И. Бурдонов, А. Косачев.
Исследование графов коллективом двигающихся автоматов.
Программная инженерия. т.7, №12, 2016 , стр. 559-567.
9 стр.

УДК

И. Б. Бурдонов, д-р физ.-мат. наук, вед. науч. сотр., igor@ispras.ru,

А. С. Косачев, канд. физ.-мат. наук, вед. науч. сотр., kos@ispras.ru.

Институт системного программирования РАН, Москва

Исследование графов коллективом двигающих автоматов

Эта статья – вторая в серии трёх статей, посвящённых исследованию графов автоматами. Если в первой статье автомат был один, то в данной статье по дугам ориентированного графа двигаются несколько автоматов, обмениваясь сообщениями по независимой от графа сети связи. Также изучается возможность обхода недетерминированного графа. В следующей статье будет рассмотрена модель с неподвижными автоматами, находящимися в вершинах, и посылающих сообщения только по дугам графа.

Ключевые слова: ориентированный граф, упорядоченный граф, нумерованный граф, неизвестный граф, недетерминированный граф, обход графа, Δ-обход, автомат, робот, полуробот.

1. Введение

Эта статья является второй в серии статей по исследованию графов автоматами. В первой статье [1] рассматривался обход графа одним автоматом, двигающимся по рёбрам графа. В данной статье задача обхода графа решается коллективом автоматов, которые могут не только двигаться по рёбрам графа, но и обмениваться между собой сообщениями по сети связи, независимой от графа. В конце статьи рассмотрен случай недетерминированного графа. В следующей, третьей, статье будет рассмотрена «инвертированная» модель, в которой автоматы неподвижно «сидят» в вершинах графа, но могут обмениваться между собой сообщениями, передаваемыми по рёбрам графа.

Сначала повторим кратко основные понятия, введённые в [1].

Мы будем рассматривать ориентированные графы, а их рёбра, как это принято, будем называть дугами. Автомат на графе можно рассматривать как

аналог машины Тьюринга: вместо ячеек ленты машины используются вершины графа, а вместо движения по ленте влево или вправо выполняется движение по дугам графа в направлении их ориентации.

Автомат, который конечен на всём рассматриваемом классе графов, будем называть *роботом*. Если автомат не конечный, но граф не помещается в его память, то такой автомат будем называть *полуроботом*. По сути, это то же самое, что локальный алгоритм на графе. А если граф помещается в память автомата, то это *неограниченный автомат*.

Обход графа. Как правило, исследование графа сводится к *обходу графа*, под которым понимается построение и проход автоматом маршрута, содержащего все дуги (и, тем самым, все вершины) графа. Когда автоматов много и они работают параллельно, можно говорить о покрытии дуг графа не одним маршрутом, а набором маршрутов. Иными словами, при обходе требуется, чтобы по каждой дуге прошёл хотя бы один автомат. Автомат начинает работать с выделенной начальной вершины графа, которую для краткости будем называть *корнем*.

Структура на графе. При любом обходе графа возникает структура на графе, при описании которой будем использовать три «краски»: белую, серую и чёрную. Непройденные дуги, т.е. дуги, по которым автомат не проходил, окрашены в белый цвет; пройденные дуги серые или чёрные. Непройденная вершина, т.е. вершина, в которой автомат ещё не был, тоже белая. Пройденная вершины чёрная, если автомат прошёл по всем выходящим из вершины дугам (все они серые или чёрные), иначе вершина серая (есть выходящая белая дуга). *Пройденный граф* – подграф, порождённый пройденными дугами (серыми и чёрными). Он представляет собой цепочку компонентов сильной связности, причём из каждого компонента, кроме последнего, ведёт одна дуга в следующий компонент. Эта дуга называется *разделяющей*, а её конец – корнем следующего компонента. Корень графа понимается как корень первого

компонента. *Прямая дуга* – это дуга, по которой автомат первый раз пришёл в какую-то вершину; прямые дуги порождают прямое дерево – остов пройденного графа, ориентированный от корня. Остальные пройденные дуги будем называть *хордами* прямого дерева. Серое дерево – это поддерево прямого дерева, содержащее корень и все серые вершины, причём все его листья тоже серые. Дуги серого дерева серые, остальные прямые дуги и хорды чёрные.

Три типа алгоритмов. Основные типы алгоритмов обхода – это DFS (обход в глубину), BFS (обход в ширину) и «жадный» алгоритм.

В DFS-алгоритме серое дерево состоит только из одного пути. Пройдя хорду, автомат по пройденному графу всегда возвращается в начало хорды. Это называется *откат по хорде*. Если конец серого пути становится чёрным, автомат возвращается в начало последней дуги серого пути и «перекрашивает» эту дугу в чёрный цвет. Это называется *откат по дереву*.

В BFS-алгоритме серое дерево может «ветвиться». После прохода хорды автомат движется по пройденному графу не до начала хорды, а просто до серого дерева, а потом по нему «вверх» до ближайшей серой вершины. Если листовая вершина серого дерева становится чёрной, то выполняется откат по дереву в начало той дуги дерева, которая заканчивается в этой листовой вершине, и эта дуга «перекрашивается» в чёрный цвет.

Оба эти алгоритма для откатов используют лес *обратных деревьев*. Обратное дерево – это остов сильно-связного компонента пройденного графа, ориентированный к корню компонента.

«Жадный» алгоритм не использует прямое и обратные деревья. Каждый раз, когда текущая вершина становится чёрной, по пройденному графу определяется путь из текущей вершины в ближайшую серую вершину, который автомат и проходит.

