И. Бурдонов, А. Косачев.
Исследование ориентированного графа коллективом неподвижных автоматов.
Программная инженерия. т.8, №1, 2017, стр. 16-25.
10 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 ориентированного графа коллективом неподвижных автоматов

Эта статья – третья в серии статей, посвящённых исследованию графов автоматами. Если в первых двух статьях автоматы двигались по дугам графа и, для коллектива автоматов, обменивались между собой сообщениями по независимой от графа сети связи, то в данной статье автоматы не двигаются, находятся в вершинах графа и обмениваются сообщениями, пересылаемыми по дугам графа. Кроме исследования графа, рассматривается также задача параллельных вычислений на графе, в том числе динамически меняющемся.

Ключевые слова: ориентированный граф, упорядоченный граф, нумерованный граф, динамический граф, исследование графа, параллельные вычисления, автомат, робот, полуробот.

1. Введение

Эта статья является последней в серии из трёх статей по исследованию графов автоматами. Граф предполагается ориентированным. В первых двух статьях [1,2] рассматривалась модель аналогичная машине Тьюринга, в которой лента заменена графом: автоматы двигаются по дугам графа, а вершины графа используются как ячейки памяти для чтения/записи.

Если автомат один [1], то эта модель эквивалентна «инвертированной» модели, в которой автоматы неподвижно «сидят» в вершинах графа, а по дугам пересылается сообщение от автомата к автомату. Одному автомату соответствует одно сообщение, но оно меняется при прохождении через вершину, т.е. меняется автоматом, находящимся в этой вершине. В случае коллектива двигающихся автоматов [2] использовалась дополнительная сеть

связи, по которой двигающиеся автоматы могли пересылать сообщения друг другу. В данной статье мы рассматриваем «инвертированную» модель со многими автоматами и многими сообщениями. В каждой вершине находится «неподвижный» автомат, а по дугам перемещается множество сообщений. Это, в каком-то смысле, более «суровая» постановка задачи по сравнению с [2], поскольку нет никакой независимой от графа сети связи автоматов, позволяющей пересылать сообщение от автомата любому другому автомату по его адресу. Сам исследуемый граф и есть такая сеть связи, и автомат в вершине может послать сообщение только «ближайшим» автоматам, а именно, автоматам, находящимся на концах дуг, выходящих из этой вершины.

Как в [1,2] *упорядоченным* графом будем называть граф, в котором дуги, выходящие из вершины, перенумерованы, начиная с 1, – для каждой вершины задана своя нумерация выходящих дуг. В [1,2] автомат указывал номер выходящей дуги, чтобы по ней пройти, здесь же номер дуги указывается, чтобы послать по ней сообщение. Также будем считать, что в графе выделена начальная вершина, которую будем для краткости называть *корнем* графа. Вначале все автоматы находятся в своих начальных состояниях, а на дугах нет сообщений. Если для исследования графа требуется «толчок» извне, то он реализуется как сообщение, которое извне поступает в автомат в корне.

В [1,2] выделялись три типа автомата в зависимости от размера их памяти. Автомат, который конечен на всём рассматриваемом классе графов, назывался *роботом*. Если автомат не конечный, но граф не помещается в его память, то такой автомат назывался *полуроботом*. По сути, это то же самое, что локальный алгоритм на графе. А если граф помещается в память автомата, то это *неограниченный автомат*. В данной статье все автоматы будут считаться полуроботами.

Мы рассмотрим два типа графа и две задачи, решаемые коллективом автоматов. Граф *статический*, если он не меняется в процессе работы

автоматов, и *динамический*, если может изменяться: его дуги могут исчезать, появляться и менять свой конец. Первая задача – исследование графа с целью узнать, как он устроен. Вторая задача – параллельные вычисления на графе.

2. Неподвижные автоматы на статическом графе

Пусть имеется ориентированный упорядоченный граф, в каждой вершине которого находится автомат. Автомат «знает» только полустепень исхода вершины. За одно срабатывание автомат принимает сразу все поступившие к нему сообщения по всем входящим дугам и может послать сразу несколько сообщений по выходящим дугам с указанием номеров этих дуг.

Дуга – это буфер сообщений ёмкостью k, то есть на ней может одновременно находиться не более k сообщений. Если на ней уже k сообщений, ещё не дошедших до автомата конца дуги, то автомат начала дуги не может посылать по этой дуге сообщения.

Говоря о ёмкости дуги, мы предполагаем, что размер сообщения ограничен сверху. Очевидно, что посылка k сообщений максимального размера эквивалентна посылке одного большого сообщения, максимальный размер которого в k раз больше. Отличие лишь в том, что если на дуге меньше k маленьких сообщений, то на неё можно добавлять сообщения. Однако мы будем рассматривать алгоритм, для которого оценка времени работы не меняется по порядку, если такого добавления не делать и не посылать по дуге сообщения, пока она не освободится, а потом сразу послать несколько сообщений, но, конечно, не больше k.

Одни сообщения посылаются из автомата вершины сразу по всем выходящим дугам, другие – только по части выходящих дуг, а третьи – только по одной выходящей дуге. Тем не менее, для простоты мы будем считать, что сообщение ожидает в вершине освобождения сразу всех выходящих дуг. Понятно, что время работы алгоритма от этого не уменьшится. Таким образом,

для автомата вершины не нужны сигналы об освобождении каждой выходящей дуги, а достаточно общего сигнала об освобождении всех выходящих дуг.

