Министерство образования и науки Российской федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

Институт математики и компьютерных наук

Кафедра алгебры и дискретной математики

Реализация алгоритма проверки автомата на синхронизируемость с линейным временем работы в среднем

«Допущен к защите»

Зав. кафедрой

М.В Волков

_____

«___»_____2016 г.

Квалификационная работа на
соискание степени магистра наук
по направлению 02.04.02
Математические основы
компьютерных наук
Магистерская программа
«Математика и компьютерные науки»
студента гр. МКМ-240102
Агеева Павла Сергеевича
Научный руководитель
Волков Михаил Владимирович,
профессор,
доктор физико-математических наук

Екатеринбург
2016

РЕФЕРАТ

Агеев П.С. РЕАЛИЗАЦИЯ АЛГОРИТМА ПРОВЕРКИ АВТОМАТА НА СИНХРОНИЗИРУЕМОСТЬ С ЛИНЕЙНЫМ ВРЕМЕНЕМ РАБОТЫ В СРЕДНЕМ, квалификационная работа на степень магистра наук: стр. 30, рис. 4, табл. 3.

Ключевые слова: ДЕТЕРМИНИРОВАННЫЙ КОНЕЧНЫЙ АВТОМАТ, СИНХРОНИЗИРУЕМОСТЬ, АЛГОРИТМ, ГРАФ, ВЫЧИСЛИТЕЛЬНАЯ СЛОЖНОСТЬ

Объект исследования – алгоритмы проверки автоматов на синхронизируемость. В работе описана реализация и экспериментальное исследование алгоритма, разработанного М.В. Берлинковым [3], основанного на доказанной им гипотезе П.Дж. Кэмерона [4] о синхронизируемости с высокой вероятностью случайного детерминированного конечного автомата над алфавитом мощности больше одного.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной работе будет использоваться понятие полного *детерминированного конечного автомата*, не зависящего от начального и конечных состояний. Таким образом, под автоматом будем далее подразумевать тройку $\mathcal{A} = (Q, \Sigma, \delta)$, где Q обозначает конечное *множество состояний*, Σ – конечный *входной алфавит*, а $\delta : Q \times \Sigma \to Q$ – всюду определённую *функцию переходов*, которую можно рассматривать как множество действий букв из Σ на множестве состояний Q. Далее n – количество состояний автомата. Пусть Σ^* - множество всех слов, составленных из элементов входного алфавита. Тогда можно рассматривать действие слова из Σ^* на множестве состояний как последовательность действий букв, из которых слово составлено. Результат действия буквы a (слова w) на состояние q будем обозначать $q.a$ (соответственно $q.w$). Аналогичным образом можно ввести действия букв и слов на произвольные подмножества D множества состояний Q: $D.a = \{q.a \mid q \in D\}$. Автомат $\mathcal{A}$ называется *синхронизируемым*, если существует такое слово $w \in \Sigma^*$, под действием которого все состояния автомата переходят в одно, т.е. $q.w = p.w$ для всех $p, q \in Q$. Слово w в таком случае называются *синхронизирующим*.

Понятие синхронизируемости появляется на заре зарождения теории автоматов. В явном виде его ввёл Černý в 1964г. [5]. Так же в своей статье он формулирует знаменитую гипотезу о длине кратчайшего синхронизирующего слова, которая уже на протяжении более пятидесяти лет является нерешённой математической задачей. За это время интерес исследователей к этой области математики и компьютерных наук нисколько не угас. Напротив, в последнее десятилетие проведена масса теоретических исследований, в том числе разрешена известная задача о раскраске дорог [17], а рост производительности вычислительной техники позволяет проводить всё больше вычислительных экспериментов, используя их в качестве методов проверки гипотез и поиска

автоматов, обладающих важными для этой области свойствами [10], [1]. Кроме того, был найден ряд интересных приложений теории синхронизируемых автоматов в самых разных областях от теории кодирования [2] до робототехники [11], которые подробно разбираются в обзорных статьях М.В. Волкова [18] и [9] (последняя в соавторстве с J. Kari).

Таким образом, понятие синхронизируемости порождает ряд теоретических и алгоритмических вопросов, одним из которых является непосредственно проверка заданного автомата на синхронизируемость. Классическим решением этой задачи является основанный на приведённом ниже критерии синхронизируемости алгоритм, имеющий сложность $O(|\Sigma|n^2)$.

Предложение 1(Černý [5]). Автомат $\mathcal{A}$ синхронизируемый тогда и только тогда, когда для любой пары состояний $p, q \in Q$ существует слово w такое, что $q.w = p.w$.

Данный критерий может быть проверен поиском в ширину в автомате упорядоченных пар состояний автомата $\mathcal{A}$. Пусть $\mathcal{A}^2 = (Q \times Q,\ \Sigma,\ \delta^2)$, где δ^2 действует таким образом, что $(p, q).a = (p.a, q.a)$ для всех $p, q \in Q, a \in \Sigma$. Тогда условие синхронизируемости $\mathcal{A}$ эквивалентно наличию пути по обратным рёбрам в автомате $\mathcal{A}^2$ от множества всех состояний вида $(q, q), q \in Q$ до всех остальных состояний. Количество состояний и переходов в $\mathcal{A}^2$ соответственно $\frac{n(n+1)}{2}$ и $|\Sigma|\frac{n(n+1)}{2}$, следовательно, поиск в таком автомате потребует $O(\Sigma n^2)$ времени и памяти. Более того, время работы такого алгоритма будет квадратичным вне зависимости от специфики входного автомата.

Далее будем обозначать описанный выше алгоритм $IsSynchronizableSLow$.

Стоит отдельно отметить, что за более чем полувековую историю теория автоматов породила множество труднорешаемых задач. В качестве примера можно привести задачу нахождения кратчайшего синхронизирующего слова.

Доказано, что для заданного автомата $\mathcal{A}$ и числа l проверка того, существует ли у автомата синхронизирующее слово длины не более l, является NP-полной задачей [6]. Однако, существует ряд результатов, авторы которых предлагают полиномиальные алгоритмы, дающие приближенное решение для этой задачи [8], [12], [15]. Такие алгоритмы, помимо прочего, позволяют экспериментально оценивать ожидаемую длину кратчайшего синхронизирующего слова для случайного автомата [15].

Задача проверки автомата на синхронизируемость, как было показано выше, может быть решена за квадратичное время. Тем не менее, как и для некоторых трудных задач, в этом случае для случайных автоматов существует алгоритм, часто показывающий более высокую производительность, нежели алгоритмы, дающие точное решение.

Основной целью данной работы является описание и реализация алгоритма, предложенного М.В. Берлинковым [3], время работы которого в среднем растёт линейно по отношению к количеству состояний автомата.

ОПИСАНИЕ АЛГОРИТМА

Поскольку полученный в [3] алгоритм имеет линейное время работы в среднем, необходимо ввести понятие *случайного автомата*. Пусть, как и ранее, $|Q| = n$, а $|\Sigma| = k > 1$. Обозначим через Σ_n множество всех отображений множества Q в себя. Тогда случайный автомат $\mathcal{A}$ с n состояниями и k-буквенным алфавитом можно определить как $\{Q, \{\sigma_1, \sigma_2, \dots, \sigma_k\}\}$, где $\sigma_i \in \Sigma_n$ и все отображения σ_i выбраны случайно и равномерно с вероятностью $\frac{1}{n^n}$.

Предложение 2 (Теорема 1 из [3]). Вероятность быть синхронизируемым для случайного двухбуквенного автомата с n состояниями составляет $1 - \Theta\left(\frac{1}{n}\right)$.