Упорядоченный граф. Упорядоченным графом называется граф, в котором дуги, выходящие из вершины, перенумерованы, начиная с 1, – для каждой

вершины задана своя нумерация выходящих дуг. Для движения по графу автомат, находящийся в вершине, указывает номер выходящей дуги, по которой он «хочет» пройти. Поскольку номер дуги является выходным символом автомата, для конечности автомата нужно, чтобы полустепень исхода вершины была ограничена сверху. Это ограничение обычно обходится с помощью преобразования каждой вершины в цепочку вершин, каждая из которых имеет полустепень исхода не больше 2 (Рис. 1).

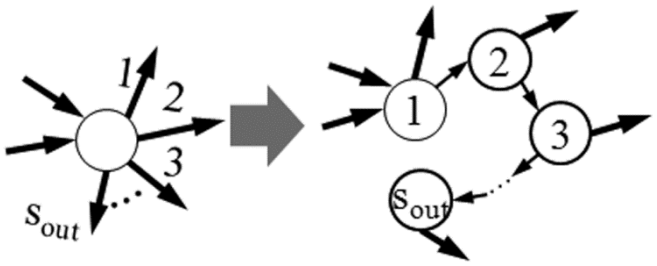

Рис. 1. Преобразование вершины в цепочку вершин.

Неизвестный граф, неизбыточные и свободные автоматы. Если граф известен и помещается в память автомата, то можно вычислить его обход минимальной длины, а потом пройти по этому маршруту. Граф неизвестен, если он неизвестен заранее, до прохода по всем его дугам. На каждом шаге автомат может «знать» только часть графа: пройденный граф и, быть может, что-то о белых дугах, выходящих из серых вершин. Автомат *неизбыточный*, если он «знает» полустепень исхода каждой пройденной вершины. Автомат *свободный*, если он этого не «знает», но «узнаёт» о наличии или отсутствии белой выходящей дуги при попытке пройти по ней. Если дуга с указанным номером есть, автомат идёт по ней, а если нет, то получает отказ и остаётся на месте. Для свободного автомата, допуская вольность речи, мы будем говорить о цвете вершин и дуг «с точки зрения» автомата: если вершина чёрная, но автомат ещё не «знает» об этом, то для него она серая; соответственно понимается серое дерево.

И. Бурдонов, А. Косачев.
Исследование графов коллективом двигающихся автоматов.
Программная инженерия. т.7, №12, 2016 , стр. 559-567.
9 стр.

Нумерованный граф. Если все вершины графа пронумерованы и в каждой вершине записан её номер, то такой граф будем называть нумерованным. Робот, как конечный автомат, не может использовать номер вершины, поскольку этот номер неограничен на любом классе графов с произвольным числом вершин. Но для полуробота и неограниченного автомата номер вершины бывает полезным. Такой автомат может и сам нумеровать вершины графа в порядке их обнаружения (когда первый раз попадает в вершину). Неограниченный автомат по мере обхода может запоминать в своей памяти весь пройденный граф. Для такого автомата номер вершины – это единственное, что нужно хранить в самой вершине, поскольку всё остальное он может хранить в своей памяти, индексируя номером вершины. На нумерованном графе неограниченный автомат может только читать из вершины её номер и не будет в неё писать.

Обмен сообщениями. При обходе графа коллективом автоматов предполагается, что автоматы могут не только читать и писать в вершины графа, но и обмениваться между собой сообщениями по независимой от графа сети связи. Предполагается, что сеть связи гарантирует прохождение сообщения «от точки к точке», т.е. от автомата-отправителя к автомату-получателю по адресу последнего.

Генерация автоматов. Автоматы генерируются в корне графа и оттуда двигаются по дугам графа. Сам обход понимается как покрытие дуг графа маршрутами, которые проходятся всеми автоматами, то есть по каждой дуге должен пройти хотя бы один автомат.

2. Обход известного графа неограниченными автоматами

Теперь рассмотрим задачу обхода графа коллективом неограниченных автоматов. Автоматы генерируются в корне графа, автомат, который генерирует другие автоматы, будем называть *генератором*, а остальные – *движками*. Будем предполагать, что число генерируемых автоматов не ограничено. Граф не обязательно сильно связный, предполагается, что

покрываются дуги графа достижимости. Поскольку, кроме прохода дуг, есть ещё обмен сообщениями, будем оценивать время обхода, считая, что пересылка сообщения и проход по дуге занимают не больше 1 такта.

Сначала рассмотрим случай известного графа. Автоматы не пишут в вершины, и не читают из них. Граф известен генератору и все нужные пометки он может делать в своей памяти. Генератор сообщает каждому движку, куда ему идти, в виде последовательности номеров дуг. Если за 1 такт генерируется 1 движок, то время обхода на самом плохом графе равно m (Рис. 2 слева вверху), а на самом хорошем графе $\boldsymbol{O(\sqrt{m})}$ (Рис. 2 слева внизу). Если за 1 такт генератор может сгенерировать неограниченное, но конечное, число движков, то для известного графа генератор может сгенерировать сразу m движков. Время обхода на самом плохом графе равно $d+1$ (Рис. 2 справа вверху), а на самом хорошем графе равно d (Рис. 2 справа внизу).

