ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2014 Управление, вычислительная техника и информатика № 3 (28)

ДИСКРЕТНЫЕ ФУНКЦИИ И АВТОМАТЫ

УДК 681.3.06

И.Б. Бурдонов, А.А. Косачев

ИССЛЕДОВАНИЕ ГРАФА ВЗАИМОДЕЙСТВУЮЩИМИ АВТОМАТАМИ

Исследование графа автоматами – корневая задача при верификации программно-аппаратных систем на основе формальных моделей, сводимых к графу переходов. При непрерывном росте используемых систем один компьютер уже не справляется с этой задачей по времени или памяти. Возникает задача параллельной и распределённой верификации, формализуемая как исследование графа взаимодействующими автоматами. Предлагается алгоритм такого исследования.

Ключевые слова: исследование графа; взаимодействующие автоматы; параллельная обработка; распределенные системы; верификация.

Задача обхода неизвестного ориентированного графа автоматом используется во многих приложениях. В данной статье подразумевается тестирование детерминированных систем: граф – это граф автомата тестируемой системы, автомат на графе (автомат-обходчик) – тестирующая система, а проход по дуге – это тестовое воздействие и наблюдение результата [1]. В качестве практического примера можно привести работу [2], где выполнялось функциональное тестирование различных подсистем модели процессора: кэш третьего уровня, управление прерываниями и пр. Модельные графы содержали от нескольких тысяч до нескольких миллионов узлов и несколько миллионов дуг. Тест выполнялся максимально на 150 компьютерах.

Автомат-обходчик выполняется на одной машине (процессор с памятью), а наличие нескольких автоматов на разных машинах позволяет существенно распараллелить работу. Автомат-обходчик начинает работу с начальной вершины графа, и за один такт создаётся не более одного автомата-обходчика.

Нижняя оценка времени обхода для одного или ограниченного числа автоматов равна $\Omega(nm)$, где n – число вершин графа, а m – число дуг [3]. Если число автоматов не ограничено, нижняя оценка $\Omega(m)$.

Для того чтобы автомат мог обходить любой конечный граф, требуется доступ по чтению / записи к неограниченной рабочей памяти, в которой накапливается информация о пройденной части графа. Если эта память – часть памяти автомата (машины), автомат не конечен на классе всех графов. Если автомат один, то существуют алгоритмы с оценкой $\Theta(nm)$ [Там же]. Если число автоматов больше одного, но ограничено, то распараллеливание ускоряет обход, но не меняет порядок времени обхода в наихудшем случае [4]. Если число автоматов не ограничено и все вычисления автоматов и передачи сообщений между ними выполняются не дольше (по порядку), чем проход дуги графа, то существуют алгоритмы с оценкой $\Theta(m)$.

Проблема возникает, когда граф не помещается в память машины, что эквивалентно конечности автомата. Есть два подхода.

Первый подход применим, когда рабочая память существует отдельно от памяти автоматов и реализуется на вершинах графа: автомат может писать / читать из текущей вершины символы конечного алфавита. Такой подход может применяться, например, для сети Интернета, когда вершина – это узел сети, а проход по дуге – передача сообщения между узлами.

Для одного конечного автомата известен алгоритм с оценкой $\Theta(nm+n^2\mathrm{loglog}n)$ [5], а при повторном обходе $\Theta(nm+n^2l(n))$, где $l(n)$ – число логарифмирований, при котором достигается соотно-

шение $1 \le \log(\log...(n)...) < 2$ [6]. Отличие от нижней оценки $\boldsymbol{\Omega}(nm)$ объясняется тем, что автомату бывает нужно «вернуться» в начало только что пройденной дуги.

Если конечных автоматов несколько, каждый из них может читать пометки в вершинах, оставленные другими автоматами, и обмениваться с ними сообщениями. Для двух автоматов оценка равна уже $\boldsymbol{\Theta}(nm)$. Эти автоматы двигаются синхронно, кроме случая прохода по новой дуге. В этом случае один автомат (первый) идёт по дуге, а второй остаётся на месте, ожидая сообщения от первого автомата. В этом сообщении указывается, нужно ли второму автомату оставаться на месте, поджидая первый автомат (для возвращения первого автомата в начало только что пройденной дуги), или, наоборот, двигаться вперёд, догоняя первый автомат (возвращения по дуге не требуется).