Для оценки времени работы алгоритма мы будем пренебрегать временем срабатывания автомата и считать, что сообщение находится на дуге не более 1 такта, т.е. не более чем через 1 такт после посылки сообщения из автомата начала дуги оно будет принято автоматом конца дуги.

3. Разметка статического графа

Задачу исследования статического графа сформулируем так: когда извне в автомат корня придёт сообщение, дающее старт работе автоматов, нужно выполнить *разметку* графа и сообщить об этом в ответном сообщении. Такая разметка означает, что автоматы в вершинах графа приведены в нужные состояния, т.е. в памяти каждого автомата хранится некоторая локальная информация о графе, а совокупная память автоматов описывает граф и его структуру.

Разметка графа включает в себя:

1) *Прямой остов* графа, ориентированный от корня. В памяти автомата каждой вершины отмечаются все выходящие *прямые* дуги, т.е. дуги прямого остова. Остальные выходящие дуги – *хорды* прямого остова.

2) *Обратный остов* графа, ориентированный к корню. В памяти автомата каждой вершины, кроме корня, хранится номер выходящей обратной дуги, т.е. дуги обратного остова. Из вершины может выходить не более одной обратной дуги.

3) В памяти автомата каждой вершины запоминается число входящих обратных дуг.

Такую разметку графа можно понимать как результат исследования графа. Когда разметка сделана, гарантируется, что по каждой дуге передано хотя бы одно сообщение, т.е. совершён обход графа. Кроме того, такая разметка используется в дальнейшем для параллельных вычислений на графе.

И. Бурдонов, А. Косачев.
Исследование ориентированного графа коллективом неподвижных автоматов.
Программная инженерия. т.8, №1, 2017, стр. 16-25.
10 стр.

Алгоритм разметки графа подробно описан в [3,4]. Здесь мы дадим только идею этого алгоритма и приведём оценки времени его работы без доказательств. Условно разобьём алгоритм разметки на 3 части: 1) Построение прямого и обратного остовов. 2) Определение конца построения. 3) Установка счётчиков числа входящих обратных дуг.

3.1. Построение прямого и обратного остовов

Построение остовов использует 4 типа сообщений: 1) ***Старт*** – идентифицирует вершины графа, 2) ***Поиск*** – ищет пути от вершин к корню, 3) ***Прямое*** – размечает прямой остов, 4) ***Обратное*** – размечает обратный остов.

Автомат корня получает сообщение ***Старт*** извне, с этого начинается работа по разметке графа. Далее сообщение ***Старт*** рассылается «веером»: получив первый раз это сообщение, автомат вершины рассылает его по всем выходящим дугам. Повторно приходящие сообщения ***Старт*** игнорируются. Это сообщение накапливает в себе *вектор маршрута*, который оно проходит, как список номеров дуг маршрута. Идентификатором вершины становится *вектор вершины*, который определяется как вектор маршрута из первого полученного автоматом вершины сообщения ***Старт***. Вектор корня пустой.

Сообщение ***Поиск*** ищет путь от вершины к корню. Автомат вершины, получив ***Старт***, инициирует сообщение ***Поиск***. Это сообщение содержит вектор инициатора и накапливает в себе *обратный вектор* как вектор маршрута, который проходит. Сообщение ***Поиск*** также рассылается «веером». Повторно приходящие сообщения ***Поиск*** от того же инициатора игнорируются. Но инициаторов много – это все вершины. Поэтому, чтобы игнорировать повторные сообщения от тех же инициаторов, автомат вершины хранит в своей памяти множество векторов инициаторов из проходивших через него сообщений ***Поиск***. Автомат корня, приняв сообщение ***Поиск***, получает вектор инициатора, то есть вектор прямого пути от корня до инициатора, и обратный

вектор пути от инициатора до корня. Автомат корня создаёт сообщение ***Прямое***, переписывая в него вектор инициатора и обратный вектор.

Сообщение ***Прямое*** размечает прямой путь от корня до инициатора. Это сообщение двигается по прямому пути от корня до инициатора, для чего используется вектор инициатора в сообщении. Автомат каждой вершины отмечает как прямую ту дугу, по которой посылает сообщение ***Прямое***. Инициатор, получив сообщение ***Прямое***, создаёт сообщение ***Обратное***, переписывая в него обратный вектор. Заметим, что вектор каждой вершины на пути, который проходит сообщение ***Прямое***, является префиксом вектора инициатора в сообщении.

Сообщение ***Обратное*** двигается по обратному пути и размечает обратные дуги. Это сообщение содержит остаток обратного вектора: автомат каждой вершины запоминает номер той дуги, по которой посылает сообщение ***Обратное***, как номер обратной дуги, и удаляет этот номер из обратного вектора в сообщении. Если раньше был запомнен другой номер, то он забывается.