Доказательство предложения 2 основано на глубоком структурном анализе автоматов и происходит следующим образом. Последовательно выписываются некоторые условия, которые в случайном автомате выполняются с большой вероятностью, а затем доказывается, что из этих условий следует синхронизируемость автомата.

Основная идея алгоритма состоит в том, чтобы последовательно проверять ограничения, накладываемые на автомат доказательством предложения 2. Если они выполняются, то алгоритм возвращает «$true$». В обратном случае запускается алгоритм $IsSynchronizableSlow$, работающий за квадратичное время, но по предложению 2 это будет происходить с вероятностью $1 - O\left(\frac{1}{n}\right)$, так что в среднем сложность алгоритма будет линейной. Таким образом, несмотря на то, что время выполнения алгоритма является некоторой случайной величиной, он не является вероятностным, поскольку не обращается в ходе своей работы к генератору случайных чисел, а время, необходимое для получения ответа, полностью определяется входными данными.

Для дальнейшего изложения нам потребуется привести некоторые термины из теории графов.

Ориентированным графом G называется пара $G = (V, E)$, где V – множество вершин, а $E \subseteq V \times V$ – множество дуг.

Путём в графе называется последовательность $v_0 e_1 v_1 \dots e_k v_k$, где $e_i \in E$, $e_i = (v_{i-1}, v_i)$.

Пара вершин $\{v, u\}$ называется *сильно связной*, если существует путь из v в u и из u в v. Понятие сильной связности порождает отношение *сильной связности,* являющееся эквивалентностью. Классы эквивалентности этого отношения называются *компонентами сильной связности.* Сжав каждую компоненту сильной связности, можно получить ациклический *граф конденсации* исходного графа, вершинами которого являются компоненты, а дуга $([v], [u])$ существует, если в исходном графе по крайней мере из одной вершины компоненты $[v]$ есть дуга, ведущая в вершину компоненты $[u]$.

В целях описания алгоритма понадобится ряд терминов, характеризующих структуру переходов в автомате по конкретной букве. Пусть $UG(\mathcal{A})$ – *граф автомата*, т.е. ориентированный граф, множество вершин которого – Q, а мультимножество дуг - $\{(q, q.a) \mid q \in Q,\ a \in \Sigma\}$. Тогда *граф буквы* $x \in \Sigma$ (*отображения* $\sigma_x \in \Sigma_n$), т.е. граф автомата $\mathcal{A}$ с функцией переходов, ограниченной до действия одной буквы x, будем обозначать $UG(x)$. Для произвольной буквы такой граф состоит из одной или нескольких компонент (слабой) связности, которые далее будем называть *кластерами*. Каждый кластер содержит единственный цикл, во всех вершинах которого находятся корни деревьев (возможно состоящих из одной вершины). Назовём *1-ветвью* произвольное дерево в графе отображения $\sigma_x \in \Sigma_n$, корнем которого является вершина высоты 1 (вершины цикла имеют высоту 0). Пусть T – 1-ветвь максимальной высоты в $UG(x)$, а h - высота второй по высоте 1-ветви. Тогда лес, состоящий из вершин, высота которых составляет как минимум $h + 1$, будем называть *1-кроной* T.

Введём, кроме того, некоторые определения, связанные с понятием синхронизируемости. Будем говорить, что множество состояний $D \subseteq Q$ *синхронизируемо*, если оно переходит в единственное состояние под действием некоторого слова. Будем называть пару $\{p, q\}$ *тупиком*, если для любого слова $w \in \Sigma^*$ $p.w \neq q.w$. Напротив, пара $\{p, q\}$ называется *стабильной*, если для любого слова $w \in \Sigma^*$ существует такое слово $v \in \Sigma^*$, что $p.wv = q.wv$. Отметим, что *отношение стабильности* на множестве состояний автомата является транзитивным и инвариантно относительно действия функции переходов.

Ключевой идеей доказательства предложения 2 является утверждение о том, что если в автомате синхронизируемо достаточно большое множество состояний, то с большой вероятностью синхронизируем и весь автомат. Для построение такого множества состояний необходимо для каждой буквы найти достаточно большое число различных независимых от неё стабильных пар, для чего, в свою очередь, граф автомата должен удовлетворять ряду ограничений, каждое из которых при описанной ранее стратегии выбора случайного автомата выполняется с вероятностью $1 - O\left(\frac{1}{n}\right)$.

Перейдём теперь непосредственно к изложению алгоритма для $k = 2$, а уже после укажем, как обобщить его для произвольных $k > 1$.

Прежде всего, необходимо найти компоненты сильной связности графа автомата, построить его конденсацию и проверить, что в полученном ориентированном ациклическом графе существует вершина, достижимая из всех остальных (будем называть компоненту, соответствующую этой вершине *наименьшей*). В самом деле, если такой вершины нет, то нет и состояния автомата, которое может быть достижимым по переходам автомата из всех остальных состояний. В таком случае автомат несинхронизируем, и алгоритм возвращает «$false$».

Лемма 1 (Лемма 6 из [3]). Для произвольной константы $q > 1$ количество состояний в каждом подавтомате $\mathcal{A}$ как минимум n/eq^2 с вероятностью $1 - O\left(\frac{1}{n}\right)$.

Далее необходимо проверить, что в наименьшей компоненте сильной связности содержится не менее $n/4e^2$ состояний. Это потребуется далее для поиска стабильной пары состояний, не зависящей от одной из букв. По лемме 1 такое условие выполняется с вероятностью $1 - O\left(\frac{1}{n}\right)$, в обратном случае запускается алгоритм $IsSynchronizableSlow$.

Теперь необходимо найти стабильную пару состояний, не зависящую от одной из букв. Для этого по каждому из графов $UG(x), x \in \Sigma$ построим структуру данных $ClusterStructure$, характеризующую состояния автомата с точки зрения устройства $UG(x)$ и хранящую информацию о кластерах этого графа. $ClusterStructure$ должна предоставлять доступ к следующим значениям: $ClusterCount$ – количество кластеров, $Highest1BranchHeight$ и $Highest1BranchRoot$ – соответственно высота и корень уникальной 1-ветви максимальной высоты (если такая существует), $Highest1CrownSize$ и $Highest1CrownRootsCount$ – соответственно размер и количество корней в 1-кроне уникальной 1-ветви максимальной высоты (если такая существует). Кроме того, для каждого состояния p указанная структура должна хранить $Height(p)$ – высоту вершины в дереве, $ClusterIndex(p)$ – номер кластера, содержащего вершину; а для каждого кластера c должна содержать $ClusterSize(c)$ – количество состояний в нём, $CycleLength(c)$ - длина цикла и $CycleStates(c)$ – состояния, принадлежащие единственному циклу кластера.

Лемма 2 (Лемма 2 из [3]). С вероятностью $1 - o\left(\frac{1}{n^4}\right)$ случайный граф отображения $\sigma \in \Sigma_n$ содержит не более $5 \ln n$ кластеров.

Следуя условию накладываемому леммой 2, необходимо убедиться, что для каждой буквы значение $ClusterCount$ не превышает $5\ln n$, и в обратном случае вызвать алгоритм $IsSynchronizableSlow$.