Второй подход применяется при тестировании, когда вершина графа – это состояние тестируемой системы и автомат ничего не может в неё писать. Разные автоматы могут находиться в разных вершинах графа, т.е. в разных состояниях тестируемой системы. Это означает, что каждый автомат работает со своим клоном тестируемой системы. Будем считать, что за один такт создаётся не более одного клона тестируемой системы с одним автоматом-обходчиком. Тогда рабочая память – это суммарная память коллектива конечных автоматов, обменивающихся сообщениями. Для k машин можно обходить графы, в k раз большие, чем для одной машины. Если размер графа не ограничен, число автоматов в коллективе также должно быть не ограничено.

Этот подход впервые был применён в работе [7], где предложен алгоритм обхода с оценкой $\boldsymbol{O}(m+n^2)$. В настоящей статье мы предлагаем алгоритм с улучшенной оценкой.

1. Формализация автоматов на графе

1.1. Номер выходящей дуги

Автомат на графе, начиная с начальной вершины графа, двигается по дугам, проходя некоторый маршрут. Обход выполнен, если по каждой дуге прошёл хотя бы один автомат. Когда автомат находится в вершине и хочет пройти по выходящей из неё дуге, он эту дугу должен указать. Для этого используется нумерация дуг, выходящих из вершины. Для детерминированных систем разные дуги, выходящие из одной вершины, имеют разные номера. Дуги, выходящие из вершины v, нумеруются от единицы до полустепени выхода вершины (числа выходящих дуг): $1...d_{\text{out}}(v)$. Автомат указывает номер выходящей дуги. Однако для того чтобы выходной алфавит автомата оставался конечным, полустепень выхода вершин графа должна быть ограничена сверху некоторой константой r.

Это ограничение легко снимается с помощью преобразования графа, когда каждая вершина v, в которой полустепень выхода больше r, заменяется на v-цикл вершин. Добавляется $k = [d_{\text{out}}(v)/(r-1)]+1$ вершин, каждая из которых имеет $r-1$ выходящих дуг, кроме последней вершины, из которой может выходить меньше дуг. Эти вершины связываются дополнительными *внутренними* дугами в цикл, который будем называть v-циклом. Тем самым, автомату, находящемуся в некоторой вершине v-цикла, нужно идентифицировать дугу только в пределах тех r дуг, которые выходят из этой вершины, включая внутренную дугу, или меньшего числа дуг для последней вершины. Дуги, заканчивавшиеся в вершине v, теперь будут заканчиваться в первой вершине v-цикла, а остальные вершины v-цикла отличаются тем, что в каждую из них входит ровно одна (внутренняя) дуга. В результате такого преобразования получается граф, в котором полустепень выхода каждой вершины v ограничена r. В дальнейшем для простоты изложения мы будем рассматривать только такие графы, в которых из каждой вершины выходит не более r дуг.

Заметим, что замена вершины v на v-цикл вершин можно понимать как создание одним автоматом, оказавшимся в вершине v, цепочки автоматов – по одному на каждые $r-1$ выходящих дуг или меньшего числа дуг для последнего автомата в цепочке.

1.2. Идентификатор вершины

Для того чтобы автоматы могли обходить граф, автомат, попавший в вершину, должен узнать число выходящих из неё дуг в предположении, что эти дуги занумерованы так, как описано в предыдущем подразделе. Кроме того, чтобы учитывать, какие выходящие дуги в каких вершинах пройдены, необходимо различать вершины. Для этого они снабжаются уникальными идентификаторами. Автомат, попавший в вершину, может узнать её идентификатор. Исключение может быть сделано только для терминальных вершин и вершин, в которые входит только одна дуга; идентификатор этих вершин может быть *пустым*. В первом случае уникальность идентификатора вершины не требуется, поскольку терминальная вершина не имеет выходящих дуг. Во втором случае вершина уникально идентифицируется непустым идентификатором вершины, из которой в данную вершину можно попасть единственным путём, и последовательностью номеров дуг этого пути. Для того чтобы такая вершина с непустым идентификатором всегда существовала, начальная вершина снабжается непустым идентификатором. Заметим, что дополнительные вершины *v*-цикла, о которых шла речь в предыдущем подразделе, имеют по одной входящей дуге и поэтому могут снабжаться пустым идентификатором.

Поскольку идентификатор вершины является входным символом автомата, для того чтобы автомат был конечным, алфавит идентификаторов должен быть конечным, что означает ограниченное число вершин графа с непустыми идентификаторами.