Почему сообщение ***Обратное*** идёт до корня, а не до какой-нибудь вершины, где уже есть обратная дуга? Дело в том, что иначе может возникнуть цикл обратных дуг. Это видно на Рис. 1. Здесь одновременно размечаются два обратных пути от разных инициаторов (чёрный и серый кружки) до корня (белый кружок внизу), обозначенные одинарной и двойной линиями (1). Пунктир – ещё не пройденная часть пути. Если сообщения идут не до корня, то может получиться цикл обратных дуг (2). Если же сообщения продолжают движение, то возникают новые обратные дуги (3), а старые забываются (4). И дальше до самого корня получается фрагмент обратного остова.

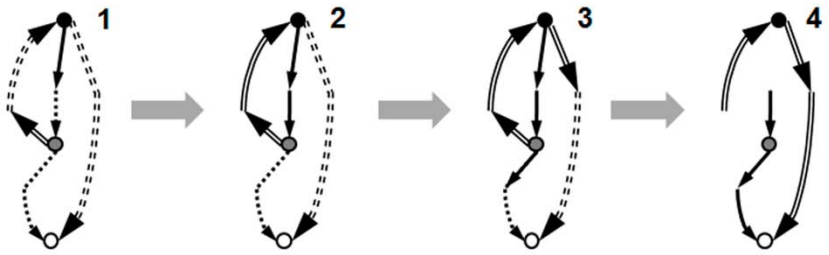

Рис. 1. Сообщение ***Обратное*** идёт до корня.

3.2. Определение конца построения остовов

Как определить конец построения прямого и обратного остовов? Идея основана на том, что у дуги есть одно начало и один конец. Считаем дуги по их началу, то есть выходящие дуги, и по их концу, то есть входящие дуги. Оба числа должны, в конце концов, совпасть. Для этого в автомате корня ведётся подсчёт числа дуг графа. Сначала счётчик равен числу дуг, выходящих из корня.

Модифицируем сообщение ***Поиск***, добавив в него параметр: число дуг, выходящих из вершины-инициатора. Это число автомат корня прибавит к счётчику дуг, когда получит сообщение ***Поиск***. Это подсчёт дуг по их началу.

Ещё добавим два новых сообщения: ***Финиш*** и ***Минус***. Сообщение ***Финиш*** посылается из начала дуги в её конец при получении сообщения ***Прямое*** и заставляет конец дуги учесть эту входящую дугу. Сообщение ***Минус*** предназначено для того, чтобы вычитать из счётчика дуг число учтённых дуг, входящих в вершину. Это подсчёт дуг по их концу.

Сообщение ***Минус*** посылается по обратному остову до корня. До того, как появилась обратная дуга, автомат вершины ведёт счётчик учтённых входящих дуг, значение которого и посылает в первом сообщении ***Минус***, когда появляется обратная дуга. Но и после этого в автомат вершины могут продолжать приходить сообщения ***Финиш*** по ещё неучтённым входящим дугам, которые будут вызывать уже немедленную отправку сообщений ***Минус***.

Как это всё работает? Пусть у нас есть две смежные прямые дуги *ab* и *bc*. На Рис. 2 показана временная диаграмма с последовательностями сообщений.

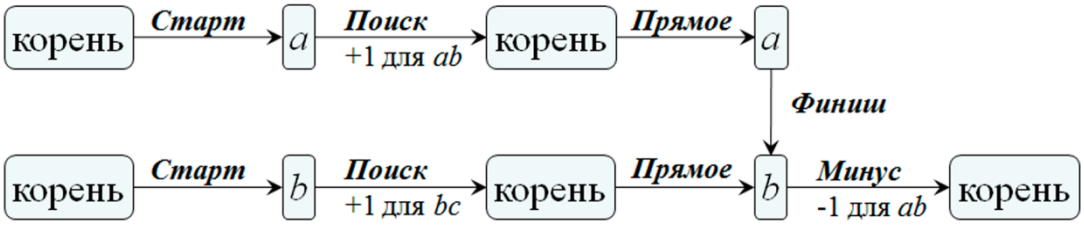

Рис. 2. Временная диаграмма для определения конца построения остовов.

Сообщение ***Поиск*** от автомата вершины *a* прибавляет к счётчику дуг в автомате корня единицу для дуги *ab*, а сообщение ***Поиск*** от автомата вершины *b* прибавляет к этому счётчику единицу для дуги *bc*. Сообщение ***Минус*** от автомата вершины *b* вычитает из счётчика единицу для дуги *ab*. Мы видим, что для каждой дуги *ab* вычитание единицы из счётчика происходит не только после прибавления к нему единицы для этой дуги *ab*, но и после прибавления единицы для каждой следующей дуги *bc*. Из-за этого счётчик дуг в автомате корня обнулится только после того, как прямой и обратный остовы будут построены.

3.3. Установка счётчиков числа входящих обратных дуг

После того, как построены прямой и обратный остовы и определен конец построения, выполняется установка счётчиков входящих обратных дуг. Для этого используются два сообщения: ***Начало*** и ***Конец*** подсчёта. Сообщение ***Начало*** создаётся автоматом корня и рассылается по прямому остову. Получив такое сообщение, автомат вершины создаёт сообщение ***Конец***, которое посылается по обратному остову до корня.

Сначала в сообщении ***Конец*** есть признак «*первая обратная дуга*». По этому признаку автомат вершины добавляет единицу к своему счётчику входящих обратных дуг, а дальше ***Конец*** пересылается уже без этого признака.

