МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА СИСТЕМЫ ОПРЕДЕЛЕНИЯ ИМПЛИЦИТНОГО НАУЧЕНИЯ НА ОСНОВЕ КОНЕЧНЫХ АВТОМАТОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Андросова Ивана Алексеевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Исследование имплицитного научения (научения, происходящего без осознания того, что именно является его предметом) начались в 1967 году с эксперимента когнитивного психолога Артура Ребера, в котором испытуемым предлагалось изучить некоторые последовательности букв, составленные по определенным правилам, и на основе этих последовательностей определять, соответствуют ли правилам другие последовательности (при этом испытуемым не говорилось о наличии формальных правил построения последовательностей, чтобы исключить намеренное выявление закономерностей и их обобщение). С этого эксперимента началось развитие одной из наиболее активно применяемых парадигм исследования имлицитного научения — научения искусственной грамматике [1, 2].

Исследование научения искусственной грамматике, как правило, происходит в два этапа. Сначала испытуемые изучают последовательности букв, которые подчиняются некоторым правилам. Затем испытуемому предъявляется новая последовательность букв, а он определяет, соответствует ли она этим правилам. Исследования в этой области показали, что испытуемые могут статистически достоверно отличать «грамматически верные» последовательности букв от «грамматически не верных». Однако успешно обучавшиеся люди не могли ответить на вопрос, как они справляются с этим заданием [2, 3].

Как правило, тесты в исследованиях научения искусственной грамматике включают в себя только задания с бинарным ответом (либо «последовательность является верной», либо «последовательность не является верной»). Однако для проверки умения усвоения правил можно использовать другие задания, как, например, самостоятельное составление грамматически верной последовательности. После того, как испытуемый составил последовательность, можно определить, является ли она верной — но для получения более полного представления об имплицитном научении стоит также учитывать, насколько близка составленная последовательность к правильной (если она не является корректной).

Кроме того, восприятие человеком последовательностей может зависеть от того, в каком виде задаются эти последовательности — в символьном, в звуковом, в цветовом или в каком-то другом. Необходимо изучить, является ли способ представления информации фактором, влияющим на способность

человека к имплицитному научению искусственной грамматике.

С учетом вышесказанного было принято решение разработать игровое приложение для исследования имплицитного научения. Игра должна состоять из двух этапов: на первом этапе должна присутствовать возможность пройти игровые уровни логическим путем, однако цвета объектов, с которыми взаимодействует игрок, должны образовывать правильные цветовые последовательности (этот этап соответствует первому этапу имплицитного научения); на втором этапе игрок должен определять объекты, с которыми он должен взаимодействовать, по тому, является ли раскраска этих объектов правильной цветовой последовательностью, или же самостоятельно составлять правильные цветовые последовательности. Кроме того, при составлении последовательности игроком приложение должно проверять составленную последовательность на допустимость для заданной грамматики в нечетком смысле, то есть необходимо уметь оценивать, «насколько хорошо» входная последовательность допускается недетерминированным конечным автоматом.

Цель данной работы — разработать и реализовать алгоритмы генерации и обработки последовательностей. Для решения поставленной цели требуется выполнить следующие задачи:

- разработать методы, позволяющие автоматически генерировать тесты для имплицитного научения грамматике — допустимые и недопустимые входные последовательности для заданного недетерминированного конечного автомата;
- построить математическую модель для определения нечеткой допустимости последовательности и разработать методы ее подсчета;
- обеспечить масштабируемость вышеописанных методов, то есть сделать их работающими корректно и эффективно для целого класса недетерминированных конечных автоматов.

Структура и объем работы. Бакалаврская работа состоит из раздела «Определения, обозначения и сокращения», введения, двух разделов основной части, заключения, списка использованных источников и одного приложения. Общий объем работы — 51 страница, из них 42 страницы — основное содержание, включая 4 рисунка, в списке использованных источников содержится 20 наименований.