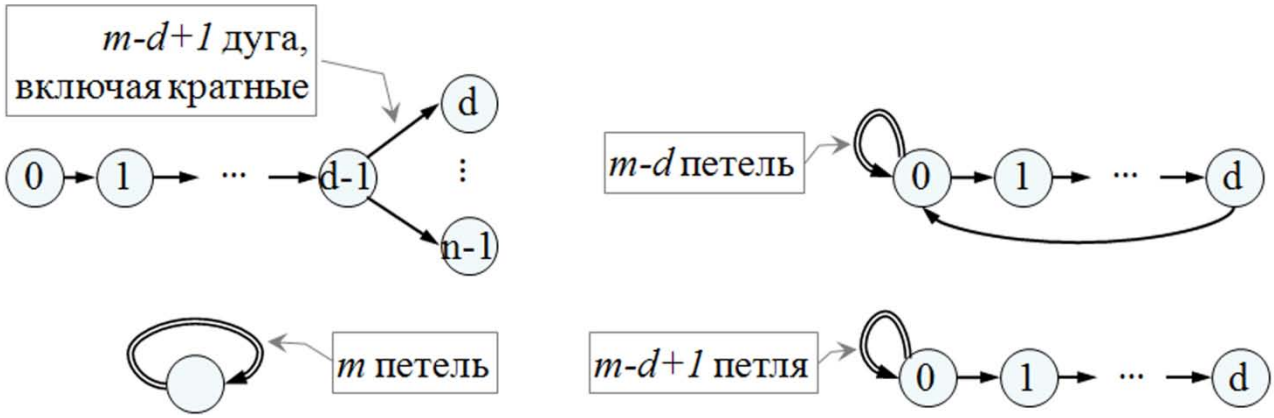

Рис. 2. Примеры графов для коллектива неограниченных автоматов.

3. Обход неизвестного графа неограниченными автоматами

Для неизвестного графа в памяти генератора хранится только пройденный граф. Когда движок попадает в новую вершину, ему надо узнать её номер и, если автомат неизбыточный, то ещё и полустепень исхода вершины. Для простоты будем считать, что граф нумерованный. Генератор на каждом такте сообщает движку, по какой дуге ему сейчас идти. Движок либо идёт по указанной дуге и сообщает генератору номер вершины в конце этой дуги, либо

(если автомат свободный) получает отказ, если дуги с таким номером нет, и сообщает об этом генератору. После чего движок ждёт дальнейших указаний. Генератор направляет движок по непройденной дуге, а если её нет, то по любой дуге, через которую достижима серая вершина, если же и такой дуги нет, то генератор останавливает движок.

Если за 1 такт генерируется 1 движок (*алгоритм 1*), то порядок времени обхода не меняется по сравнению с известным графом, $O(m)$, даже для свободного автомата. Действительно, будем говорить, что движок находится на дуге, если это последняя дуга, которую он прошёл. Тогда: 1) движок через $O(1)$ тактов уходит с дуги или останавливается, если из конца этой дуги недостижима серая вершина, 2) на каждой дуге одновременно находится не более 1 движка. Поэтому, пока есть серые вершины, генератор может генерировать один движок каждые $O(1)$ тактов без нарушения этих двух условий. Тем самым, через $O(m)$ тактов обход закончится (мы указываем лишь порядок времени обхода, чтобы учесть затраты на передачу сообщений и срабатывание отказов, когда дуги с указанным номером нет).

Если за 1 такт генератор может сгенерировать неограниченное, но конечное, число движков (*алгоритм 2*), то оценка меняется. Если автоматы неизбыточные, то движок, «узнавая» полустепень исхода вершины, сообщает её генератору. Тем самым, генератор «знает» число непройденных дуг, выходящих из каждой пройденной вершины. На каждом такте генератор генерирует число движков, равное разности между числом непройденных дуг, выходящих из пройденных вершин, и числом уже сгенерированных движков. Рассмотрим путь длины L, выходящий из корня. Если i-ая вершина пути достигается на такте t_i, то на этом такте генератор может сгенерировать столько движков, сколько дуг выходит из i-ой вершины минус 1 (один движок уже в вершине). Значит, вершина $i+1$ будет достигнута не более чем через i тактов

после этого: $t_{i+1} \le t_i+i$. Поскольку $t_1=0$, имеем $t_L \le L(L-1)/2$. Т.к. $L \le d$, время обхода $\boldsymbol{O}(d^2)$.

Покажем, что алгоритм не может работать быстрее. Рассмотрим граф, состоящий из пути длины d, вершины которого пронумерованы от корня $1,2,3\ldots d$, и из каждой вершины i выходят m_i дуг, причём все они, кроме дуги пути $i \to i+1$, терминальные. Пусть нумерация выходящих дуг такая, что сначала проходятся терминальные дуги, выходящие из вершины, а потом уже дуга пути. Пусть первый движок достигает вершины i за t_i тактов. Тогда число сгенерированных за это время движков $\boldsymbol{f}(t_i)$ не зависит от m_i. Поэтому всегда можно подобрать такой граф, чтобы $\boldsymbol{f}(t_i) < m_i$. Следовательно, на такте t_i «не хватает» движков, чтобы пройти все m_i дуг, выходящих из вершины i. Поэтому движок, который пройдёт по дуге $i \to i+1$, а эта дуга проходится последней среди всех дуг, выходящих из вершины i, будет сгенерирован после такта t_i. Для того чтобы этому движку пройти от корня до вершины $i+1$, потребуется не менее i тактов. Следовательно, $t_{i+1} \ge t_i+i$. Имеем: $t_1=0$, $t_2 \ge 1$, $t_3 \ge 3$, $t_4 \ge 5,\ldots$, $t_d \ge d(d-1)/2$. Тем самым, время обхода $\boldsymbol{\Omega}(d^2)$ при достаточно большом m.

Если совместить алгоритмы 1 и 2, генерируя движки первого и второго рода, то получится алгоритм с оценкой $\boldsymbol{O}(\min\{d^2,m\})$. Необходимость в таком совмещении показывается примером графа, состоящего из пути длины d, в каждой вершине которого, кроме последней, добавлена терминальная дуга. Алгоритмы 1 и 2 обходят этот граф за время порядка $\boldsymbol{O}(m)=\boldsymbol{O}(d)$ и $\boldsymbol{O}(d^2)$, соответственно.

