11. *Sipper M.* Evolution of Parallel Cellular Machines //Lecture Notes in Computer Science. 2001. Vol. 1194. www.cs.bgu.ac.il/~sipper/papabs/epcm.pdf

12. *Гач П., Курдюмов Г.Л., Левин Л.А.* Одномерные однородные среды, размывающие конечные острова // Проблемы передачи информации. 1976. 14, 3.
13. *Тоом А.Л.* Монотонные бинарные мозаичные автоматы // Проблемы передачи информации. 1976. 12, 1.
14. *Тоом А.Л., Митюшин Л.Г.* Два результата о невычислимости для одномерных мозаичных автоматов // Проблемы передачи информации. 1976. 12, 2.
15. *Тоом А.Л.* Неэргодичность в однородных случайных средах //Вероятностные методы исследований. МГУ. 1972. 41. с. 34–42. http://www.de.ufpe.br/~toom/articles/rusmat/NONERG-R.PDF
16. *Курдюмов Г.Л.* Консервативность бесконечного строя // Квант. 1979. № 7. с.33–36. http://kvant.mccme.ru/au/kurdyumov_g.htm
17. *Колмогоров А.Н., Курдюмов Г.Л.*, Теоритико-алгоритмический метод исследования однородных случайных сред //Доклады АН СССР. 1978. Т. 238, № 5.
18. *Наумов Л.А, Шалыто А.А.* Клеточные автоматы. Реализация и эксперименты //Мир ПК. 2003, № 8, с. 64–71. http://is.ifmo.ru/works/klet/
19. *Наумов Л.А., Шалыто А.А.* "Цветные" клеточные автоматы, или клонирование Мона Лизы //Мир ПК. 2004, № 5, с. 64–71. http://is.ifmo.ru/works/cellaut/
20. *Wolfram S.* A New Kind of Science. Wolfram Media, 2002.
21. *Гуревич Л., Вахитов А., Макарова А.* Мультиагентные системы //Семинар “Введение в Computer Science” под руководством Ю. Матиясевича. 2005. http://teormin.ifmo.ru/education/intro/multiagent-systems.html
22. *Levin L. A.* Self-stabilization of Circular Arrays of Automata //Theoretical Computer Science. 2000. 235(1):143,144. http://arxiv.org/abs/cs.DC/0602033
23. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations. Physica D, 75: 361–391, 1993. http://web.cecs.pdx.edu/~mm/mech-imped.pdf

24. *Das R., Mitchell M., Crutchfield J.P.* A genetic algorithm discovers particle-based computation in cellular automata // Lecture Notes in Computer Science. 1994. www.santafe.edu/research/publications/workingpapers/94-03-015.pdf
25. *Land M., Belew R. K.* No Two-State CA for Density Classification Exists // Physical Review Letters. 1995. Vol. 74, Issue 25, pp. 5148–5150 http://citeseer.ist.psu.edu/67844.html
26. *Fuks H.* Solution of the Density Classification Problem with Two Cellular Automata Rules. 1997. //Physical Review E. 1997. Vol. 55, Issue 3, pp.R2081–R2084. http://www.citebase.org/fulltext?format=application%2Fpdf&identifier=oai%3AarXiv.org%3Acomp-gas%2F9703001
27. *Paredis, Jan*. Coevolving cellular automata: Be aware of the red queen! / Proceedings of the Seventh International Conference on Genetic Algorithms. 1997, pp. 393–400. http://citeseer.ist.psu.edu/paredis97coevolving.html
28. *Juillie H., Pollack J. B*. Coevolving the "Ideal" Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
29. *Werfel J., Mitchell M., Crutchfield J. P*. Resource Sharing and Coevolution in Evolving Cellular Automata. 1998. http://citeseer.ist.psu.edu/werfel99resource.html
30. *Ferreira C*. Discovery of the Boolean Functions to the Best Density Classification Rules Using Gene Expression Programming // Lecture Notes in Computer Science. 2002. Vol. 2278, pp. 51–60. www.gene-expression-programming.com/webpapers/ferreira-EuroGP02.pdf
31. *Andre D., Bennet F.H., Koza J.R.* Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem. 1996. http://citeseer.ist.psu.edu/andre96discovery.html

32. *Whitley D., Rana S., Heckendorn R. B.* The Island Model Genetic Algorithm: On separability, Populztion Size and Convergence. 1998. http://citeseer.ist.psu.edu/whitley98island.html
33. *Gustafson S. M.* An Analysis of Diversity in Genetic Programming. Ph.D. Dissertation. School of Computer Science and Information Technology. University of Nottingham. Nottingham. U.K., 2004. http://citeseer.ist.psu.edu/gustafson04analysis.html