1 Выбор математической модели и программной структуры

Первый раздел «Выбор математической модели и программной структуры» посвящен описанию и математическому определению функций, которые должны присутствовать в работе, и базовой реализации структуры недетерминированного конечного автомата на языке Python.

Для того, чтобы проанализировать способность человека к имплицитному научению на цветовых последовательностях, необходимо каким-то образом задать правила построения корректных и некорректных последовательностей. Помимо этого, также требуется определить способ простой проверки последовательности на корректность, способ нечеткой оценки корректности последовательности (т. е. надо уметь отвечать не только на запрос «является ли последовательность корректной», но и на запрос «насколько эта последовательность корректна», «сколько изменений в нее надо внести, чтобы она стала корректной») и способ выбора случайной корректной или некорректной последовательности. При этом, если требуется, чтобы игра «подстраивалась» под игрока, выдавая ему чаще последовательности именно с такими цветами, с которыми у игрока возникают сложности, необходимо уметь добавлять в реализованный способ генерации последовательности различные весовые коэффициенты для различных цветов.

В данной работе для определения корректности цветовой последовательности используется недетерминированный конечный автомат, которому на вход подаются цвета.

Определим, каким образом оценивается, «насколько корректна» заданная последовательность относительно автомата. Пусть дана последовательность a, и необходимо каким-то образом «заставить» автомат принять эту последовательность a. Можно совершать четыре различных вида операций:

- go — подать автомату на вход первый элемент a и удалить его из a;
- $insert(c)$ — подать автомату на вход цвет c, никак не меняя t;
- $ignore$ — удалить первый элемент a, не подавая его на вход автомату;
- $change(c)$ — заменить первый элемент a на цвет c и сразу подать на вход автомату;

Пусть $S(a)$ — множество всех таких последовательностей операций, что после применения всех операций из последовательности автомат находится в терминальном состоянии, а последовательность a пуста. Мы можем поставить

каждой операции стоимость (которая может зависеть не только от типа операции, но и от текущего состояния автомата или от конкретных цветов, используемых в операции), и назвать стоимостью $cost$ последовательности операций суммарную стоимость всех применяемых операций. Тогда можно определить взвешенную некорректность последовательности a следующим образом:

$$WeightedIncorrectness(a) = \min_{m \in S(a)} cost(m).$$

Как будет показано далее, для вычисления взвешенной некорректности также существует эффективный алгоритм, основанный на динамическом программировании или поиске кратчайшего пути во взвешенном ориентированном графе.

Выбор произвольной корректной или некорректной последовательности хотелось бы проводить равновероятно среди всех корректных или некорректных последовательностей фиксированной длины. Если для «подстраивания» игры под игрока необходимо добавить какие-то весовые коэффициенты каждому цвету, то можно определить вес последовательности как сумму весов элементов последовательности, и выбирать одну последовательность уже не равновероятно среди всех подходящих, а со взвешенной вероятностью. Одна из проблем, возникающих при этом, заключается в том, что пространство цветовых последовательностей фиксированной длины может быть неоднородным: в нем могут редко встречаться корректные последовательности, или они могут быть неравномерно распределены по пространству.

Чтобы реализовать недетерминированный конечный автомат, создан класс, включающий в качестве полей начальное состояние автомата, множество конечных состояний автомата и функцию переходов. Кроме того, для обработки последовательностей требуется хранить множество всех состояний, в которых автомат может находиться в текущий момент. Это множество также будет полем класса автомата.

2 Алгоритмы обработки и построения тестов

Второй раздел работы «Алгоритмы обработки и построения тестов» содержит математическое описание и доказательство алгоритмов генерации тестов на имплицитное научение и подсчета нечеткой некорректности, их реализацию на языке Python, а также тестирование методов генерации.

2.1 Проверка корректности последовательности

Для проверки корректности последовательности a достаточно проверить, допускает ли автомат эту последовательность. Для этого воспользуемся следующим алгоритмом: для каждого префикса последовательности a (обозначим префикс a длины k как $a[1..k]$) найдем множество $D(a[1..k])$ тех состояний, в которых может оказаться автомат, если изначально он был в стартовом состоянии, а затем ему на вход последовательно подавались все элементы $a[1..k]$ [4].