Для удобства изложения будем рассматривать расширенный автомат (EFSM), память которого хранит управляющее состояние, пробегающее конечный алфавит, и конечное число ячеек памяти, в которых могут храниться идентификаторы вершин и только они: в каждой ячейке по одному идентификатору. Эти ячейки будем называть ячейками идентификаторов (вершин).

Для идентификаторов вершин определены следующие операции и только они:

- скопировать идентификатор из указанной ячейки идентификаторов в сообщение, посылаемое другому автомату, как его параметр;
- скопировать идентификатор как параметр сообщения, полученного от другого автомата, в указанную ячейку идентификаторов;
- сравнить на равенство идентификаторы в двух указанных ячейках идентификаторов;
- проверить, является ли идентификатор в указанной ячейке пустым.

1.3. Адрес автомата и среда связи автоматов

Будем считать, что среда связи автоматов позволяет передавать сообщение от любого автомата любому указанному автомату за время, ограниченное сверху константой. При посылке сообщения автомат должен указать получателя сообщения как уникальный адрес автомата. Для конечного автомата число таких получателей должно быть конечным и ограниченным: адрес автомата должен пробегать конечный алфавит. Последнее ограничение приводит к тому, что число автоматов ограничено.

Мы будем считать, что в памяти расширенного автомата имеется конечное число ячеек памяти для хранения адресов автоматов. Эти ячейки будем называть ячейками адресов (автоматов). Тем самым, граф динамических связей автоматов – это конечный ориентированный граф с ограниченной полустепенью выхода вершин (автоматов). Будем считать, что выделен один *пустой* адрес, который не может быть адресом никакого созданного автомата.

Для адресов автоматов определены следующие операции и только они:

- послать сообщение автомату, адрес которого находится в указанной ячейке адресов;
- скопировать адрес из указанной ячейки адресов в посылаемое сообщение как его параметр;
- скопировать адрес как параметр полученного сообщения в указанную ячейку адресов;
- проверить, является ли адрес в указанной ячейке пустым.

Будем предполагать выполнение следующих правил обмена сообщениями:

• для передачи сообщения есть один примитив «послать сообщение», обязательным параметром которого является адрес получателя;

• для получения сообщений есть один примитив «принять сообщение от любого отправителя»;

• сообщения не теряются и не искажаются в среде передачи;

• при передаче нескольких сообщений от одного и того же отправителя одному и тому же получателю нет обгона сообщений.

1.4. Сообщение

Сообщение, передаваемое от одного автомата другому, является как входным, так и выходным символом автомата. Будем считать, что для расширенного автомата сообщение также «расширено»: оно состоит из *тега*, пробегающего конечный алфавит тегов, и ограниченного конечного числа параметров, которыми могут являться только идентификаторы вершин или адреса автоматов.

1.5. Порождение автоматов

Самый первый автомат порождается некоторым *внешним автоматом* и начинает свою работу с ожидания сообщения от этого внешнего автомата. Для порождения остальных автоматов предлагается использовать сообщение ***создай автомат***, посылаемое внешнему автомату. В ответном сообщении ***автомат создан*** указывается адрес порождённого автомата.

Порождаемый автомат будет начинать работать с начальной вершины графа. Это требование объясняется тем, что мы рассматриваем тестирование системы, которая не допускает произвольное клонирование, а допускает только создание нового экземпляра системы в её начальном состоянии. Исключение составляет создание автомата, которому передаётся граф от другого автомата вместе с текущей вершиной. Передатчик теряет связь с графом.

1.6. Взаимодействие с графом

Предлагается рассматривать граф, точнее его экземпляр для каждого взаимодействующего с ним автомата, как отдельный автомат графа. Такой автомат графа создаётся внешним автоматом, когда ему посылается сообщение ***создай граф***. В ответном сообщении ***граф создан*** содержатся адрес (автомата) графа, идентификатор начальной вершины и число дуг, выходящих из начальной вершины. Созданный автомат графа находится в состоянии, которое соответствует начальной вершине графа.

Далее потребуются только одна операция по взаимодействию с графом, которая реализуется как сообщение, посылаемые автомату графа, и ответное сообщение:

1. ***Проход по дуге*** графа с параметром: номер выходящей дуги.

2. Ответное сообщение ***ответ на проход*** с параметрами: идентификатор вершины и число выходящих дуг. Имеются в виду идентификатор конечной вершины указанной дуги и число дуг, выходящих из этой вершины.