Если автоматы свободные, то время обхода неограниченно растёт вместе с ростом числа дуг m даже при фиксированном диаметре d. Действительно, в противном случае для каждого алгоритма A максимальное время обхода на классе графов диаметром d конечно: $t(A,d)<\infty$. Обход завершается тогда, когда какой-то движок получает последний отказ, т.е. получен отказ для каждой пройденной вершины. (После этого достаточно одного такта, чтобы сообщение

об отказе дошло до генератора, который определяет, что это был последний отказ, и сообщает «вовне» о завершении обхода). Пусть время *t(A,d)* достигается на графе *G* с числом дуг *m*. Тогда для любого *m`>t(A,d)* можно рассмотреть граф *G`* с тем же диаметром *d* и числом дуг *m`*, который отличается от графа *G* тем, что в той вершине графа *G*, где получается последний отказ, добавляется *m`-m* петель. Время обхода графа *G`*, очевидно, больше *t(A,d)*, что противоречит его максимальности. Утверждение доказано. Итак, время обхода свободными автоматами по порядку не меньше d^2 при достаточно большом числе дуг *m* и неограниченно растёт вместе с ростом *m*.

4. Практический пример и проблема больших графов

Практическим примером обхода ориентированного графа коллективом двигающихся автоматов может служить работа [2], посвящённая тестированию модели цифровой аппаратуры с помощью сети компьютеров. В графе переходов модели 84561 состояние и 338244 дуги. Число компьютеров менялось от 1 до 100 (Intel Core2 Quad Q9400, 2.66 GHz; 4 Gb; Linux; Ethernet). В применявшемся DFS-алгоритме не было генератора, движки статически распределялись по компьютерам (один в один) и каждый из них общался только с соседями в соответствии с топологией связей. Такой распределённый обход графа преследовал цель уменьшения времени обхода. В Таб. 1 показано время тестирования в зависимости от числа компьютеров и топологии связей.

Таб. 1. Тестирование модели цифровой аппаратуры.

Число компьютеров *k*	Топология связей	Время прогона теста, мин *T(k)*	Ускорение *T(1) / T(k)*	Коэффициент эффективности *T(1) / kT(k)*
1	—	803.3	1	1
81	Кольцо	12.2	66	0.81
81	Тор 9×9	11.4	70	0.87
100	Кольцо	10.2	79	0.79
100	Тор 10×10	9.5	85	0.85

В этом практическом примере граф, хотя и большой, всё же помещался в память одного компьютера, т.е. в память автомата. А что делать, если граф не помещается в память автомата? Когда выше мы рассматривали обход графа коллективом свободных неограниченных автоматов, выяснилось, что память вершин автоматам, в общем-то, не нужна, достаточно, чтобы граф был нумерованным.

Сохраняя эти предположения, будем считать, что, если граф не помещается в память одного автомата, то есть этот автомат полуробот, то граф всё же должен помещаться в суммарную память нескольких полуроботов, число которых неограниченно. Это важно при тестировании, где никакой памяти вершин нет, поскольку вершина – это состояние тестируемой системы. Его можно наблюдать, но в него нельзя писать, можно только перейти в другое состояние в результате тестового воздействия.

5. Обход неизвестного графа коллективом полуроботов

Алгоритм обхода опубликован в [3], здесь мы ограничимся изложением его основной идеи. Кроме генератора и движков добавляется новый тип автомата – регулятор вершины, в суммарной памяти регуляторов как раз и хранится описание пройденного графа. Все автоматы свободные. Алгоритм строит прямое дерево пройденного графа, помечая прямые дуги и хорды в их начальных вершинах. Регулятор вершины не взаимодействует с графом и хранит информацию только о ближайшем окружении вершины: номер вершины, адрес *родителя*, то есть регулятора начала входящей прямой дуги, и для каждой выходящей прямой дуги – адрес *потомка*, то есть регулятора конца дуги. Регулятор управляет перемещением движков через свою вершину.

Откуда берутся регуляторы? Функции регулятора корня выполняет генератор. Для другой вершины регулятор создаёт движок, первым попавший в вершину. Как движок узнаёт, что он первый? Для этого выполняется поиск регулятора по номеру вершины. Регуляторы связаны в список по их адресам, и

по списку посылается сообщение с номером вершины и адресом движка. Если регулятор не найден, то движок в эту вершину попал первым.

Как регулятор управляет движением движков? Движок «спрашивает» (с помощью обмена сообщениями) у регулятора текущей вершины, ***куда идти***? Регулятор отвечает: ***иди по дуге*** номер такой-то. Дуги перебираются в порядке возрастания номеров: 1,2,3, и так далее. Если такой дуги не оказалось, движок сообщает об этом регулятору и тот теперь знает число выходящих дуг. Далее регулятор перебирает номера по циклу. Этого достаточно, чтобы обойти граф.

Как узнать о конце обхода? Как и BFS-алгоритме для одного автомата здесь имеется прямое дерево и его серое поддерево. Отличие лишь в том, что один автомат сам должен после хорды пытаться строить новую ветвь дерева, а когда автоматов много, то эти ветви создаются ими параллельно. Вместо отката используется сообщение ***конец***, посылаемое по дуге в обратном направлении, то есть из автомата в конце дуги регулятору начала дуги. При откате по хорде отправитель – движок, прошедший по хорде, а при откате по прямой дуге отправитель – регулятор конца дуги. Регулятор-получатель отмечает дугу как *законченную*. Законченная дуга, по сути, то же, что чёрная дуга. Как движок узнаёт, что он прошёл по хорде? Это происходит, когда движок не первым пришёл в вершину, т.е. при опросе регуляторов регулятор найден. Сам регулятор посылает ***конец*** родителю, когда все выходящие дуги стали законченными. Если это регулятор корня, то конец обхода.