2.2 Оценка взвешенной некорректности последовательности

Для того, чтобы эффективно вычислить $WeightedIncorrectness(a)$, попробуем понять, какими свойствами обладают четыре типа операций, которые могут быть в последовательностях операций. До и после каждой операции автомат может находиться в каком-то из своих состояний (или сразу в нескольких состояниях, но этим можно пренебречь, так как мы можем при изменении состояния автомата выбирать новое состояние оптимальным образом), а последовательность a представляет собой суффикс изначальной последовательности. Поэтому различные состояния, в которых может находиться пара (автомат, последовательность a) после каких-то операций, описываются парой значений (u, l), где u — текущее состояние автомата, а l — количество элементов, уже удаленных из последовательности a. Между этими состояниями существуют четыре типа переходов:

- переходы по операции go переводят состояние (u, l) в состояние (u', $l+1$), если $u' \in \delta(u, a_{l+1})$;
- переходы по операции $insert(c)$ переводят состояние (u, l) в состояние (u', l), если $u' \in \delta(u, c)$;
- переходы по операции $ignore$ переводят состояние u, l в состояние u, $l+1$;

— переходы по операции $change(c)$ переводят состояние $(u,\ l)$ в состояние $(u',\ l+1)$, если $u' \in \delta(u, c)$.

Начальным состоянием является $(s,\ 0)$, конечными — все состояния вида $(t,\ |a|)$, где $t \in T$.

Стоимость достижения состояния можно рассматривать как длину кратчайшего пути в графе состояний из вершины $(s, 0)$ в вершину (u, l). Это позволяет моделировать задачу вычисления взвешенной некорректности последовательности как задачу поиска кратчайшего пути во взвешенном графе и решать ее одним из стандартных алгоритмов, например, поиском в ширину с обновлениями или алгоритмом Дейкстры [5]. Более того, если автомат является ацикличным, то граф состояний также является ацикличным (переходы go, $insert$ и $change$ переводят автомат в состояние, из которого нельзя вернуться в исходное, а переход $ignore$ уменьшает длину оставшегося суффикса последовательности, которую невозможно увеличить другими переходами), и задачу поиска кратчайшего пути можно решать за линейное (относительно размеров графа) время при помощи динамического программирования [6].

2.3 Генерация произвольной последовательности

Для генерации используется метод спуска по динамическому программированию, который полностью описан и доказан в бакалаврской работе. Задача генерации случайной корректной последовательности длины n состоит в том, что среди всех корректных последовательностей длины n мы должны равновероятно выбрать одну. Так как пространство поиска может быть разреженным и неоднородным, нельзя просто генерировать последовательность длины n и проверять ее на корректность до тех пор, пока мы не получим корректную последовательность.

Если недетерминированный конечный автомат рассматривать как граф, где каждая дуга соответствует переходу из одного состояния в другое, и на дуге записан соответствующий входной символ, то каждому пути (не обязательно простому) из стартового состояния в терминальное состояние соответствует корректная последовательность. Для подсчета количества путей из произвольного состояния v в терминальное состояние длины k можно определить рекуррентное соотношение:

— $F(v, 0) = [v \in T]$ (база рекурсии);

— $F(v,k) = \sum\limits_{u \in g(v)} F(u, k-1)$ (шаг рекурсии).

Данное рекуррентное соотношение можно оптимально считать динамическим программированием (так как в нем нет циклов). В полученной задаче динамического программирования всего $|Q|(n+1)$ состояний и nM переходов, поэтому таким образом мы можем подсчитать количество путей за время $O(n(M + |Q|))$ [7, 8].