Сообщения ***Конец*** нужны автомату корня, чтобы определить завершение этого этапа, да и всей разметки. Каким образом? От каждой вершины до корня

дойдёт ровно одно сообщение ***Конец***. Поэтому автомату корня достаточно «знать» число вершин. А он это «знает» как число элементов во множестве инициаторов, которое хранилось в автомате корня, когда передавались сообщения ***Поиск***.

3.4. Оценки алгоритма

Время работы алгоритма $O(n/k + d)$, где n – число вершин графа, k – ёмкость дуги, d – диаметр графа (длина максимального пути). Память автомата вершины $O(ndlogs_{out})$, где s_{out} – ограничение сверху на полустепень исхода вершин. Размер сообщения $O(dlogs_{out})$.

4. Параллельные вычисления и агрегатные функции

Теперь мы переходим ко второй задаче – параллельным вычислениям на графе. Что будет параллельно вычисляться? Это будет функция от мультимножества значений в вершинах графа. Можно считать, что эти значения записаны в вершины графа, а автоматы могут их читать. Или у каждого автомата есть операция – получить значение для той вершины, в которой он находится. Способ, каким автомат узнаёт значение, приписанное вершине, не важен.

Сначала посмотрим, как устроены те функции, которые будут вычисляться. Пусть X – это некоторое базовое множество. Вершинам будут приписываться значения из этого множества X. Через $X^{\#}$ обозначим множество всех конечных мультимножеств элементов из X. Нам нужно мультимножество, потому что разным вершинам могут быть приписаны одинаковые значения.

Функция g на множестве $X^{\#}$ называется *агрегатной*, если существует такая функция e, что значение функции g от объединения двух мультимножеств a и b можно вычислить так: сначала вычисляем g для каждого из этих мультимножеств, а потом к двум полученным значениям применяем функцию e. Формально: $\forall a,b \in X^{\#} \quad g(a \cup b) = e(g(a),g(b))$. Иными словами, агрегатную функцию можно вычислить «по частям».

И. Бурдонов, А. Косачев.
Исследование ориентированного графа коллективом неподвижных автоматов.
Программная инженерия. т.8, №1, 2017, стр. 16-25.
10 стр.

Примеры агрегатных функций: сумма, минимум и сумма квадратов.

$\Sigma : \{a_1,\ldots a_n\} \to (a_1+\ldots+a_n)$ $\quad [e = +]$

$min : \{a_1,\ldots a_n\} \to min\{a_1,\ldots a_n\}$ $\quad [e = min]$

$Q : \{a_1,\ldots a_n\} \to (a_1^2+\ldots+a_n^2)$ $\quad [e = +]$

Понятно, что не все функции являются агрегатными. Например, среднее арифметическое не агрегатно. Однако можно использовать *агрегатное расширение* функции. Если функцию f можно представить как композицию $f = h \circ g$ двух функций h и g, где g – агрегатная функция, то эта функция g называется агрегатным расширением функции f. Это значит, что мы можем делать вычисления «по частям», используя агрегатное расширение g, а в конце один раз применяем функцию от одного аргумента – функцию h.

Однако возможны такие расширения, которые на практике ничего не дают. Например, можно взять в качестве g тождественную функцию на $X^{\#}$, а в качестве функции h – саму функцию f. Чтобы избежать этого, используется *минимальное* агрегатное расширение. Интуитивно, это агрегатная функция, вычисляющая минимум информации, по которой еще можно восстановить f. Формально функция $g : X^{\#} \to B$ минимальное агрегатное расширение f, если $g(X^{\#}) = B$ и для каждого агрегатного расширения $g` : X^{\#} \to B`$ функции f существует такая функция $j : B` \to B$, что $g = j \circ g`$. Минимальное расширение для любой функции f существует и единственно с точностью до изоморфизма. Примеры минимальных агрегатных расширений:

$f : \{a_1,\ldots,a_n\} \to (a_1+\ldots+a_n)/n$	среднее арифметическое
$g : \{a_1,\ldots,a_n\} \to (a_1+\ldots+a_n, n)$	$h : (a, n) \to a/n$
$f : \{a_1,\ldots,a_n\} \to (a_1\times\ldots\times a_n)^{1/n}$	среднее геометрическое
$g : \{a_1,\ldots,a_n\} \to (a_1\times\ldots\times a_n, n)$	$h : (a, n) \to a^{1/n}$
$f : \{a_1,\ldots,a_n\} \to ((a_1^2+\ldots+a_n^2)/n)^{1/2}$	среднее квадратичное
$g : \{a_1,\ldots,a_n\} \to (a_1^2+\ldots+a_n^2, n)$	$h : (a, n) \to (a/n)^{1/2}$.

Такая теория агрегатных функций – это модификация теории индуктивных функций [5], доказательства утверждений смотри в [4].

5. Параллельные вычисления на статическом графе

Теперь перейдём к алгоритму вычисления функции на статическом графе (подробнее в [4,6]). Для этого будет использоваться разметка графа из раздела 3. Заметим, что разметка делается один раз, а потом может использоваться для многократного вычисления различных функций.