1.7. Замечание о ячейках автомата и параметрах сообщений

Если идентификатор вершины занимает a битов памяти, а число вершин в графе с непустым идентификатором равно n_1, то $2^a \geq n_1$. Если адрес автомата занимает b битов памяти, то сгенерировано может быть не более 2^b автоматов. В описываемом ниже алгоритме генерируется не более $2n_2+m$ автоматов, где n_2 – число нетерминальных вершин графа, а m – число дуг графа. Тем самым, $2^b \geq 2n_2+m$. Огрубляя, достаточно ограничений $2^a \geq n$ и $2^b \geq 2n+m$, где n – число вершин графа.

Для удобства изложения будут использоваться дополнительные ячейки и параметры фиксированного размера, кроме тех, что хранят идентификаторы вершин и адреса автоматов.

2. Идея алгоритма

2.1. Граф

Рассматривается обход ориентированного графа с выделенной начальной вершиной и пронумерованными выходящими из каждой вершины дугами. Все вершины снабжены идентификаторами, как было описано в п.1.2. *Началом* и *концом* дуги для краткости будем называть соответственно начальную и конечную вершины дуги. *Терминальной дугой* будем называть дугу, заканчивающуюся в терминальной вершине.

После инициализации в каждый момент времени имеются:

• *Пройденный граф* – подграф графа, определяемый всеми пройденными дугами.

• *Пройденное дерево* – ориентированный от корня, которым является начальная вершина, остов подграфа, получаемого из пройденного графа удалением терминальных дуг. Такое пройденное дерево содержит все нетерминальные пройденные вершины пройденного графа.

В начале работы алгоритма, после инициализации, пройденный граф совпадает с пройденным деревом и состоит из одной начальной вершины графа.

В процессе работы алгоритма пройденный граф и пройденное дерево увеличиваются (добавляются новые дуги и вершины). Пройденное дерево, получающееся в конце работы, будем называть *законченным деревом*. Под *хордой* будем понимать нетерминальную дугу, не принадлежащую законченному дереву; пройденная хорда является хордой пройденного дерева. Для данной вершины *входящей* дугой будем называть дугу законченного дерева, заканчивающуюся в этой вершине. У начальной вершины нет входящей дуги, а в любую другую вершину входит ровно одна дуга.

В конце работы алгоритма получается разметка графа: для каждой дуги в её начале указывается её тип: дуга законченного дерева, терминальная дуга или хорда.

2.2. Работа автоматов

Автомат работает в трёх режимах: генератор, регулятор, движок. Обход графа, т.е. движение по его дугам, выполняют движки, регуляторы предназначены для управления перемещением движков по дугам, а генератор создаёт новые движки и связанные с ними копии графов.

Каждый создаваемый генератором движок предназначен для того, чтобы, начиная с начальной вершины графа, пройти путь в пройденном дереве, после чего пройти непройденную дугу. С каждой пройденной нетерминальной вершиной связан ровно один регулятор, который направляет движки, приходящие в эту вершину, по тем или иным выходящим из вершины дугам. Для этого движок, оказавшийся в данной вершине, спрашивает у регулятора этой вершины (посылает ему сообщение) ***куда идти***, а регулятор в ответном сообщении ***иди по дуге*** сообщает номер выходящей дуги. В некоторой ситуации (описанной ниже) движок оказывается «лишним» и останавливается, не пройдя ни одной непройденной дуги.

Автомат (копии) графа, связанный с движком, «помнит» текущую вершину графа, в которой находится движок. Для прохода по дуге движок посылает автомату графа сообщение ***проход по дуге***, задавая номер выходящей дуги, и получает в сообщении ***ответ на проход*** идентификатор конца дуги и число выходящих из него дуг.

Вначале извне создаётся генератор. После инициализации генератор одновременно становится регулятором начальной вершины, которая в этот момент времени является единственной пройденной вершиной и единственной вершиной пройденного дерева, а также создаёт первый движок, находящийся в начальной вершине графа.