При такой работе, если не принять никаких дополнительных мер, то время обхода экспоненциально. Это видно из примера на Рис. 3. Дуга вниз имеет номер 1, а дуга направо номер 2. Над дугой написан номер движка в порядке генерации, который первым проходит эту дугу. Из этого примера видно, как «вредно» ходить по законченным дугам.

И. Бурдонов, А. Косачев.
Исследование графов коллективом двигающихся автоматов.
Программная инженерия. т.7, №12, 2016 , стр. 559-567.
9 стр.

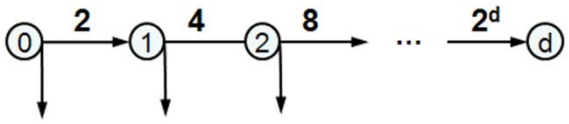

Рис. 3. Пример экспоненциального времени обхода.

Как уменьшить время обхода? Для этого регулятору не надо посылать движки по законченным дугам, такие дуги блокируются до конца обхода.

Как уменьшить число одновременно существующих движков? Для этого нужно уничтожать движок, если он: a) прошёл по хорде, или b) прошёл по прямой дуге, а дальше ему некуда идти, потому что все выходящие дуги оказались законченными. Тогда каждый движок проходит путь длиной $O(n)$, после чего выполняет опрос за время $O(n)$ и становится регулятором или уничтожается. Значит, движок живёт $O(n)$ тактов. Если за такт генерируется не более одного движка, число одновременно существующих движков $O(n)$.

Как уменьшить число генерируемых движков? Если не тормозить генерацию движков, то их будет сгенерировано столько, сколько длится обход. Для торможения реализуется старт-стопный механизм перемещения движков по прямой дуге. Регулятор направляет движок по прямой дуге только после получения от регулятора конца дуги сообщения ***запрос*** движка. Дуга временно блокируется до получения следующего ***запроса***, который будет отправлен, когда движок пойдёт дальше. Если все выходящие дуги заблокированы, то приходящий движок либо уничтожается, если все дуги закончены, либо ожидает разблокировки. Запрос не посылается, поэтому в вершине не более одного ждущего движка. Если все выходящие из корня дуги блокированные, генератор приостанавливает генерацию движков. Общее число движков $O(m)$.

Какова оценка времени обхода? Для оценки времени обхода будем считать, что проход по дуге и пересылка сообщения выполняются за 1 такт. Также за 1 такт генерируется не более 1 движка. Адрес автомата – это его номер. Время обхода равно $O(m+nd)$. Второе слагаемое возникает из-за опроса регуляторов:

время одного опроса не больше числа регуляторов, которое не больше числа вершин *n*, а на прямом пути от корня до листа длиной не более *d* опросы регуляторов происходят строго последовательно в порядке от корня к листу.

Таб. 2. Размеры сообщений и памяти автомата.

Преобразование вершины в цепочку вершин: в терминах исходного графа вместо *n* читать *m*.					
Полустепень исхода ограничена?		нет	нет	да	да
Адрес автомата используется повторно?		нет	да	нет	да
Размер сообщения	неограниченный	***O**(log m)*		***O**(log n)*	
	полуробот	***O**(log m)*		***O**(log n)*	
Размер (памяти) автомата	неограниченный	***O**(m log n)*		***O**(n log n)*	
	полуробот	***O**(m log m)*	***O**(m log n)*	***O**(log n)*	

В Таб. 2 приведены размеры сообщения и памяти полуробота, а также, для сравнения, те же размеры для неограниченных автоматов. Мы видим, что размер сообщения остаётся тем же, там два основных параметра: адрес автомата и номер дуги. Если полустепень исхода вершины не ограничена, то выигрыша по памяти автомата не получается. Но этого и следовало ожидать, поскольку в этом случае локализация информации о графе не может дать выигрыша: полустепень исхода может достигать числа дуг в графе. Если адрес автомата не используется повторно, то получается даже проигрыш, поскольку число автоматов тоже может достигать числа дуг в графе. Для ограниченной полустепени исхода число дуг столько же по порядку, сколько вершин. Получается выигрыш в *n* раз. Если граф получен с помощью преобразования вершины в цепочку вершин (Рис. 1), когда получается полустепень исхода 2, число вершин преобразованного графа по порядку равно числу дуг исходного графа. И получается выигрыш в *m* раз. Здесь мы имеем пример классического правила – «если хотите сэкономить память, придётся пожертвовать временем»: ***O**(m+nd)* вместо ***O**(m)*.

6. Роботы

Минимальная длина обхода (маршрута, проходящего по всем дугам графа) ориентированного графа с n вершинами и m дугами равна $\boldsymbol{O(nm)}$, или, в терминах диаметра графа d как максимальной длины пути (маршрута без самопересечения) в графе, $\boldsymbol{O(dm)}$. В [1] рассматривались алгоритмы обхода ориентированного графа одним роботом. Наилучшая известная оценка длины обхода $\boldsymbol{O(nm+n^2 loglogn)}$, где n – число вершин, m – число дуг, или, в терминах диаметра графа d как максимальной длины пути (маршрута без самопересечения) в графе, $\boldsymbol{O(dm+dnloglogn)}$.

Второе слагаемое возникает из-за проблемы отката, то есть возврата из конца дуги в её начало (откат по хорде и откат по дереву), для чего автомат должен пройти некоторый маршрут. Этот маршрут существует в пройденном графе и состоит из пути по обратному дереву до серого дерева и пути по серому дереву до начала дуги. Вместе с этой дугой маршрут образует цикл, по которому автомат и может двигаться. Проблема в том, что робот не «знает», в какую вершину ему надо вернуться, поскольку начало дуги специально не помечено. Поэтому он вынужден двигаться по циклу несколько раз (подробнее см. в [1]).