С вероятностью $1 - O\left(\frac{1}{n}\right)$ хотя бы для одной из букв x в графе $UG(x)$ будет существовать 1-ветвь наибольшей высоты (т.е. уникальная 1-ветвь максимальной высоты), для 1-кроны H которой выполняется $Highest1CrownSize > 2 * Highest1CrownRootsCount > 0$ (Замечание 1 из [3]). А это значит, что в условиях леммы 1 с вероятностью $1 - O\left(\frac{1}{n}\right)$ 1-крона H пересекается с наименьшей компонентой сильной связности (Теорема 6 из [3]). Пусть без ограничения общности перечисленные условия выполнились для буквы a. Тогда по Теореме 3 из [3] корень r 1-ветви наибольшей высоты и вершина q, предшествующая в цикле корню дерева (0-ветви), содержащего H, образуют стабильную пару $\{r, q\}$, не зависящую от буквы b.

Лемма 3 (Лемма 7 из [3]) Если в автомате $\mathcal{A}$ существует стабильная пара $\{p, q\}$, не зависящая от b, то для любой константы $k > 1$ с вероятностью $1 - O\left(\frac{1}{n}\right)$ существует k различных стабильных пар $\{p.b, q, b\}, \{p.b^2, q.b^2\}, \dots, \{p.b^k, q.b^k\}$, не зависящих от a.

Лемма 4 (Лемма 8 из [3]) Если в автомате существует 6 стабильных пар $\{p_i, q_i\}, i \in \{1,2,\dots,6\}$, не зависящих от a, то с вероятностью $1 - O\left(\frac{1}{n}\right)$ для одного из индексов $j \in \{1,2,\dots,6\}$ существует $n^{0.4}$ различных стабильных пар $\{p_j.a, q_j, a\}, \{p_j.a^2, q_j.a^2\}, \dots, \{p_j.a^{n^{0.4}}, q_j.a^{n^{0.4}}\}$, не зависящих от b.

Теперь, имея стабильную пару, не зависящую от буквы b, воспользовавшись леммами 3 и 4, с вероятностью $1 - O\left(\frac{1}{n}\right)$ можно построить множество Z_b в точности $\lceil n^{0.4} \rceil$ стабильных пар, не зависящих от b.

Воспользовавшись этими леммами симметрично, можно построить аналогичное множество Z_a стабильных пар, не зависящих от a.

Далее необходимо проверить на синхронизируемость достаточно большие множества вершин для каждой из букв. В качестве такого множества для буквы x возьмём множество $\widehat{S_x}$ всех состояний, лежащих в таких кластерах S_x из $UG(x)$, размер которых превышает $n^{0.45}$.

Пусть $\Gamma(S_x, Z_x)$ – *граф больших кластеров*, вершинами которого являются кластеры S_x, а между парой кластеров $c_1, c_2 \in S_x$ есть ребро, если существует стабильная пара $\{p, q\} \in Z_x$ такая, что $p \in c_1$, а $q \in c_2$. Пусть, кроме того, d – наибольший общий делитель длин всех циклов кластеров S_x. Для проверки синхронизируемости множества $\widehat{S_x}$ воспользуемся леммами 3 и 4 из [3]. Достаточно проверить, что граф $\Gamma(S_x, Z_x)$ связный и выполняется хотя бы одно из условий:

1. $d = 1$
2. Не существует такого набора $x_i \mid 0 \le x_i \le d-1, i \in \{1, 2, \dots, |S_x|\}$, что для любой пары $\{p, q\} \in Z_x$

$$d \mid \big(Height(p) - Height(q)\big) - (x_{ClusterIndex(p)} - x_{ClusterIndex(q)})$$

Т.к. $|Z_x| \ge n^{0.45}$, то по леммам 3 и 4 из [1] перечисленные свойства выполняются с вероятностью $1 - O\left(\frac{1}{n}\right)$. В обратном случае запустим $IsSynchronizableSlow$.

В финальной стадии алгоритма необходимо проверить, синхронизируем ли весь автомат в предположении, что множества $\widehat{S_a}$ и $\widehat{S_b}$ синхронизируемы. Для этого достаточно последовательно разобрать все случаи из доказательства Теоремы 2 из [3].

Пусть $\widehat{T_x}$ – множество состояний автомата, лежащих в кластерах, размер которых не превосходит $n^{0.45}$. Другими словами, $\widehat{T_x} = Q \backslash \widehat{S_x}$.

В доказательстве Теоремы 2 из [1] представлен ряд необходимых условий наличия тупика, выполняющихся с вероятностью $O\left(\frac{1}{n}\right)$ в случайном автомате при условии синхронизируемости множеств $\widehat{S_a}$ и $\widehat{S_b}$. (т.е. невыполнение этих условий является достаточным для синхронизируемости всего автомата $\mathcal{A}$).

Пусть c_p и c_q – кластеры, содержащие соответственно состояния p и q, а s_p и s_q – циклы этих кластеров. Для каждой буквы необходимо проверить (условия будут описаны для буквы a):

(1.1) В каждом цикле s длины большей двух графа $UG(a)$ по крайней половина всех состояний лежит в $\widehat{T_b}$. ($\left|s \cap \widehat{T_b}\right| \geq \lceil 0.5|s| \rceil$)

(1.2) Для каждого цикла длины два, состоящего из вершин $\{p, q\}$, хотя бы одна из которых лежит в $\widehat{T_b}$, построим множество $R = \{p.b, q.b, p.b^2, q.b^2\}$ и пусть $R_1 = \{p.b, q.b\}$, $R_2 = \{p.b^2, q.b^2\}$. Выполняется одно из следующих условий:

- $|R| = 2$
- $|R| = 3$ и хотя бы один элемент из R_1 лежит в $\widehat{T_b}$.
- $|R| = 4$, хотя бы один элемент из R_1 лежит в $\widehat{T_b}$ и хотя бы один элемент из R_2 лежит в $\widehat{T_b}$.

(1.3) Среди всех пар различных кластеров $\{c_p, c_q\}$, таких что хотя бы один из них лежит в T_a и хотя бы один из них имеет длину цикла 1, по крайней мере один из циклов s_p и s_q лежит в $\widehat{T_b}$ и по крайней мере одно из множеств $s_p.b$ и $s_q.b$ лежит в $\widehat{T_a}$.

(1.4) Среди всех пар различных кластеров $\{c_p, c_q\}$, таких что хотя бы один из них лежит в T_a и оба имеют длину цикла больше 1, существует такая, что для неё выполнится следующее. Пусть d – наибольший общий делитель s_p и s_q, $Z_d = \{0,1, \dots, d-1\}$. Пусть, кроме того, для всех $r \in \{p, q\}$ для состояний на цикле s_r задан порядок $s_{r,i}$, такой что $s_{r,i}.a = s_{r,i+1 \bmod |s_r|}$. Построим множества индексов $I_r \subseteq Z_d$, такие что для

всех $i \in I_r$ и для всех $k > 0$ верно $c_{r,i+kd} \in \widehat{T_b}$. Тогда существует такое смещение $x \in Z_d$, что $\{x + i \mid i \in I_p\} \cup I_q = Z_d$.

Если хотя бы одно из условий (1.1)-(1.4) верно, что произойдёт с вероятностью $O\left(\frac{1}{n}\right)$, то нужно вызвать $IsSynchronizableSlow$. В обратном случае алгоритм возвращает «$true$». Эти рассуждения завершают описание алгоритма для двухбуквенного автомата.