Пусть вершинам приписаны значения из базового множества X. Вычисление начинается, когда автомат корня получает извне сообщение, содержащее три функции: h, g и e. На практике могут присылаться программы вычисления значений этих функций, или какие-то ссылки на эти программы, по которым автоматы могут получить к ним доступ. Вычисление основано на алгоритме пульсации.

От корня вверх по прямому остову распространяется сообщение ***Вопрос***, содержащее две функции: g и e. А вниз по обратному остову распространяется сообщение ***Ответ***, содержащее значение функции g от мультимножества значений в вышележащих вершинах обратного остова.

Автомат листовой вершины обратного остова, получив ***Вопрос***, вычисляет функцию g от значения, записанного в этой вершине, и посылает его как параметр сообщения ***Ответ*** по обратной дуге.

Автомат нелистовой вершины точно также вычисляет функцию g от значения в этой вершине, запоминает его, но не посылает, пока не получит ***Ответы*** по всем входящим обратным дугам. Число таких дуг уже установлены в автоматах вершин при разметке графа. Каждый раз, получив ***Ответ*** по входящей обратной дуге, автомат вершины применяет функцию e, параметрами которой будет значение, полученное в сообщении ***Ответ***, и запомненное значение. Когда получены ***Ответы*** по всем входящим обратным дугам,

автомат вершины сам посылает по выходящей обратной дуге ***Ответ*** с накопленным значением функции g.

Автомат корня посылает ***Ответ*** вовне, предварительно применив к накопленному значению функции g завершающую вычисления функцию h.

Время вычисления не более $3d$, память автомата $\boldsymbol{O}(s_{out}+\log s_{in}+y+z)$, размер сообщения $\boldsymbol{O}(y+z)$, где d – диаметр графа, $s_{out}\leq m$ – максимальная полустепень исхода вершины, $s_{in}\leq m$ – максимальная степень захода вершины, y – размер вопроса (функций h, g и e), z – размер ответа (значения функции).

6. Неподвижные автоматы на динамическом графе

Динамическим графом будем называть такой граф, дуги которого могут меняться со временем. При этом мы будем считать, что вершины графа, в которых «сидят» автоматы, не изменяются. Есть три вида изменения дуг:

1) Дуга может появиться.

Об этом автомат начала дуги извещается специальным сигналом *появления дуги* с параметром номер дуги.

2) Дуга может исчезнуть.

Если по дуге передавалось сообщение, то оно пропадает, о чём автомат начала дуги извещается сигналом *исчезновения дуги* с параметром номер дуги. Если же на дуге не было сообщений, то при её исчезновении никаких сигналов не возникает. Однако, если автомат вершины попытается послать сообщение по исчезнувшей выходящей дуге, сообщение не будет передано, а автомат получит сигнал исчезновения дуги.

3) Дуга может поменять свою конечную вершину.

В этом случае никаких сигналов не предусмотрено.

Заметим, что мы выбрали модель с минимальным набором сигналов, всё ещё позволяющих рано или поздно распознавать изменение дуги. Кроме указанных, есть ещё сигнал *освобождения дуги*, означающий, что сообщение, которое по дуге передавалось, уже передано и можно послать новое сообщение.

Мы будем предполагать, что ёмкость дуги равна 1: одновременно по дуге передаётся не более одного сообщения.

Если дуга меняется слишком часто и автомат не успевает обрабатывать сигналы от этой дуги, то могут возникнуть два необработанных сигнала: старый и новый. Заметим, что новым сигналом может быть только сигнал появления дуги. В этом случае работает правило замещения, которое говорит, какой из двух сигналов остаётся, а какой теряется: появление + появление → появление, исчезновение + появление → появление, освобождение + появление → освобождение или появление. Для того, чтобы автоматы могли отслеживать изменения дуг, важно, чтобы после сигнала исчезновения дуги не потерялся сигнал её появления, иначе автомат останется «в убеждении», что дуги нет. Если же после сигнала освобождения дуги возникает сигнал её появления, то это означает, что сообщение передано по дуге, затем дуга исчезла (без сигнала), а потом опять появилась. Это эквивалентно смене конца дуги, поэтому всё равно, какой из двух сигналов остаётся, а какой теряется.

Граф будем считать *нумерованным*: все вершины пронумерованы и в каждой вершине записан её номер, который автомат может прочитать.

7. Мониторинг динамического графа

Ставится задача *мониторинга* графа: сбор полной информации о графе и отслеживание изменений дуг. Мы будем предполагать, что такая информация собирается не в каком-то выделенном автомате, а в каждом автомате. На самом деле для динамического графа это всё равно.

Понятно, что если граф постоянно меняется, то мы не можем гарантировать, что описания текущего состояния всех его дуг отражены во всех автоматах: сообщения о последних изменениях дуг могут просто не дойти до каких-то автоматов. Поэтому требуется только, чтобы через некоторое конечное время T_0 после изменения дуги все автоматы «узнали» об этом или более позднем изменении дуги. Если после данного изменения дуга больше не

меняется, по крайней мере, в течение времени T_0, то во всех автоматах будет одинаковое (и верное) описание этой дуги.