Проблема отката легко решается, если используются два робота. Длина обхода уменьшается до минимальной по порядку $\boldsymbol{O(nm)}$ или $\boldsymbol{O(dm)}$. Идея алгоритма в том, что роботы идут по графу синхронно, но с расстоянием в одну дугу. Поэтому, когда первый робот «узнаёт», что нужно делать откат, т.е. возвращаться на одну дугу назад, в начале этой дуги как раз стоит второй робот. Достаточно сообщить ему, чтобы он пометил вершину, в которой находится, и вместе идти по циклу до помеченной вершины. При откате по хорде первый робот находится в конце хорды, а второй – в начале хорды. При откате по дереву первый робот находится в листовой вершине серого дерева, а второй – в предыдущей вершине дерева.

Темой дальнейших исследований могли бы стать алгоритмы обхода графа большим числом роботов. Для коллектива неограниченного числа полуроботов или неограниченных автоматов, как показано выше, оценки времени обхода существенно лучше, чем в случае одного автомата как в первой статье [1]. Можно было бы ожидать, что аналогичный результат может быть получен и для роботов, число которых больше двух. Правда, здесь возникает проблема обмена сообщениями между автоматами, поскольку требуется указывать адрес автомата-получателя, который тем самым становится частью выходного символа автомата-отправителя, а размер выходного символа должен быть ограничен для конечного автомата, т.е. робота. Поэтому либо число роботов должно быть ограничено сверху, что может не дать ожидаемого эффекта, либо нужно найти какой-то другой способ адресации автоматов, например, опираясь на топологию сети связи автоматов (тор, кольцо и т.д. как в разделе 4), использовать локальный адрес автоматов.

7. Недетерминированный граф

7.1. Справедливый недетерминизм и предположение о Δ-дугах

Упорядоченный граф будем называть *недетерминированным*, если одним номером дуги может быть помечена не одна, а несколько выходящих дуг. *Δ-дугой* будем называть множество дуг с одним номером и одной начальной вершиной. Когда автомат в вершине указывает номер выходящей дуги, недетерминированным образом выбирается одна из дуг соответствующей Δ-дуги. Будем считать, что этот недетерминированный выбор задаётся недетерминированной функцией выбора, которая для каждой Δ-дуги возвращает дугу из этой Δ-дуги.

Понятно, что если на функцию выбора не налагается никаких ограничений, то, вообще говоря, обход графа невозможен: может оказаться, что при любом вызове функции выбора одна из дуг Δ-дуги никогда не выбирается. Будем говорить, что недетерминизм *справедливый* [5], если для любой Δ-дуги в любой

бесконечной последовательности значений функции выбора каждая дуга из этой Δ-дуги встречается бесконечное число раз. Заметим, что дуга должна встречаться бесконечное число раз, а не хотя бы один раз, поскольку в противном случае возможность обхода графа не гарантируется. Примером может служить граф на Рис. 3 при $d>1$, если считать, что две дуги, выходящие из одной вершины, образуют одну Δ-дугу, функция выбора для каждой Δ-дуги только один раз возвращает дугу, ведущую направо, а автоматы генерируются в корне – вершине 0.

Если граф – это граф переходов модели тестируемой системы, то гипотеза о справедливом недетерминизме эквивалентна гипотезе о глобальном тестировании: при бесконечном числе прогонов теста (включая рестарт системы) бесконечное число раз будут получены все возможные варианты поведения тестируемой системы.

Гипотеза о справедливом недетерминизме гарантирует возможность обхода любого нумерованного графа неограниченным числом свободных полуроботов. Для этого достаточно, чтобы генератор в корне постоянно генерировал движки, а в каждой вершине (идентифицируемой по её номеру) движки перебирали номер Δ-дуги по циклу (число Δ-дуг в вершине свободный автомат определяет, получая отказ).

Но как узнать о конце обхода? Для этого достаточно следующего *предположения о Δ-дугах*: 1) в Δ-дуге нет кратных дуг, 2) для каждой Δ-дуги можно узнать число дуг в ней. Первое нужно для того, чтобы можно было различить дуги одной Δ-дуги по их конечным вершинам. Второе нужно для того, чтобы узнать, все ли дуги Δ-дуги пройдены. Мы будем считать, что когда движок пытается пройти по дуге с указанным номером, он либо получает отказ, либо проходит по дуге и получает число дуг в соответствующей Δ-дуге.

И. Бурдонов, А. Косачев.
Исследование графов коллективом двигающихся автоматов.
Программная инженерия. т.7, №12, 2016 , стр. 559-567.
9 стр.

7.2. Обход недетерминированного графа коллективом полуроботов

Алгоритм обхода справедливо недетерминированного ориентированного графа коллективом двигающихся полуроботов опубликован в [5], здесь мы ограничимся изложением его основной идеи. Попробуем приспособить к недетерминированному графу алгоритм обхода детерминированного графа с помощью движков и регуляторов (5).

Первое решение, которое приходит на ум, простое: пусть движок, проходя по дуге, сообщит, в какую вершину он попал, регулятору начала дуги. Регулятор проверяет, туда ли он направлял движок: если не туда, движок уничтожается (или останавливается), а если туда, ему разрешают продолжить работу. Однако это решение неудачное. Например, Δ-дуга состоит из двух незаблокированных прямых дуг: *a* и *b*. Регулятор посылает первый движок по дуге *a*, а второй – по дуге *b*. Но из-за недетерминизма всё получается наоборот: первый движок проходит дугу *b*, а второй – дугу *a*. Регулятор убивает оба движка. А на следующем шаге всё повторяется. Недетерминизм справедливый: «честно» перебираются дуги в Δ-дуге: *a*, *b*, *a*, *b* и так далее. Но регулятор убивает все движки.