Для проверки синхронизируемости автомата $\{Q, \{\sigma_1, \sigma_2, \dots, \sigma_k\}\}, \sigma_i \in \Sigma_n$ с размером алфавита $k > 2$ достаточно запустить описанный алгоритм для двухбуквенного автомата $\{Q, \{\sigma_1, \sigma_2\}\}$, но в случае невыполнения одного из условий не запускать квадратичный алгоритм и не возвращать отрицательный результат, а переходить к следующему автомату $\{Q, \{\sigma_3, \sigma_4\}\}$. Если ни для какого из двухбуквенных автоматов не будет получен положительный ответ, что произойдёт с вероятностью $O\left(\frac{1}{n^{[k/2]}}\right)$, необходимо запустить алгоритм $IsSynchronizableSlow$.

ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМА

Прежде всего, опишем структуры данных для хранения автоматов и графов. Автомат $\mathcal{A}$ хранится в виде двумерного массива целых чисел $\mathcal{A}[p][c]$ в p-й строке и в c-м столбце которого хранится номер состояния $q = p.c$. Произвольный ориентированный граф G представлен в коде в виде массива списков целых чисел или *списков смежности*. В списке $G[v]$ содержатся концы всех дуг, выходящих из вершины v.

Шаг 1. Построение компонент сильной связности.

Компоненты сильной связности графа автомата $UG(\mathcal{A})$ можно найти воспользовавшись алгоритмом Тарьяна [16] или более простым, но имеющим большую скрытую константу, алгоритмом Шарира [14]. Оба алгоритма имеют линейную сложность от количества вершин и на выходе дают массив $component[p]$, хранящий номер компоненты сильной связности, в которой находится состояние p. Имея этот массив и исходный автомат $\mathcal{A}[p][c]$, можно построить граф конденсации.

$\boldsymbol{BuildCondensation}(\mathcal{A}, component)$

```
1       Создать массив пустых списков condensationGraph[0 ... n]
2       foreach(p ∈ Q)
3           foreach(c ∈ Σ)
4               to ← A[p][c]
5               cP ← component[p]
6               cQ ← component[q]
7               if cP == cQ
8                   Вставить cQ в condensationGraph[cP]
```

Приведённый псевдокод строит граф конденсации, некоторые рёбра которого, возможно, дублируются, и в нашем случае, при $\Sigma = 2$, работает за $O(n)$.

Имея граф конденсации, можно отвечать на вопрос о существовании наименьшей компоненты сильной связности. В самом деле, достаточно убедиться в единственности вершины степени ноль этого графа. Кроме того, воспользовавшись информацией массива $component[p]$, можно получить номер и размер такой компоненты для проверки условия Леммы 1.

Шаг 2. Построение для каждой из букв соответствующей $ClusterStructure$.

По Лемме 9 из [3] При построении $ClusterStructure$ достаточно для каждого необработанного состояния идти по единственному пути

$$p = p_0, p_1 = p_0.x, \dots, p_m = p_{m-1}.x$$

до тех пор, пока не найдётся состояние $p_m = p_k = p_k.x^{m-k}$ для некоторого $k < m$. Тогда состояния p_i для всех $k \le i < m$ образуют единственный цикл в кластере, содержащем p. Запуская поиск в глубину в графе рассматриваемой буквы с развёрнутыми рёбрами из всех состояний высоты 1 в деревьях, корни которых находятся в состояниях цикла, можно вычислить высоту и номер кластера для каждого состояния.

В дополнение к упомянутым ранее значениям, будем также хранить $secondHighest1BranchHeight$ – высота второй по высоте 1-ветви, сравнив которую с $highest1BranchHeight$, можно ответить на вопрос об уникальности 1-ветви с максимальной высотой.

BuildClusterStructure$(\mathcal{A}, \mathrm{c})$

1 $highest1BranchRoot \leftarrow NULL$

2 $highest1BranchHeight \leftarrow -1$

3 $highest1BranchCyclePredecessor \leftarrow NULL$

4 $secondHighest1BranchHeight \leftarrow -1$

5 Массив логических переменных $used[p]$ заполнить значениями $false$

6 $currentCluster \leftarrow 0$

7 $\boldsymbol{foreach}(p \in Q)$

8 $\quad \boldsymbol{if}$ p лежит в обработанном кластере

9 $\quad\quad$ Перейти на следующий шаг

10 $\quad q \leftarrow p$

11 $\quad$ Создать стек состояний $stack$

12 $\quad \boldsymbol{while}\ used[q] == false$

13 $\quad\quad stack.push(q)$

14 $\quad\quad q = q.a$

15 $\quad\quad used[q] = true$

16 $\quad CycleStates(currentCluster) \leftarrow$ состояния, лежащие в стеке между парой одинаковых состояний вместе с одним из них.

17 $\quad$ Для всех $q \in stack \backslash CycleStates(currentCluster)$ записать $used[x] = false$

18 $\quad CycleLength(currentCluster) \leftarrow |CycleStates(currentCluster)|$

19 $\quad \boldsymbol{foreach}(q \in CycleStates(currentCluster))$

20 $\quad\quad \boldsymbol{foreach}(r \in \{t \mid t \in Q, t.a = q\} \setminus CycleStates(currentCluster))$

21 $\quad\quad\quad$ Запустить из r поиск в глубину по обратным дугам, вычисляющий для всех состояний x 1-ветви с корнем r значение $Height(x)$ и записывающий в $ClusterIndex(x)$ значение $currentCluster$.

22 $\quad\quad\quad$ Для всех состояний x 1-ветви с корнем r записать $used[x] = true$

23 $\quad\quad\quad branchHeight = \max\limits_{x \text{ из } 1-\text{ветви с корнем } r} Height(x)$

24 $\quad\quad\quad \boldsymbol{if}(branchHeight \geq highest1BranchHeight)$

25 $\quad\quad\quad\quad secondHighest1BranchHeight \leftarrow highest1BranchHeight$

26 $\quad\quad\quad\quad highest1BranchHeight \leftarrow branchHeight$

27 $\quad\quad\quad\quad highest1BranchRoot \leftarrow r$

28 $\quad\quad\quad\quad highest1BranchCyclePredecessor \leftarrow$ предшественник q в цикле.

29 $\quad\quad\quad \boldsymbol{if}(branchHeight > secondHighest1BranchHeight)$

30 $\quad\quad\quad\quad secondHighest1BranchHeight = branchHeight$

31 $\quad currentCluster = currentCluster + 1$

Оценим сложность приведённого псевдокода. Инициализация переменных в строках 1-6 работает за линейное время. Необходимо убедиться, что в основном цикле в строках 19-30 каждое состояние просматривается не более, чем константное количество раз. Условие в строках 8-9 пропускает все вершины из уже просмотренных кластеров. Таким образом, достаточно убедиться, что оставшаяся часть основного цикла за линейное количество операций пометит все вершины текущего кластера, содержащего вершину p. Цикл в строках 12-15 закончит работу и имеет линейную сложность от количества состояний в текущем кластере в силу того, что, двигаясь по дугам графа буквы из произвольной вершины, всегда можно попасть в цикл. Далее, в цикле в строках 19-30 просматриваются все состояния r высоты один в деревьях с корнями в состояниях цикла. Из каждого такого состояния запускается алгоритм поиска в глубину с линейной сложностью от размера соответствующей 1-ветви T, а в строке 22 помечаются все вершины T. Поскольку проверки, выполняемые алгоритмом в строках 24-30, выполняются за константное время, а каждая вершина кластера будет посещена поиском в глубину не более одного раза, сложность алгоритма составит $O(n)$.

Шаг 3. Проверка синхронизируемости $\widehat{S_a}$ и $\widehat{S_b}$.