Если дуга меняется так часто, что никакое сообщение по ней не успевает дойти или уходит неизвестно куда, то нельзя гарантировать возможность доставки сообщения в любую вершину. Это нужно как-то ограничить. Будем говорить, что дуга *долгоживущая*, если за то время, пока она не меняется, по ней может успеть пройти хотя бы одно сообщение. Мы будем пренебрегать временем срабатывания автомата, а время передачи сообщения по дуге ограничим сверху 1 тактом. Так что долгоживущая дуга должна не меняться хотя бы в течение 1 такта. Ограничение, которое нам нужно, звучит так: в каждый момент времени подграф, порождённый долгоживущими дугами, является сильно связным суграфом, то есть содержит все вершины графа.

Алгоритм мониторинга подробно описан в [7], здесь мы рассмотрим только основную идею алгоритма и приведём оценки времени работы без доказательств.

7.1. Описания дуг и сообщения

Полная информация о графе, которую нужно собрать, это множество описаний всех его дуг. Описание дуги – это тройка: номер начала дуги, то есть начальной вершины дуги, номер дуги (в её начале) и номер конца дуги, то есть конечной вершины дуги. Если номер конца дуги ещё не известен, то указывается номер 0 (мы считаем, что вершины пронумерованы, начиная с 1).

Сразу можно сказать, что в одном сообщении автомат должен посылать сразу описания всех известных ему дуг. Это *правило одного сообщения*. Оно объясняется тем, что по долгоживущей дуге гарантированно проходит только одно сообщение, поэтому в него надо поместить всю имеющуюся информацию. Во-вторых, сообщение нужно посылать по дуге не тогда, когда информация в автомате обновилась, а каждый раз, когда появляется такая возможность, то есть при появлении дуги и при освобождении дуги. В противном случае

информация от автомата данной вершины может не достигнуть других автоматов, поскольку выходящая из вершины дуга может несколько раз менять свой конец без какого-либо сигнала для автомата этой вершины.

Кто и когда обнаруживает изменение дуги, в результате чего создаёт или корректирует описание дуги? Если дуга появляется, то это обнаруживает автомат в начале дуги по сигналу появления дуги. Но он ещё не знает конца дуги. Если дуга исчезает, то это обнаружит тоже автомат начала дуги по сигналу исчезновения дуги. Этот сигнал он получит либо сразу, если по дуге передавалось сообщение, либо позже, когда при обработке предыдущего сигнала освобождения дуги попытается послать по дуге сообщение. Если дуга меняет свой конец, то это обнаруживает автомат нового конца дуги, когда получает по дуге сообщение, в котором описание дуги содержит другой номер конца. В частности, номер конца в сообщении будет нулевым, если автомат начала дуги ещё не знает конца дуги, то есть сразу после появления дуги. Для того, чтобы автомат конца дуги мог это делать, он должен знать, по какой дуге передавалось сообщение. Для этого сообщение, кроме множества описаний дуг, содержит указание на описание дуги, по которой оно передаётся.

7.2. Ранг дуги

Поскольку дуга может меняться несколько раз, автомат, получая сообщение, должен «понять», получил ли он описание дуги более новое, чем то, что у него есть, или нет. Новое описание дуги должно заменять собой старое описание. Для этого вводится ранг описания дуги, который будем называть просто *рангом дуги*. Каждый раз, когда какой-то автомат обнаруживает изменение дуги и поэтому создаёт или корректирует её описание, ранг дуги в этом описании увеличивается.

Однако недостаточно просто увеличить ранг дуги, например, на 1. Нужно быть уверенным, что после такого увеличения в любом другом автомате дуга имеет строго меньший ранг. Здесь возникает проблема из-за серии изменений

конца дуги, происходящих в течении времени, пока автомат начала дуги ещё не «узнал» ни об одном из этих изменений. Автоматы этих сменяющих друг друга концов дуги увеличивают один и тот же ранг, естественно, одинаково, например, на 1. Поэтому сразу в нескольких автоматах эта дуга будет иметь одинаковый, увеличенный на 1, ранг. Эту проблему решает автомат начала дуги, который «оставляет последнее слово всегда за собой». Каждый раз, когда он получает описание дуги с большим рангом, чем у него хранится, он ещё раз увеличивает ранг дуги на 1. Соответственно, при появлении или исчезновении дуги автомат начала дуги тоже увеличивает её ранг сразу на 2.

7.3. Оценки

Время T_0 имеет порядок числа вершин $O(n)$. Но если все изменения в графе прекращаются, то время T_1, которое нужно, чтобы во всех автоматах оказалась правильная информация, учитывающая последние изменения, ещё меньше – порядка диаметра графа $O(d)$.

Размер памяти автомата (и сообщения) довольно большой $O(mlogsnv)$, где m – число дуг графа, s – ограничение сверху на полустепень исхода вершин, v – максимальное число изменений одной дуги. Но это и естественно, поскольку в памяти автомата создаётся описание всего графа. Это же описание, по сути, передаётся в сообщении, размер которого имеет тот же порядок.

Можно обратить внимание, что размер памяти зависит от максимального числа изменений одной дуги. Это затраты на ранг дуги. Для того, чтобы оценка памяти не зависела от числа изменений, можно предложить две модификации модели.

7.4. Модификации модели

Первая модификация самая простая: потребуем, чтобы все дуги были долгоживущими. Тогда число изменений одной дуги не превосходит времени работы алгоритма в тактах, а оно имеет порядок числа вершин n.