После того, как мы посчитали все значения рекуррентного соотношения, можно равновероятно выбирать один из путей в автомате одним из следующих способов:

— Начать с пустой последовательности и состояния (s, n). На каждом шаге просматривать все такие состояния (v, k), что в них существует переход из текущего (обозначим их множество как H), и выбирать из них конкретное состояние (v_0, k_0) с вероятностью $\frac{F(v_0,k_0)}{\sum\limits_{(u,x)\in H} F(u,x)}$;

— сгенерировать случайное число x от 1 до $F(s, n)$, и спуском по динамическому программированию найти путь под номером x. Это делается следующим образом: на каждом шаге будем рассматривать все состояния, в которые можно перейти, по очереди; если в рассматриваемом состоянии (u, k) $F(u, k) \geqslant x$, то перейти в него, иначе заменить x на $x - F(u, k)$ и пропустить это состояние.

Однако у такого метода выбора входной последовательности есть один существенный недостаток. Рекуррентное соотношение 2.3 считает не количество последовательностей, а количество путей из стартового состояния в терминальное, и каждый из вышеописанных алгоритмов выбора будет выбирать не последовательность, а путь. Каждому пути соответствует ровно одна входная последовательность, но обратное в общем случае неверно:

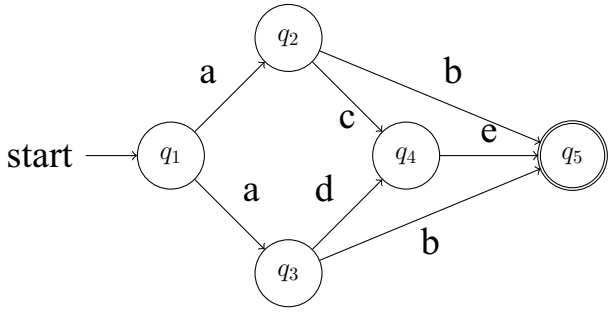

Рисунок 1 – Автомат, для которого одна и та же последовательность соответствует нескольким путям

Для автомата 1 существует входная последовательность ab, которой соответствуют два пути. Необходимо проанализировать подобные последовательности и понять, как их исключить или обработать.

Определение 1. *Назовем входную последовательность* $a_1 a_2 \dots a_n$ ***неоднозначно допускаемой****, если существуют две последовательности состояний* $q_0 q_1 \dots q_n$ *и* $q'_0 q'_1 \dots q'_n$*, таких, что:*

1. $q_0 = q'_0 = s$ *(первое состояние в обеих последовательностях — стартовое состояние автомата);*
2. $q_n \in T$, $q'_n \in T$ *(последние состояния в обеих последовательностях являются терминальными состояниями автомата);*
3. $\forall i \in [1, n]\ q_i \in \delta(q_{i-1}, a_1)$ *(для каждой пары соседних состояний в первой последовательности существует переход из предыдущего состояния в следующее по соответствующему входному символу);*
4. $\forall i \in [1, n]\ q'_i \in \delta(q'_{i-1}, a_1)$ *(для каждой пары соседних состояний во второй последовательности существует переход из предыдущего состояния в следующее по соответствующему входному символу);*
5. $\exists i\ q_i \neq q'_i$ *(последовательности различны).*

Очевидно, если входная последовательность является неоднозначно допускаемой, то в автомате существуют хотя бы два пути, соответствующие ей, и рекуррентное соотношение 2.3 учтет эту последовательность дважды (или более двух раз). Если же входная последовательность не является неоднозначно допускаемой, то либо она корректная и ей соответствует ровно один путь (и она будет учтена в рекуррентном соотношении ровно один раз), либо она некорректная (и не будет учтена вообще).

Как можно учесть неоднозначно допускаемых последовательности или избавиться от них? Одно из очевидных решений этой проблемы — построить по заданному автомату эквивалентный детерминированный конечный автомат. Для заданного автомата в его детерминированном эквиваленте будет 8 состояний и 40 переходов. В детерминированном автомате не существует нескольких путей, соответствующих одной и той же входной последовательности (так как по каждому символу из каждого состояния ровно один переход), поэтому неоднозначно допускаемых последовательностей не существует [9].