Второе решение подсказывает этот же пример. На самом деле всё равно, какую дугу пройдёт движок *a* или *b*: они обе незаблокированные. Так что регулятор должен вести себя просто «умнее»: если движок прошёл одну из незаблокированных дуг, то всё в порядке. Только при проходе заблокированной дуги движок уничтожается. Это решение правильное, но неэффективное. Дело в том, что движку приходится опрашивать регуляторы после прохода каждой дуги.

Третье решение использует приём, аналогичный тому, что использовался в детерминированном случае. Там движок не опрашивал регуляторы, если проходил по прямой дуге, так как регулятор её начала уже знал регулятор конца дуги. Однако в недетерминированном случае движок может пройти по

другой дуге. С другой стороны, движок всегда знает номер вершины, в которую попал. Если регулятор для каждой прямой выходящей дуги хранит не только регулятор её конца, но и номер вершины, он может «подсказать» движку по номеру вершины адрес её регулятора. Это делается обменом сообщениями между движком и регулятором. В этом случае поиск регулятора выполняется, как и в детерминированном случае, только после прохода ещё непройденной дуги.

Как оценить такой алгоритм? Размер сообщений и памяти автомата увеличиваются по сравнению с детерминированным случаем, но имеют тот же порядок. А время обхода, конечно, зависит от недетерминированной функции выбора. Если она на самом деле детерминирована, то получаем ту же оценку $\boldsymbol{O}(m+nD)$.

Пример другой функции справедливого выбора – это t-недетерминизм [4,5], который для фиксированного числа t гарантирует, что за t попыток пройти по Δ-дуге будут пройдены все её дуги. Заметим, что для t-недетерминированного графа нам не нужно предположение о Δ-дугах (из 7.1). Действительно, если автоматы t раз выполнят проход по Δ-дуге, каждый раз запоминая (в регуляторе начала дуги) дугу, по которой они проходят, то, во-первых, гарантированно будут пройдены все дуги Δ-дуги, включая кратные дуги, а, во-вторых, регулятор «узнает» число дуг в Δ-дуге с точностью до кратности, т.е. «узнает» число различных концов дуг этой Δ-дуги. Алгоритм обхода неизвестного ориентированного t-недетерминированного графа одним неограниченным автоматом предложен в [4]. Оценка времени обхода экспоненциальная. Для коллектива полуроботов алгоритм предложен в [5], но тоже, по крайней мере, с экспоненциальной оценкой. Примером может служить граф на Рис. 3, если все дуги, выходящие из одной вершины, принадлежат одной Δ-дуге, а «зловредная» функция выбора только каждый 2-ой раз направляет движок по дуге направо.

Для практического применения желательна полиномиальная оценка, для чего нужно использовать какие-то дополнительные ограничения. Например, в *t*-недетерминированном графе есть *k* детерминированных дуг (число дуг в Δ-дуге равно 1), и по ним можно попасть из корня в любую вершину. Будем считать, что после прохода недетерминированной дуги движок уничтожается, как после прохода хорды в детерминированном случае. Алгоритм состоит из двух этапов. На первом этапе будем проходить недетерминированные Δ-дуги по одному разу, что эквивалентно обходу детерминированного графа, который получается из исходного заменой каждой недетерминированной Δ-дуги одной детерминированной терминальной дугой. Строится прямое дерево из детерминированных дуг. Время обхода $O(p+nd)$, где p – число Δ-дуг. На втором этапе не будем проходить детерминированные хорды, а каждую недетерминированную Δ-дугу будем проходить «оставшиеся» $t-1$ раз, что эквивалентно обходу детерминированного графа, который получается из исходного удалением детерминированных хорд (остаётся $n-1$ детерминированная дуга) и заменой каждой недетерминированной Δ-дуги $t-1$ детерминированными терминальными дугами. Время обхода $O((p-k)(t-1)+n-1+nd)$. Суммарное время $O(pt+nd)$.

7.3. Δ-обход

Другой подход к обходу недетерминированного графа – это модификация самой цели обхода: вместо того, чтобы проходить по всем дугам, требуется пройти по всем Δ-дугам, то есть хотя бы по одной дуге в каждой Δ-дуге [6]. Это имеет практический смысл при тестировании: в каждом состоянии системы мы пробуем подать на неё каждое тестовое воздействие хотя бы по одному разу.

Будем называть *Δ-маршрутом* множество маршрутов с одной начальной вершиной, которое «ветвится» по всем дугам каждой проходимой Δ-дуги. *Δ-обход* – это Δ-маршрут, проходящий по всем Δ-дугам. Если при движении по

графу автомат называет номер следующей Δ-дуги в соответствии с Δ-обходом, то он гарантированно пройдёт какой-то маршрут из Δ-обхода, то есть пройдёт по всем Δ-дугам. Граф называется *сильно Δ-связным*, если для каждой пары вершин *a* и *b* существует Δ-маршрут, все маршруты которого начинаются в *a* и заканчиваются в *b*.

Для существования Δ-обхода, начиная с любой начальной вершины, необходима и достаточна сильно Δ-связность. При фиксированной начальной вершине сильно Δ-связность только достаточна, а необходимое и достаточное условие использует соответствующим образом определённые недетерминированные графы 1-го и 2-го рода [6]. Граф 1-го рода (Рис. 4 вверху) – это цепочка компонентов сильной Δ-связности, где для каждого *i*-го компонента, кроме последнего, все дуги, выходящие из него и ведущие *вперед* (в компоненты с большим номером), образуют одну связующую Δ-дугу, ведущую в следующий *i+1*-ый компонент, а корень графа лежит в 1-ом компоненте. Заметим, что некоторые дуги (но не Δ-дуги!) могут вести *назад*, в компоненты с меньшими номерами. Δ-обход существует тогда и только тогда, когда граф 1-го рода. Однако, если граф неизвестен, то автомат может гарантированно выполнить Δ-обход только тогда, когда каждый компонент, кроме последнего, состоит из одной вершины, из которой выходит только одна дуга, ведущая в следующий компонент. Такой граф называется графом 2-го рода (Рис. 4 внизу). Для детерминированного графа эти определения совпадают с соответствующими определениями в подразделе 4.1 в [1].