Для управления обходом графа используются сообщения ***запрос*** и ***конец***. Такое сообщение посылается «по пройденной дуге» в обратном направлении: от автомата, находящегося в конце дуги, регулятору начала дуги. Сообщение ***запрос*** посылается регулятором вершины по входящей дуге и означает, что по этой дуге нужно послать один движок, когда будет такая возможность. Следующий

движок по этой дуге можно посылать после того, как будет получен новый ***запрос***. Сообщение ***конец***, посылаемое по пройденной дуге, означает, что по этой дуге больше никогда не нужно посылать движки. Сообщение ***конец*** посылает по хорде или терминальной дуге движок, который эту дугу проходит. Кроме того, сообщение ***конец*** посылает регулятор вершины по входящей дуге, если в пройденном дереве выше этой вершины нет вершин, из которых выходят непройденные дуги. Это происходит тогда, когда по всем дугам, выходящим из вершины, получено сообщение ***конец***. У начальной вершины нет входящей дуги, поэтому регулятор начальной вершины посылает сообщение ***конец*** внешнему автомату, что означает конец работы алгоритма. ***Запросы*** посылаются с некоторым опережением, поэтому может оказаться, что движку, пришедшему в вершину, некуда двигаться. Такой движок становится *ждущим* в вершине (запоминается регулятором вершины). Далее либо по выходящей дуге придёт ***запрос***, и движок будет послан по этой дуге, либо по всем выходящим дугам будут получены сообщения ***конец***, и тогда движок оказывается «лишним» – он должен остановиться.

Каждая дуга имеет в регуляторе её начала три состояния:

• *Активная* = дуга, по которой нужно посылать движки. Это может быть либо непройденная дуга (такие дуги всегда активны), либо дуга пройденного дерева, по которой пришёл ***запрос***, но движок по дуге ещё не послан.

• *Пассивная* = пройденная дуга, по которой пока не нужно посылать движки. Это дуга, по которой был послан движок, но после этого по дуге не приходили сообщения ***запрос*** или ***конец***.

• *Законченная* = пройденная дуга, по которой больше никогда не нужно посылать движки. Это дуга, по которой пришло сообщение ***конец***.

На рис. 1 изображена схема изменения состояния дуги под влиянием сообщений. В начале все дуги не пройдены, т.е. они активны.

Регулятор посылает движки по активным выходящим дугам «по кругу». Для этого у регулятора есть номер текущей выходящей дуги i. Это дуга, по которой последний раз посылался движок, или $i = 0$ в самом начале. Движок, находясь в вершине пройденного дерева, спрашивает у регулятора вершины, по какой дуге ему идти, посылая сообщение ***куда идти***. Регулятор корректирует номер i текущей дуги: новой текущей дугой становится следующая после дуги i активная дуга в цикле выходящих дуг, а если активных дуг больше нет, то $i := 0$. Если после коррекции $i>0$, регулятор посылает движку сообщение ***иди по дуге*** с номером дуги i, а сам по входящей дуге посылает ***запрос***. У начальной вершины нет входящей дуги, поэтому сообщение ***запрос*** не посылается, а генератор продолжает генерацию движков. Если $i = 0$, регулятор запоминает адрес движка, движок становится ждущим, ***запрос*** не посылается. Если ждущий движок образуется в начальной вершине, генератор приостанавливает генерацию движков.

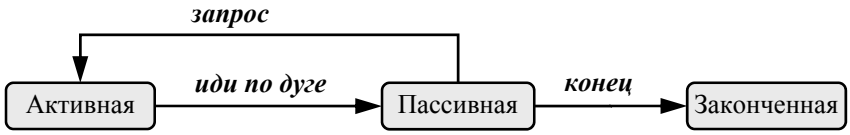

Рис. 1. Схема изменения состояния дуги

После того как появляется ждущий движок, возможны два случая: 1. Регулятор в дальнейшем получает сообщение ***запрос*** с указанием дуги, по которой регулятор направляет ждущий движок (сообщение ***иди по дуге***) и сам посылает сообщение ***запрос*** регулятору начала входящей дуги. У начальной вершины нет входящей дуги, поэтому сообщение ***запрос*** не посылается, а генератор возобновляет генерацию движков. 2. Регулятор обнаруживает, что все выходящие дуги стали законченными (получено сообщение ***конец*** по последней незаконченной выходящей дуге). Регулятор посылает ждущему движку сообщение ***иди по дуге*** с нулевым номером дуги (это «лишний» движок и он останавливается) и сообщение ***конец*** регулятору начала входящей дуги. У начальной вершины нет входящей дуги, поэтому сообщение ***конец*** посылается внешнему автомату и генератор прекращает генерацию движков.

Как создаются регуляторы? Рассмотрим подробнее работу автоматов, связанную с проходом движком непройденной ранее дуги $a–i→b$, ведущей из вершины a, имеющей в ней номер i и ведущей в вершину b.

• Если вершина b терминальная, для неё не создаётся регулятор. Движок посылает регулятору вершины a сообщение ***конец*** с признаком «терминальная вершина» и номером дуги i, а сам останавливается.