Второе решение основано на том факте, что для некоторых автоматов не существует неоднозначно допускаемых последовательностей, и для них можно

использовать алгоритм генерации без детерминизации. Для проверки автомата на наличие неоднозначно допускаемых последовательностей был разработан и реализован алгоритм со сложностью $O(|Q|^4)$ на основе серии поисков в ширину.

Алгоритм генерации некорректной последовательности во многом совпадает с алгоритмом генерации корректной последовательности, однако для вывода рекуррентного соотношения, по которому можно делать спуск, требуются дополнительные математические вычисления. Так же, как и в случае генерации корректной последовательности, были предложены две версии алгоритма: с детерминизацией автомата и без детерминизации (вторая версия работает только в случае отсутствия неоднозначно допускаемых последовательностей).

Вышеописанные алгоритмы реализованы на языке Python. Также описаны модификации этих алгоритмов на случай, когда требуется выбирать последовательности не равновероятно, а взвешенно.

Для реализованных алгоритмов было проведено тестирование, которое показало, что они действительно корректно и равновероятно генерируют необходимые последовательности.

ЗАКЛЮЧЕНИЕ

В результате работы были разработаны и реализованы на языке Python алгоритмы анализа недетерминированных конечных автоматов и последовательностей, которые можно применить для автоматического составления и проведения тестов на имплицитное научение. Для особого класса недетерминированных конечных автоматов (к этому классу принадлежат автоматы, для которых не существует неоднозначно допускаемых входных последовательностей) все алгоритмы являются полиномиальными относительно размера автомата, размера алфавита и длины последовательности; для остальных автоматов алгоритмы генерации последовательностей являются полиномиальными относительно размера эквивалентного детерминированного автомата. Кроме того, представлен алгоритм со сложностью $O(|Q|^4)$, позволяющий определить, существуют ли для данного автомата неоднозначно допускаемые последовательности. Методы построения корректных и некорректных последовательностей для имплицитного научения формальным грамматикам были протестированы, и по результатам тестирования они действительно выбирают последовательности равновероятно из всех подходящих.

Дальнейшие исследования данной задачи могут быть связаны с тем, что для автоматов, для которых существуют неоднозначно допускаемые последовательности, процесс детерминизации может привести к экспоненциальному разрастанию автомата. Возможно, существует способ «частично детерминировать» автомат, то есть построить эквивалентный автомат, для которого не будет существовать неоднозначно допускаемых последовательностей, но который будет по размерам значительно меньше эквивалентного детерминированного автомата.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Miller, G. A.* Project Grammarama. Psychology of communication / G. A. Miller. — New York: NY: Basic Books.

2 *Reber, A. S.* Implicit learning and tacit knowledge: An essay on the cognitive unconscious / A. S. Reber. — New York: Oxford University Press, 1993.

3 *Cohen-Cole, Jamie*. The reflexivity of cognitive science: the scientist as model of human nature / Jamie Cohen-Cole // *History of the Human Sciences*. — 2005. — Vol. 18, no. 4. — Pp. 107–139.

4 *Hopcroft, John E.* Introduction to automata theory, languages, and computation, 2nd edition / John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman // *ACM SIGACT News*. — 2001. — Vol. 32, no. 1. — P. 60.

5 *Dijkstra, E. W.* A note on two problems in connexion with graphs / E. W. Dijkstra // *Numerische Mathematik*. — 1959. — Vol. 1, no. 1. — Pp. 269–271.

6 *Hu, T. C.* Dynamic programming and graph optimization problems / T. C. Hu, J. D. Morgenthaler // *Computers & Mathematics with Applications*. — 1994. — Vol. 27, no. 9-10. — Pp. 53–58.

7 *Bellman, Richard Ernest*. Dynamic Programming / Richard Ernest Bellman. — Courier Corporation, 2003.

8 *Skiena, Steven S.* The Algorithm Design Manual / Steven S. Skiena. — Springer London, 2008.

9 NDFA to DFA Conversion [Электронный ресурс]. — URL: https://www.tutorialspoint.com/automata_theory/ndfa_to_dfa_conversion.htm (Дата обращения 26.05.2019). Загл. с экр. Яз. англ.