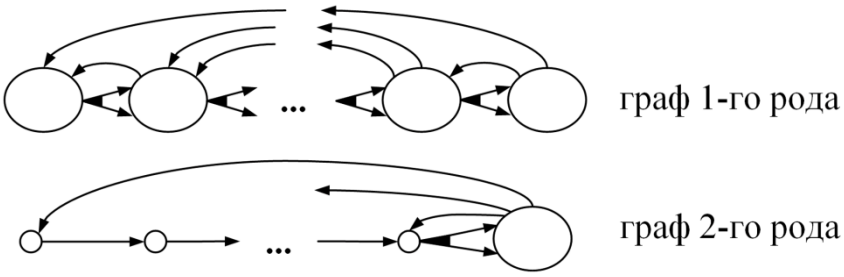

Рис. 4. Графы 1-го и 2-го рода для Δ-обхода.

В [6] опубликован алгоритм Δ-обхода неизвестного сильно Δ-связного нумерованного графа. Длина обхода $O(np)$, где n – число вершин, p – число Δ-дуг. Память вершины и автомата $O(s\log n)$, где s – ограничение сверху на число Δ-дуг, выходящих из одной вершины. Однако этот алгоритм имеет одну особенность: автомат, пройдя по дуге и прочитав информацию из конечной вершины дуги, может выполнить запись в начальную вершину дуги. Эту работу может моделировать неограниченный автомат, если для каждой вершины (по её номеру) он хранит информацию, связанную с этой вершиной, в своей памяти.

Другой вариант: два полуробота, обменивающиеся между собой сообщениями. Аналогично разделу 6, автоматы двигаются синхронно с отставанием на одну дугу. Тогда первый автомат, пройдя дугу, посылает сообщение, содержащее информацию из конца дуги, второму автомату, который в это время находится в начале дуги и может сделать запись в неё.

Можно также использовать идею о регуляторах и движках из раздела 5. Имеется один движок, который создаёт регулятор вершины, когда впервые приходит в эту вершину. Если движок, пройдя дугу, попадает в вершину, где уже есть регулятор, он «узнаёт» у этого регулятора хранящуюся в нём информацию (она относится к концу дуги), и посылает сообщение, содержащее эту информацию, регулятору начала дуги, который делает запись в свою память. Проблема лишь в том, что движок должен делать опросы регуляторов. Как и в разделе 5 опрос можно не делать в том случае, когда движок проходит уже пройденную ранее дугу, если при первом проходе дуги в регуляторе начала дуги запоминать пару <номер конца дуги, адрес регулятора конца дуги>. Но даже в этом случае время Δ-обхода увеличится до $O(nm)$, поскольку всё равно нужно опрашивать регуляторы после первого прохода дуги (число дуг m), а каждый опрос требует времени $O(n)$. Темой дальнейших исследований могла

бы стать адаптация алгоритма из [6] для неограниченного числа движков, генерируемых в корне графа, как в разделе 5.

8. Заключение

Эта статья вместе с предыдущей статьей [1] завершает рассмотрение модели исследования графа автоматами, двигающимися по графу. Оценки, приведённые в данной статье для коллектива автоматов, позволяют сделать вывод о том, что параллельный обхода графа несколькими автоматами существенно уменьшает время работы. Нерешёнными остаются задачи обхода графа коллективом большого числа роботов и Δ-обхода недетерминированного графа коллективом большого числа полуроботов. Также представляет интерес поиск таких функций справедливого выбора и/или ограничений на недетерминированные графы, которые позволяют получать полиномиальные оценки времени обхода. В данной статье приведён только один такой пример: недетерминированный граф с несколькими детерминированными дугами, по которым достижимы все вершины графа.

В последней, третьей, статье серии будет рассмотрена «инвертированная» модель, в которой автоматы неподвижно «сидят» в вершинах графа, но зато обмениваются между собой сообщениями не по независимой от графа сети связи, а по дугам графа.

Литература

1. И.Б.Бурдонов, А.С.Косачев. Исследование графа автоматом. // Программная инженерия. — год. — Т.???, №???. — С. ???-???.
2. И.Б.Бурдонов, С.Г.Грошев, А.В.Демаков, А.С.Камкин, А.С.Косачев, А.А.Сортов. Параллельное тестирование больших автоматных моделей. // Вестник ННГУ — 2011. — №3. — С. 187-193.
3. И.Б.Бурдонов, А.С.Косачев. Обход неизвестного графа коллективом автоматов. // Труды Института системного программирования РАН. — 2014. — Т. 26, №2. — С. 43-86.

4. И.Б.Бурдонов, А.С.Косачев. Полное тестирование с открытым состоянием ограниченно недетерминированных систем. // Программирование. — 2009. — №6. — С. 3-18.
5. И.Б.Бурдонов, А.С.Косачев. Обход неизвестного графа коллективом автоматов. Недетерминированный случай. // Труды Института системного программирования РАН. — 2015. — Т. 27, №1. — С. 51-68.
6. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Неизбыточные алгоритмы обхода ориентированных графов. Недетерминированный случай. // Программирование. — 2004. — №1. — С. 2-17.