• Если вершина b нетерминальная, то движок должен узнать по идентификатору вершины, пройдена она или ещё нет, т.е. имеется ли регулятор этой вершины. Вершина b заведомо не пройдена, если её идентификатор пуст (в такую вершину входит только одна дуга). В противном случае производится опрос регуляторов, описанный ниже. Итак:

  • Если вершина b уже пройдена, то движок прошёл по хорде. Движок посылает регулятору вершины a сообщение ***конец*** с признаком «хорда», номером дуги i и адресом регулятора вершины b, а сам останавливается.

  • Если вершина b ещё не пройдена, то движок сам становится регулятором вершины b и создаёт новый движок в текущей вершине b, передавая ему граф. После того как этот новый движок будет создан и послан по выходящей дуге, новый регулятор вершины b посылает регулятору вершины a сообщение ***запрос*** с указанием номера дуги i и своим адресом как адресом регулятора вершины b.

Опрос регуляторов выполняется тогда, когда движок проходит по непройденной дуге и оказывается в нетерминальной вершине с непустым идентификатором. В этом случае ему нужно найти регулятор той вершины, в которой он оказался, или самому стать регулятором, если поиск не увенчался успехом. Этот поиск выполняется по идентификатору вершины: две вершины не могут иметь один и тот же непустой идентификатор. Для опроса регуляторов используется односторонний список регуляторов вершин с непустыми идентификаторами, регулятор начальной вершины (он же генератор) всегда находится в голове списка (напомним, что идентификатор начальной вершины всегда не пустой). Движок посылает по списку сообщение ***опрос***, в котором указываются (непустой) идентификатор вершины и адрес движка. Если регулятор вершины уже есть, он, получив такое сообщение, посылает ***ответ на опрос***, указывая движку свой адрес. Если регулятора вершины ещё нет, то сообщение ***опрос*** доходит до последнего в списке регулятора, который пересылает его движку, а сам ставит у себя ссылку по списку на этот движок. Получив обратно своё сообщение ***опрос***, движок сам становится регулятором – последним в списке, и создаёт новый движок, передавая ему свой адрес (как адрес регулятора текущей вершины) и адрес графа.

3. Оценка сложности алгоритма

Будем считать, что времена передачи сообщения, прохода движка по дуге и срабатывания автомата ограничены сверху и снизу ненулевым положительным значением.

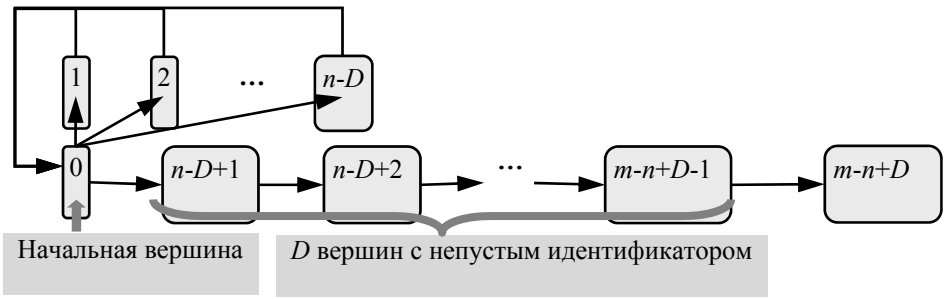

Рис. 2. Пример графа, где алгоритм работает время $\boldsymbol{\Omega}(m+nD)$

Тогда алгоритм заканчивает работу за время $\boldsymbol{O}(m+n_0D)$, где m – число дуг в исходном графе, n_0 – число нетерминальных вершин с непустым идентификатором, D – максимальное число нетерми-

нальных вершин с непустым идентификатором на пути в графе от начальной вершины, не считая самой начальной вершины. Поскольку $n_0 \le n$, верна также оценка $\boldsymbol{O}(m+nD)$. Эта оценка достигается на графе, изображённом на рис. 2. Поэтому суммарная оценка $\boldsymbol{\Theta}(m+nD)$.

Заключение

Дальнейшие исследования обхода графов коллективом автоматов возможны по нескольким направлениям.