Пусть удалось найти $ClusterStructure$ с уникальной 1-ветвью максимальной высоты, размер кроны H которой более чем в два раза превышает количество корней в ней (эти величины можно посчитать, т.к. для всех состояний посчитана высота $Height(x)$). Тогда имея вычисленный массив $component[p]$ и граф конденсации легко проверить за линейное время, что H пересекается с наименьшей компонентой сильной связности.

Кроме того, в ходе работы процедуры $BuildClusterStructure$ были вычислены вершины, образующие независимую от одной из букв стабильную пару $\{highest1BranchCyclePredecessor, highest1BranchRoot\}$. Явно

воспользовавшись Леммами 3 и 4, можно получить множества стабильных пар Z_a и Z_b. Алгоритм построения будет иметь асимптотику $O(n^{0.4})$.

Теперь, имея необходимую информацию о кластерах, для каждой буквы можно найти наибольший общий делитель d длин всех циклов графа этой буквы. Поскольку число кластеров не превышает $5 \ln n$, воспользовавшись алгоритмом Евклида, это можно сделать за время $O(\ln^2 n)$. Если для всех букв $d = 1$, то $\widehat{S_a}$ и $\widehat{S_b}$ синхронизируемы, и можно переходить к следующему шагу алгоритма. В случае $d > 1$ можно для каждой буквы x построить граф больших кластеров $\Gamma(S_x, Z_x)$. Т.к. $|S_x| \leq 5 \ln n$, а $|Z_x| \leq n^{0.40}$, сложность построения составит $O(n^{0.40} \ln n)$. Осталось проверить связность этого графа и наличие такого набора $x_i \mid 0 \leq x_i \leq d-1, i \in \{1, 2, \dots, |S_x|\}$, что для любой пары $\{p, q\} \in Z_x$

$$d \mid \big(Height(p) - Height(q)\big) - (x_{ClusterIndex(p)} - x_{ClusterIndex(q)}) \quad (1)$$

Для этого нужно из произвольной вершины v графа больших кластеров запустить поиск в глубину, выбрав для неё $x_v = 0$. При переходе по ребру $\{p, q\}$ в ещё не просмотренную вершину q, значение $x_q = \big(Height(q) - Height(p) + x_p\big) \bmod d$ для неё однозначно вычисляется по формуле (1). После окончания работы поиска нужно проверить условие (1) для всех пар из Z_x. Если для какой-то из пар условие не выполняется и граф $\Gamma(S_x, Z_x)$ связный, то $\widehat{S_x}$ синхронизируемо. В обратном случае, выполняющемся с вероятностью $O\left(\frac{1}{n}\right)$, необходимо запустить $IsSynchronizableSlow$. Вычислительная сложность всех проверок линейна по отношению к размеру графа больших кластеров и равна, таким образом, $O(n^{0.4})$.

Шаг 4. Проверка достаточных условий синхронизируемости всего автомата при условии синхронизируемости $\widehat{S_a}$ и $\widehat{S_b}$.

Если выполняется хотя бы одно из условий (1.1)-(1.4), что произойдёт с вероятностью $O\left(\frac{1}{n}\right)$, необходимо запустить $IsSynchronizableSlow$. Осталось

убедиться, что проверка этих условий будет иметь линейную сложность. Имея построенные $ClusterStructure$ для каждой буквы, в частности информацию о размерах кластеров и о номере кластера для каждого состояния, можно легко за константное время отвечать, лежит ли произвольное состояние в $\widehat{S_x}$ или в $\widehat{T_x}$ для заданной буквы $x \in \Sigma$. Таким образом, условия (1.1) и (1.2) проверяются за линейное время.

Сложность проверки условия (1.3) составит $O(\mathrm{n} + \ln^2 n) = O(n)$, т.к. можно заранее проверить каждый из циклов s_p и образ его перехода $s_p.b$ по букве b на принадлежность соответственно множествам $\widehat{T_b}$ и $\widehat{T_a}$, а уже потом проверять непосредственно само условие (1.3).

Для проверки условия (1.4) нужно для каждой пары кластеров $\{c_p, c_q\}$, таких что по крайней мере один из них лежит в T_a, построить множества индексов I_p и I_q. Размер каждого из множеств не превышает числа $d < n^{0.45}$ (наибольший общий делитель длин циклов s_p и s_q), а сложность проверки числа $i < d$ на принадлежность такому множеству индексов составит $O(n^{0.45})$. Таким образом I_p и I_q можно построить за время $O(n^{0.9})$. Далее, по уже построенному множеству I_p будем последовательно генерировать множества $I_{p,x} = \{x + i \mid i \in I_p\}$, $x \in Z_d$ и проверять, совпадает ли объединение $I_{p,x}$ и I_q с множеством Z_d. Сложность этих действий, опять же, составит $O(n^{0.9})$. Таким образом, вспомнив, что количество пар кластеров не превышает $25 \ln^2 n$, можно заключить, что асимптотика всего алгоритма проверки условия (1.4) составит $O(\mathrm{n}^{0.9} \ln^2 n) \subseteq O(n)$.

В итоге, мы получили, что все шаги алгоритма имеют вычислительную сложность $O(n)$, а алгоритм $IsSynchronizableSlow$ вызывается с вероятностью с вероятностью $O\left(\frac{1}{n}\right)$. Таким образом, описанный алгоритм работает за $O(n)$ в среднем для двухбуквенного автомата.

В случае $k > 2$ алгоритм для двухбуквенного автомата будет вызван t раз с вероятностью $O\left(\frac{1}{n^t}\right)$, а алгоритм $IsSynchronizableSlow$ с вычислительной сложностью $O(n^2k)$ будет вызван с вероятностью $O\left(\frac{1}{n^{[k/2]}}\right)$, из чего сразу следует линейная оценка времени работы всего алгоритма для автоматов над алфавитами произвольного размера.

Некоторые оптимизации

При реализации алгоритмов, основанных на достаточно сложных и объёмных теоретических результатах, важно отдельное внимание уделять природе ограничений, накладываемых в доказательствах на используемые структуры. В данном случае необходимо понимать, что часть ограничений введена для возможности подсчёта вероятностей и упрощения доказательств. Например, проверка того, что количество состояний в уникальной 1-кроне A более чем в два раза превосходит количество деревьев в ней, и ограничение на размер наименьшей компоненты сильной связности B необходимы для последующего доказательства того, что множества состояний двух этих структур пересекаются с большой вероятностью. Вместо проверки этих сложных условий, являющихся по сути достаточными для наличия состояний в пересечении A и B с большой вероятностью, можно непосредственно проверить, что $A \cap B \neq \emptyset$.

Аналогичным образом необязательным является ограничение снизу количества построенных стабильных пар. В доказательстве предложения 2 такое условие является достаточным для связности с большой вероятностью графа больших кластеров. В реализации же снова можно проверить непосредственно связность рассматриваемого графа. Ещё одно ограничение на множества стабильных пар заключается в том, что по доказательству для подсчёта вероятностей требуется, чтобы все состояния, лежащие в построенных стабильных парах, были попарно различными. В реализации это условие можно также опустить.

Такого рода оптимизации не только могут значительно упростить алгоритм, но и заметно улучшить его производительность на практике. В самом деле, в описанном алгоритме все необязательные при реализации условия выполняются для случайного автомата с определённой вероятностью P и являются достаточными для выполнения последующих проверок. Ясно, что при замене их на необходимые и достаточные условия, вероятность выполнения новых проверок P' может только увеличиться. Таким образом, среднее количество запусков квадратичного алгоритма $IsSynchronizableSlow$ может быть понижено.

ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

Технические детали

Описанный в предыдущих разделах алгоритм реализован на языке C++11. Компиляция и сборка проводилась в среде Microsoft Visual Studio 2013 (версия компилятора MSVC 18.0.31101.0). Вычисления проводились на компьютере с процессором Intel Core i7-4770K (3.50GHz) и 16Gb RAM.

Исходный код программы можно найти по адресу https://github.com/birneAgeev/AutomataSynchronizationChecker.

Результаты

Для простоты изложения далее будем именовать изложенный в предыдущих разделах алгоритм *основным*.

Для того, чтобы убедиться в эффективности и практической применимости основного алгоритма, был проведен вычислительный эксперимент, позволяющий наглядно сравнить его с квадратичным алгоритмом $IsSynchronizableSlow$, основанным на предложении 1.

Для удобства вычисления времени работы основной алгоритм был модифицирован таким образом, чтобы в случае невыполнения одной из проверок вместо запуска $IsSynchronizableSlow$ возвращалось значение «$fail$». Используя такую версию основного алгоритма можно более точно измерить производительность всех проверок и экспериментально проверить линейную зависимость времени их выполнения по отношению к количеству состояний во входном автомате. В табл. 1 и на рис.1 приведено среднее время работы в секундах модификации основного алгоритма для автоматов разных размеров. При подсчёте среднего для всех значений n и k проверка запускалась 1000 раз.

k \ n	100	1000	5000	10000	100000
2	0.00013	0.00112	0.00554	0.0114	0.144
10	0.00014	0.00113	0.00560	0.0115	0.159
100	0.00014	0.00117	0.00554	0.0119	0.169

Таблица 1

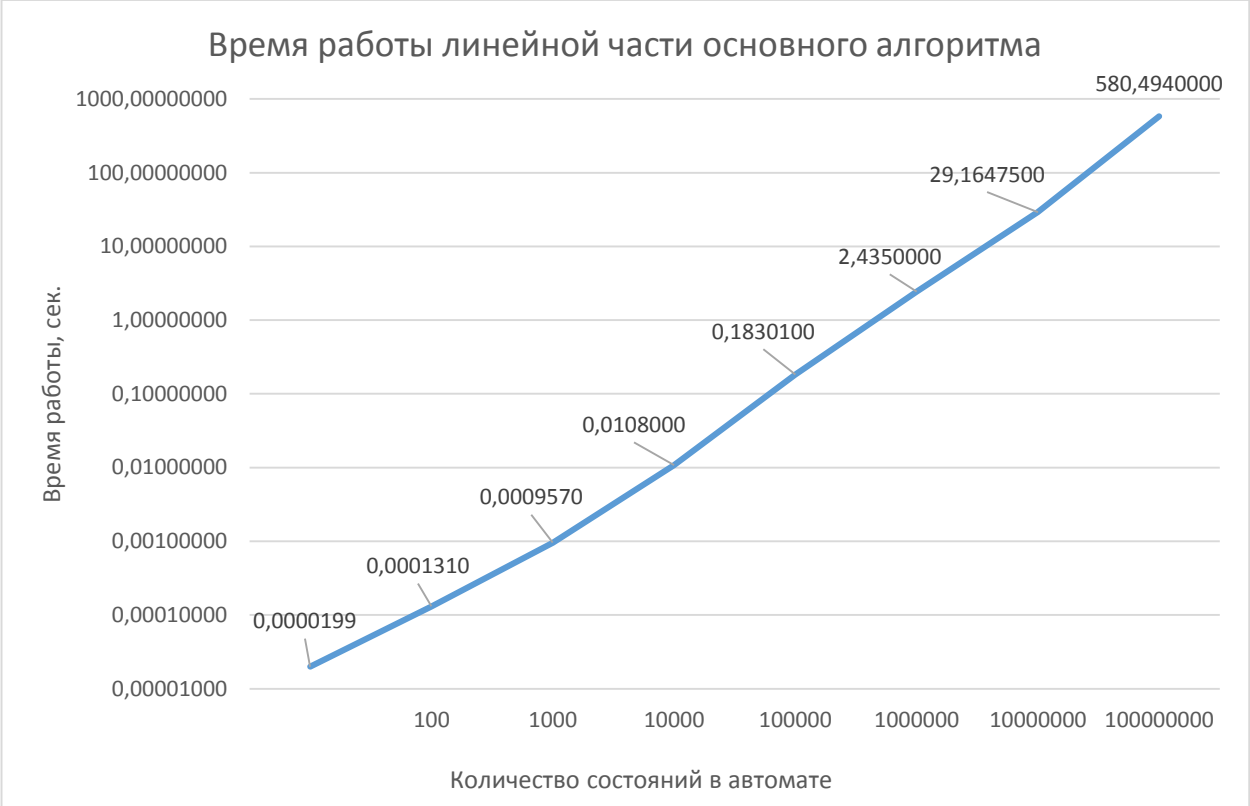

Рис. 1

Опираясь на полученные данные, можно сделать вывод, что результаты экспериментов по вычислению времени работы алгоритма совпадают с теоретическими оценками: при росте n наблюдается линейный рост времени работы. При фиксированном n и различных k значительных изменений во времени выполнения кода не наблюдается.

Экспериментальная оценка сложности основного алгоритма без модификаций вызывает определённые проблемы, поскольку в ходе его выполнения может быть вызван алгоритм $IsSynchronizableSlow$, имеющий

квадратичную сложность. Прежде всего, влияние этой части алгоритма на общее время работы велико. Кроме того, доля случайных автоматов, на которых она будет исполняться, является случайной величиной. Следовательно, необходимо достаточное количество раз повторять многократный запуск основного алгоритма, чтобы иметь возможность воспользоваться статистическими методами. Однако, поскольку даже единичный запуск квадратичного алгоритма уже при $n = 5000$ потребует значительных вычислительных ресурсов, такого рода эксперимент не представляется возможным на описанном ранее оборудовании. Как следствие, для оценки среднего времени работы основного алгоритма на автоматах с n состояниями был использован следующий альтернативный метод:

1. Получить оценку $M[c]$ константы c в асимптотике для вероятности случайного автомата быть синхронизируемым $(1 - \Theta\left(\frac{1}{n}\right))$.
2. Вычислить время t_{linear} работы основного алгоритма на n случайных автоматах с n состояниями без запуска квадратичной проверки в случае невыполнения одного из условий.
3. Вычислить время работы $t_{quadratic}$ алгоритма $IsSynchronizableSlow$ на одном автомате с n состояниями.
4. В качестве оценки среднего времени работы основного алгоритма на одном автомате с n состояниями использовать величину $\frac{t_{linear}+M[c]t_{quadratic}}{n+M[c]}$.

Отметим, что модифицированная версия описанного алгоритма позволяет экспериментально оценить константу c, описанную в пункте 1. Для этого при разных n несколько раз ($t = 100$) производился n-кратный запуск алгоритма на случайных автоматах с n состояниями. В ходе каждого запуска производился подсчёт количества автоматов, для которых алгоритм возвращает значение «$fail$». В результате для различных n была получена выборка X из t значений с элементами x_i, обозначающими количество автоматов, на которых в основном

алгоритме будет вызван $IsSynchronizableSlow$. Для неё были посчитаны выборочные среднее и несмещённая дисперсия. Эти значения для разных n представлены в таблице 2. На рис. 2 приведена гистограмма частот указанной величины при $n = 10000$. Здесь стоит отдельно отметить, что измерения проводились для алгоритма с учётом всех оптимизаций, описанных в конце предыдущего раздела. Без этих оптимизаций среднее значение изучаемой величины достигало двухсот.