Следовательно, ранг можно сделать циклическим с максимальным значением порядка n.

Вторая модификация – наличие таймера, который посылает каждому автомату временной сигнал каждый такт. Тогда все те действия, что автомат выполнял при обработке сигнала освобождения или появления дуги, он будет делать при обработке временного сигнала. Сообщения будут посылаться по дуге не чаще одного за такт. При этом время T_0 не изменится по порядку. А тогда опять ранг можно сделать циклическим с максимальным значением порядка n.

Правда, при второй модификации время существования «долгоживущих» дуг придётся увеличить с одного до двух тактов. В противном случае может оказаться, что каждый раз при появлении дуги сообщение посылается по ней не сразу, а спустя какое-то время (до получения временного сигнала), из-за чего сообщение не успевает передаться по дуге до её изменения.

8. Параллельные вычисления на динамическом графе

Параллельные вычисления на динамическом графе подробно описаны в [8], здесь мы рассмотрим только основные идеи алгоритмов и приведём оценки времени и памяти без доказательств. Для параллельных вычислений на динамическом графе нам тоже потребуется предварительная разметка графа. Мониторинг не даёт такой разметки – у него вообще другая задача. На статическом графе вычисления выполнялись с помощью сообщений ***Вопрос*** и ***Ответ***, которые использовали разметку графа, состоящую из прямого и обратного остовов и счётчики обратных дуг. Но если граф динамически меняется, возникает проблема: когда меняется дуга остова, нужно этот остов скорректировать. В общем случае это может потребовать полной перестройки остовов, т.е. фактически придётся заново их строить. Кроме того, не гарантируется доставка сообщений по дугам остова: эти дуги могут исчезать или менять свой конец до того, как по ним будет передано нужное сообщение.

В то же время, как мы видели в случае мониторинга графа, предположение о долгоживущих дугах гарантирует возможность доставки информации из любой вершины в любую вершину. Но для этого автомат вершины должен следовать правилу одного сообщения. Получив сообщение с нужной информацией, он сохраняет её в своей памяти и далее помещает во все сообщения, которые постоянно посылает по всем выходящим дугам, когда появляется такая возможность, то есть по сигналам освобождения и появления дуги.

Для доставки сообщения ***Вопрос*** этого достаточно, и никакого прямого дерева не требуется. Точно также можно было бы доставить в корень значения из всех вершин графа и уже в автомате корня выполнить вычисление функции от получившегося мультимножества. Однако, поскольку мы всю информацию вынуждены помещать в одно сообщение, размер такого сообщения был бы максимальным и пропорциональным числу вершин графа. Обратный остов и счётчики обратных дуг позволяют аккумулировать ответы: автомат вершины, получив ответы по всем входящим обратным дугам, вычисляет промежуточное значение агрегатной функции и посылает дальше по выходящей обратной дуге только один ответ. Тем самым, сообщение содержит ответ только от автомата одной вершины каждой ветви обратного остова от корня до листа. Размер сообщения становится пропорциональным не числу вершин, а ширине обратного остова.

Итак, обратный остов и счётчики обратных дуг нам нужны только для уменьшения размера сообщения, содержащего ответы, то есть вычисленные промежуточные значения агрегатной функции. Такой остов может быть виртуальным. Вершины в нём настоящие, а дуги – виртуальные. Дуга задаётся просто парой её вершин: начальной и конечной. Поскольку остов виртуальный, у нас появляется свобода: какой остов лучше построить. Его ширина w (число листовых вершин, кроме корня) определяет размер сообщения, а высота h

(длина максимальной ветви от корня до листа) – время вычисления, поскольку вычисления распараллеливаются только между разными ветвями остова, а по одной ветви выполняются последовательно от листа к корню.

Возникает задача: для данного числа вершин определить минимальную высоту дерева при заданной ширине, и описать вид такого дерева. Оно будет выглядеть как «сбалансированный веник» на Рис. 3.

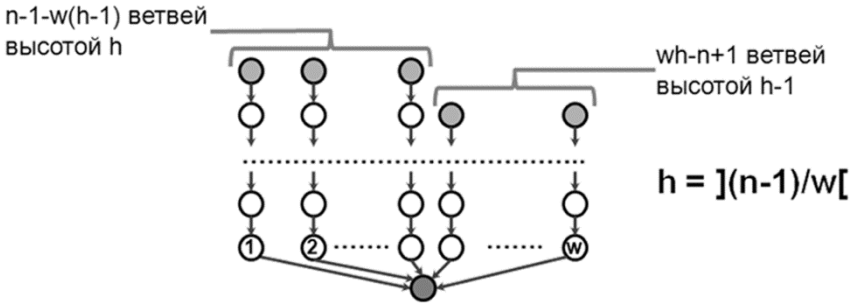

Рис. 3. «Сбалансированный веник».

«Веник» можно задать с помощью двухуровневых индексов вершин, отличных от корня. Индекс вершины состоит из номера ветви веника от 1 до *w* и номера на этой ветви от 1 до *h*, считая от корня. В таком «венике» счётчик входящих обратных дуг нужен только в автомате корня, его значение равно *w*. Автомату другой вершины достаточно «знать», является вершина листовой (счётчик равен 0) или внутренней (счётчик равен 1) вершиной веника.