Первое направление: недетерминированные графы. Граф недетерминирован, если одному номеру выходящей дуги соответствует несколько дуг с общей начальной вершиной и разными конечными вершинами. Обход такого графа может пониматься двояко: полный обход, когда требуется проход по каждой дуге графа, или частичный обход, когда нужно пройти хотя бы по одной дуге с данной начальной вершиной и данным номером выходящей дуги. Частичный обход применяется, когда интересующие нас свойства графа могут быть исследованы при таком обходе. Для того чтобы гарантировать возможность обхода недетерминированного графа, нужны дополнительные гипотезы. Например, гипотеза об ограниченном недетерминизме, которая утверждает, что, проходя t раз из данной вершины по выходящей дуге с данным номером, мы пройдём все выходящие из этой вершины дуги с этим номером, где t – заранее известная константа. Существуют алгоритмы такого обхода одним автоматом [8]. Можно поставить задачу разработки нового алгоритма такого обхода коллективом взаимодействующих автоматов. Для частичного обхода может использоваться так называемая Δ-достижимость вершин графа из начальной вершины [9]. Неформально это означает, что существует алгоритм, который гарантированно обеспечивает достижение заданной вершины при любом недетерминированном выборе выходящей дуги в рамках множества дуг с тем же началом и тем же номером дуги.

Второе направление: использование внешней памяти. Предложенный в данной статье алгоритм обходит граф только в том случае, когда регуляторы всех нетерминальных вершин помещаются в суммарной памяти машин. Если это не так, становится актуальным вопрос об использовании дополнительной внешней памяти. Обход графа коллективом автоматов с использованием внешней памяти лежит на стыке двух направлений: методы эффективной работы с внешней памятью, в частности стратегий страничной и сегментной подкачки, и алгоритмы обхода графов коллективом автоматов.

Третье направление: графы специального вида. Большой интерес представляет разработка новых алгоритмов исследования графов, принадлежащих тому или иному подклассу графов. Такие подклассы имеет смысл рассматривать при сочетании двух условий: эти подклассы имеют практическое значение (графы переходов исследуемых систем и сетей относятся к такому подклассу) и на этих подклассах алгоритмы могут работать быстрее.

ЛИТЕРАТУРА

1. *Бурдонов И.Б., Косачев А.С., Кулямин В.В., Петренко А.К.* Подход UniTesK к разработке тестов // Программирование. 2003. № 6. С. 25–43.
2. *Demakov A., Kamkin A., Sortov A.* High-Performance Testing: Parallelizing Functional Tests for Computer Systems Using Distributed Graph Exploration. In Open Cirrus Summit 2011. Moscow, 2011.
3. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай // Программирование. 2003. № 5. С. 59–69.
4. *Бурдонов И.Б., Грошев С.Г., Демаков А.В., Камкин А.С., Косачев А.С., Сортов А.А.* Параллельное тестирование больших автоматных моделей // Вестник ННГУ. 2011. № 3(2). С. 187–193.
5. *Бурдонов И.Б.* Обход неизвестного ориентированного графа конечным роботом // Программирование. 2004. № 4. С. 11–34.
6. *Бурдонов И.Б.* Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом // Программирование. 2004. № 6. С. 6–29.
7. *Бурдонов И.Б., Косачев А.С.* Обход неизвестного графа коллективом автоматов // Труды Международной конференции «Научный сервис в сети Интернет: все грани параллелизма». М. : Изд-во МГУ, 2013. С. 228–232.
8. *Бурдонов И.Б., Косачев А.С.* Полное тестирование с открытым состоянием ограниченно недетерминированных систем // Программирование. 2009. № 6. С. 3–18.
9. *Бурдонов И.Б., Косачев А.С., Кулямин В.В.* Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай // Программирование. 2004. № 1. С. 2–17.

Бурдонов Игорь Борисович, д-р физ.-мат. наук. E-mail: igor@ispras.ru
Косачев Александр Сергеевич, канд. физ.-мат. наук. E-mail: kos@ispras.ru
Института системного программирования РАН (г. Москва) Поступила в редакцию 25 января 2014 г.

Burdonov Igor B., Kossatchev Alexander S. (Institute for System Programming of the RAS, Moscow, Russian Federation).
Graph learning by interacting automata.
Keywords: graph learning; interacting automata; parallel processing.

The paper describes the learning an unknown oriented graph by automata traversing the arcs of the graph and exchanging messages. The learning completes when each arc has been traversed by at least one automaton.

Section 1 discusses the problems of identification of arcs and nodes in the graph, ways of exchanging messages between automata and interaction of the automata with the graph. The arcs outgoing from the same node are considered enumerated. When traversing the graph, each automaton indicates the number of the arc outgoing from the current node. The nodes have unique identifiers. The exceptions are sinks and nodes with only one incoming arc: these nodes can have empty identifier. Automaton can recognize the identifier of the current node and the number of outgoing arcs. The automata interaction media provides message transfer from one automaton to another in a limited time. The following primitives are recognized: "send message" indicating the receiver address and "receive message from any sender".