$n = \|Q\|$	10	50	100	500	1000	3000	5000	7000	10000
Среднее	5.73	5.74	6.15	5.20	5.09	4.76	5.07	4.76	5.32
Дисперсия	2.22	4.8	7.44	4.52	4.61	4.91	5.36	5.07	5.61
Min	1	0	1	2	1	1	0	0	0
Max	9	11	16	13	11	10	11	10	12

Таблица 2

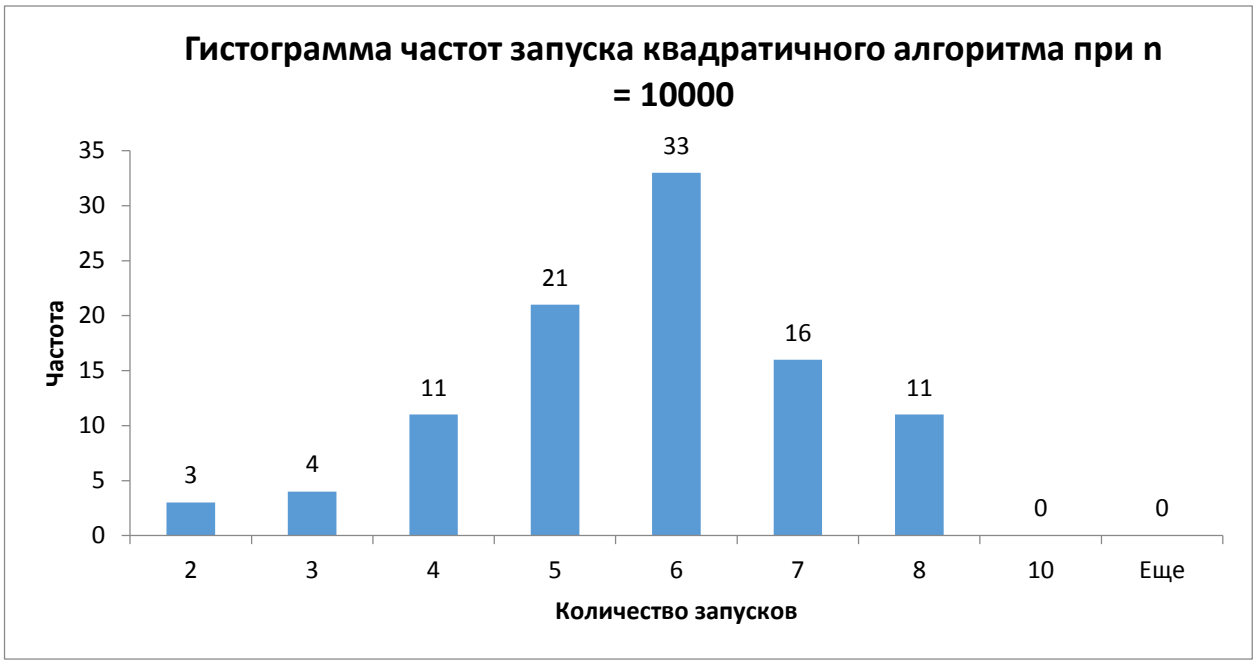

Рис. 2

Наибольший интерес представляет оценка среднего значения, поскольку именно она позволяет убедиться, что введённая случайная величина действительно близка к константе и не зависит от n. Из эксперимента вероятность случайного автомата быть синхронизируемым в среднем равна $1 - \frac{c}{n}$, где c примерно равна

5. Таким образом, результаты полностью согласуются с теоретической оценкой $1 - \Theta\left(\frac{1}{n}\right)$.

Имея полученные данные, легко вычислить оценку из пункта 4. Запускать квадратичный алгоритм при этом потребуется лишь однократно. Результаты оценки среднего времени работы основного алгоритма $k = 2$ и различных n представлены в таблице 3 и на рис. 3. Данные эксперимента говорят о его линейной зависимости от количества состояний в автомате.

$n = \lvert Q \rvert$	1000	2000	3000	4000	5000	7000	9000	10000
$\frac{t_{linear}+M[c]t_{quadratic}}{n+M[c]}$, (сек.)	0.002	0.005	0.008	0.010	0.014	0.021	0.027	0.031

Таблица 3

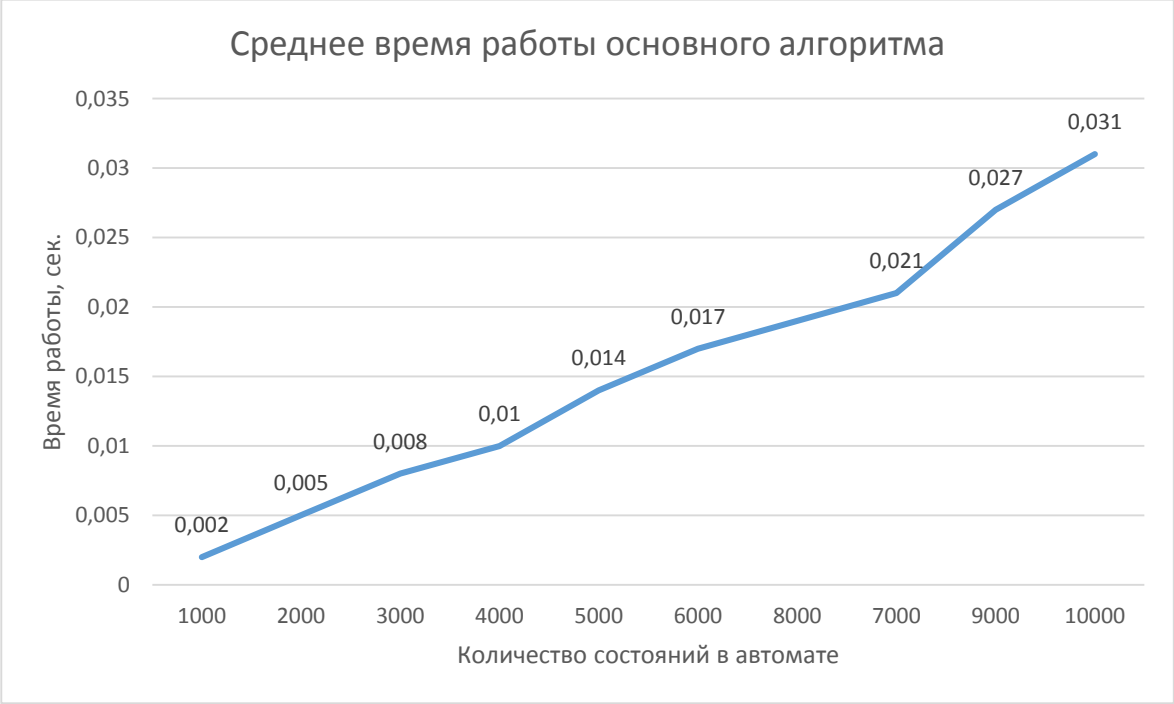

Рис. 3

Крайне важным аспектом в описании производительности любого алгоритма наряду с асимптотической оценкой времени его работы является скрытая константа асимптотики. От неё напрямую зависит практическая применимость алгоритма. Здесь в качестве примера можно привести широко известную структуру данных - кучу Фибоначчи [7], обладающую отличными

теоретическими параметрами, но при этом фактически на любых допустимых на практике размерах входных данных уступающую многим гораздо более простым и теоретически неоптимальным структурам [13].