Section 2 describes the idea of the algorithm. Automata are divided into *regulators* and *runners*. Runners traverse the arcs of the graph. When in a new (yet untraversed) node, runner becomes its regulator. The node regulator controls the runners travel along the arcs outgoing from this node. The regulator of the initial node, in addition, generates new runners. During execution, the graph description is created in the cumulative memory of the regulators as a spanning tree of the already traversed part of the graph oriented form the initial node.

To navigate the runners, the following message is used: "***where to***" from runner to regulator and response message "***traverse the arc***" indicating the arc number and the address of the regulator of the arc terminal node if this arc has already been traversed. Regulator sends runners either along the arcs of the tree or along untraversed arcs. After traversing an untraversed arc, runners learn the identifier of its terminal node, but it needs to know whether this node has been already traversed. To get this information, the "***poll***" message is sent iterating all regulators linked into a list of addresses. If the node has been traversed, the node regulator sends the response message to the runner indicating its address, and the runner resends this address to the regulator of the initial node of the traversed chord in the "***finish***" message. This message is also used to indicate the traversal end: the regulator of the tree arc terminal node sends "***finish***" to the regulator of the arc initial node if it has received "***finish***" along all outgoing arcs. The traversal is over when regulator of the initial node receives "***finish***" along all outgoing arcs.

Section 3 gives the estimate of the traversal time $\Theta(m+nD)$, where n is a number of nodes, m is a number of arcs, and D is the maximal number of nodes with nonempty identifiers on the graph path from the initial node.

The conclusion describes possible directions of further work: learning of nondeterministic graphs, use of external memory, learning graphs of special kind.

REFERENCES

1. Bourdonov I.B., Kossatchev A.S., Petrenko A.K., Kuliamin V.V. The UniTesK approach to designing test suites. *Programming and Computer Software*, 2003, vol. 29, no. 6, pp. 310-322. DOI: 10.1023/B:PACS.0000004131.54590.fb.
2. Demakov A., Kamkin A., Sortov A. *High-performance testing: Parallelizing functional tests for computer systems using distributed graph exploration*. In Open Cirrus Summit. Moscow, 2011.
3. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Irredundant algorithms for traversing directed graphs: The deterministic case. *Programming and Computer Software*, 2003, vol. 29, no. 5, pp. 245-258. DOI: 10.1023/A:1025733107700.
4. Bourdonov I.B., Groshev S.G., Demakov A.V., Kamkin A.S., Kossatchev A.S, Sortov A.A. Parallel testing of large automata models. *Vestnik NNGU – Vestnik UNN*, 2011, no. 3(2), pp. 187-193. (In Russian).
5. Bourdonov I.B. Obkhod neizvestnogo orientirovannogo grafa konechnym robotom [Backtracking problem in the traversal of an unknown directed graph by a finite robot]. *Programmirovanie*, 2004, vol. 30, no. 6, pp. 305-322.
6. Bourdonov I.B. Problema otkata po derevu pri obkhode neizvestnogo orientirovannogo grafa konechnym robotom [Traversal of an unknown directed graph by a finite robot]. *Programmirovanie*, 2004, vol. 30, no. 4, pp. 188-203.
7. Bourdonov I.B., Kossatchev A.S. [Unknown graph traversing by automata group]. *Trudy Mezhdunarodnoj superkomp'yuternoj konferentsii "Nauchnyj servis v seti Internet: vse grani parallelizma"* [The proceeding of Russian Supercomputer conference "Scientific service of Internet"]. Moscow: MSU Publ., 2013, pp. 228-232. (In Russian).
8. Bourdonov I.B., Kossatchev A.S. Polnoe testirovanie s otkrytym sostoyaniem ogranichenno nedeterminirovannykh sistem [Complete open state testing of limitedly nondeterministic systems]. *Programmirovanie*, 2009, vol. 35, no. 6, pp. 301-313.
9. Bourdonov I.B., Kossatchev A.S., Kuliamin V.V. Neizbytochnye algoritmy obkhoda orientirovannykh grafov. Nedeterminirovannyy sluchay [Irredundant algorithms for traversing directed graphs: The Nondeterministic case]. *Programmirovanie*, 2004, vol. 30, no. 1, pp. 2-17.