Для проверки практической пригодности основного алгоритма было проведено его сравнение с квадратичным алгоритмом $IsSynchronizableSlow$. А именно, оба алгоритма запускались достаточно большое количество раз ($t = 1000$) при малых значениях n, чтобы найти такое наименьшее n', при котором среднее время работы $IsSynchronizableSlow$ превышает время работы основного алгоритма. В результате было экспериментально получено $n' = 12$, что позволяет говорить о низкой скрытой константе в асимптотике и, как следствие, о возможности эффективной реализации алгоритма. На рис. 4 приведены графики зависимости работы времени работы алгоритмов от количества состояний в автомате при малых n.

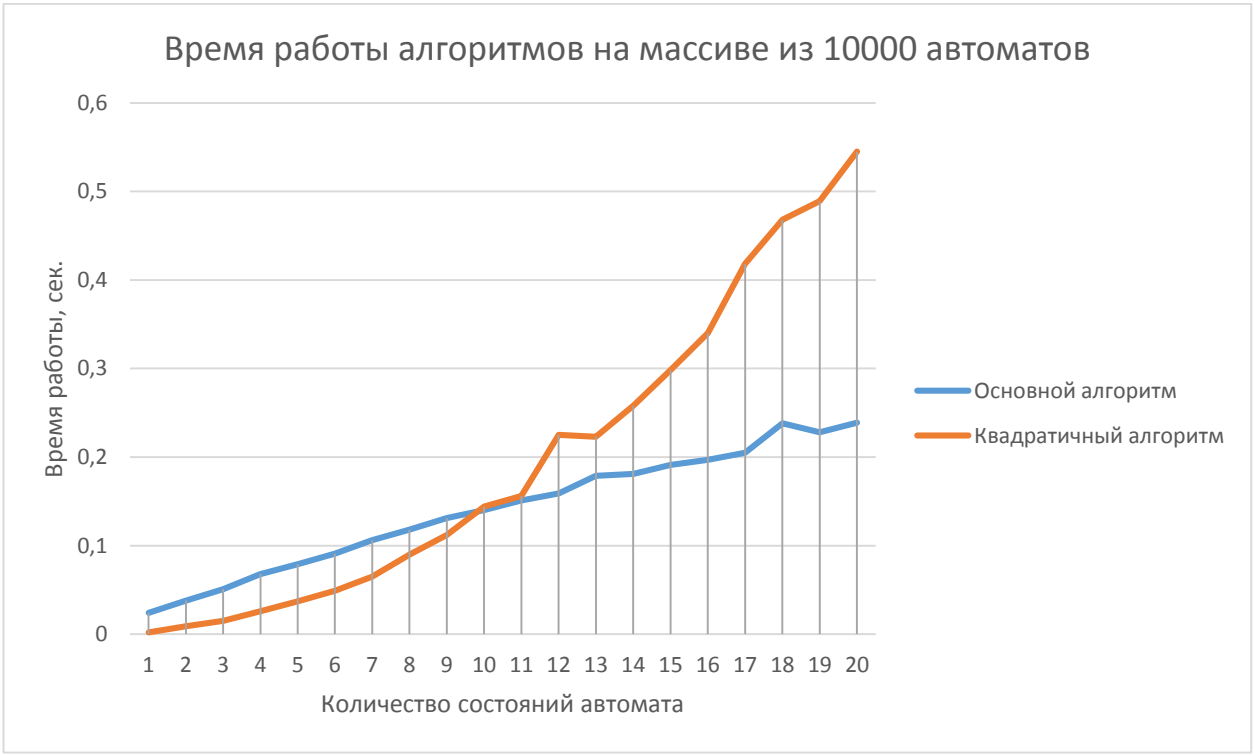

Рис. 4

ЗАКЛЮЧЕНИЕ

В рамках данной работы был описан и реализован алгоритм проверки автомата на синхронизируемость с линейным временем работы в среднем, предложенный в [3] М.В. Берлинковым. Для проверки корректности работы полученной программы критические компоненты кода были покрыты модульными тестами. Кроме того, с целью проверки эффективности работы алгоритма на практике и сравнения с теоретическими результатами был проведён вычислительный эксперимент, основными целями которого были вычисление ожидаемого времени работы алгоритма на массиве случайных данных и оценка среднего количества автоматов, на которых будет запущен квадратичный алгоритм.

Полученные данные полностью согласуются с теорией. Алгоритм действительно показывает линейный рост времени работы в среднем с ростом количества состояний в автомате и при этом не показывает значительной зависимости от размера входного алфавита. Оценка количества автоматов, на которых алгоритм показывает низкую производительность, позволила косвенно подтвердить теоретический результат из [3], приведённый в предложении 2. Более того, как показал эксперимент, описанный алгоритм превосходит классический алгоритм уже при $n = 12$, что говорит о достаточно широких границах его применимости.

СПИСОК ЛИТЕРАТУРЫ

1. Ananichev D., Gusev V., Volkov M. Slowly Synchronizing Automata and Digraphs // Mathematical Foundations of Computer Science, Vol. 6281, 2010. pp. 55-65.
2. Berlinkov M. On a conjecture by Carpi and D'Alessandro // Lecture Notes in Comput. Sci., Vol. 6224, 2010. pp. 66-75.
3. Berlinkov M. On the probability of being synchronizable // SPRINGER-VERLAG, Vol. 9602, 2016. pp. 73-84.
4. Cameron P. // Dixon's theorem and the probability of synchronization. 2011.
5. Černý J. Poznámka k homogénnym eksperimentom s konečnými automatami // Matematicko-fyzikálny Časopis Slovenskej Akadémie Vied, Vol. 14, No. 3, 1964. pp. 208-216.
6. Eppstein D. Reset sequences for monotonic automata // SIAM Journal on Computing, Vol. 19, 1990. pp. 500-510.
7. Fredman M.L., Tarjan R.E. Fibonacci Heaps and Their Uses in Improved Network // Journal of the Association for Computing Machinery, Vol. 34, No. 3, 1987. pp. 596-615.
8. Gerbush M., Heeringa. B. Approximating minimum reset sequences // Implementation and Application of Automata, Vol. 6482, 2011. pp. 154-162.
9. Kari J., Volkov M.V. Černý's conjecture and the road coloring problem // Handbook of Automata, 2013.
10. Kisielewicz A., Kowalski J., Szykuła M. Experiments with Synchronizing Automata // International Conference on Implementation and Application of Automata. Seoul. 2016.
11. Natarajan B.K. Some paradigms for the automated design of parts feeders // Internat. J. Robotics Research, Vol. 8, No. 6, 1989. pp. 89-109.
12. Roman A. Genetic algorithm for synchronization // Language and Automata Theory and Applications, Vol. 5457, 2009. pp. 684-695.
13. Saunders S. A Comparison of Data Structures for Dijkstra's Single Source Shortest Path Algorithm, 1999.

14. Sharir M. A strong connectivity algorithm and its applications to data flow analysis // Computers and Mathematics with Applications, Vol. 7, No. 1, 1981. pp. 67-72.

15. Skvortsov E., Tipkin E. Experimental study of the shortest reset word of random automata // Implementation and Application of Automata, Vol. 6807, 2011. pp. 290-298.

16. Tarjan R. Depth-first search and linear graph algorithms // SIAM Journal on Computing, Vol. 1, No. 2, 1972.

17. Trahtman A.N. The Road Coloring Problem // Israel Journal of Mathematics, Vol. 172, No. 1, 2009. pp. 51-60.

18. Volkov M.V. Synchronizing automata and the Černý conjecture // Language and Automata Theory and Applications, Vol. 5196, 2008. pp. 11-27.