8.1. Разметка динамического графа

Разметка выполняется в 2 этапа: 1) Сначала в автомате корня собирается информация о всех вершинах. Используется сообщение ***Прямое***. 2) Затем автомат корня строит в своей памяти веник и рассылает его во все вершины. Используется сообщение ***Обратное***.

Корень, получив стартовое сообщение извне, создаёт сообщение ***Прямое***, которое потом циркулирует по всему графу с помощью постоянной веерной рассылки. В этом сообщении и в автоматах вершин, через которые оно проходит, накапливаются описания дуг. Описание дуги, как и в случае мониторинга динамического графа, содержит номер начальной вершины дуги,

номер дуги в её начальной вершине и номер конечной вершины дуги, если он известен, или знак вопрос, если неизвестен. Долгоживующие дуги гарантирует доставку сообщения до любой вершины, и из любой вершины до корня.

Автомат корня проверяет замкнутость по дугам: нет знаков вопроса, то есть для каждой дуги известна конечная вершина этой дуги. Однако такая «статическая» замкнутость гарантирует, что все вершины известны, только для статического графа. В динамическом графе конец дуги может меняться без всяких сигналов, поэтому замкнутость по дугам ничего не гарантирует.

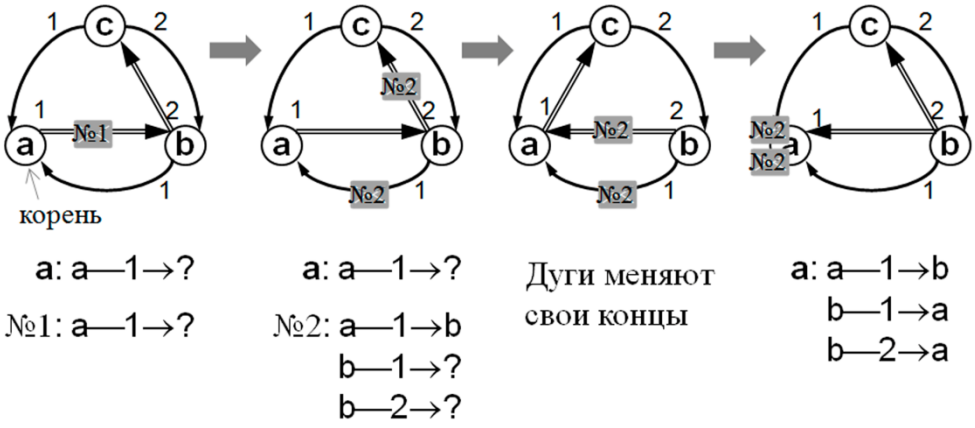

Рис. 4. Недостаток статической замкнутости.

Пример на Рис. 4. Автомат корня, вершины *a*, посылает сообщение № 1, в котором одна дуга с неизвестным концом. Автомат вершины *b*, получив сообщение, записывает вершину *b* как конец этой дуги и рассылает по двум выходящим дугам сообщение № 2, в котором неизвестны концы этих двух дуг. Одно направляется в корень *a*, а другое – в вершину *c*. Однако две дуги, помеченные двойной линией, меняют свои концы, поэтому оба сообщения № 2 приходят в корень. И автомат корня, как и положено, записывает корень как конечную вершину этих дуг. В результате в автомате корня получается замкнутое множество дуг, однако вершина *c* осталась неизвестной.

Для того, чтобы замкнутость гарантировала известность всех вершин, нам требуются два дополнительных условия.

Первое условие – это новое правило замещения сигналов: освобождение + появление → освобождение. Дело в том, что важно знать, дошло сообщение по дуге или нет, т.е. сигнал освобождения не должен теряться.

Второе условие: нам понадобятся *начальные дуги*, которые обладают двумя свойствами: 1) начальная дуга существует с самого начала и не меняется до тех пор, пока по ней не пройдёт первое сообщение; 2) по начальным дугам из корня достижимы все вершины.

Теперь у нас будет модифицированное условие «динамической» замкнутости: для каждой известной дуги либо известна конечная вершина этой дуги (по дуге послано первое сообщение, оно дошло до конца дуги и вернулось к корню), либо по этой дуге не прошло первое сообщение (дуга гарантированно не начальная).

Важно, что автомат каждой вершины регистрирует только те выходящие из этой вершины дуги, которые появились до получения им первого сообщения. А те, что появляются позже, не регистрируются. Почему?

Во-первых, этого достаточно, поскольку все начальные дуги будут зарегистрированы, так как появляются с самого начала. Тем самым будут зарегистрированы все вершины, поскольку они достижимы по начальным дугам.

Во-вторых, это важно для минимизации времени сбора информации о вершинах. Оно будет порядка числа вершин n. Если бы автоматы регистрировали все m дуг, то время увеличилось бы. Действительно, дуги могут всё время появляться, и, если они продолжают регистрироваться, то замкнутость в корне может возникнуть после появления всех m дуг, потому что для каждой появившейся дуги придётся ждать выяснения, какой же у неё конец. Пример на Рис. 5. Каждый раз появляется одна петля, т.е. когда в корень приходит информация о конце i-ой петли, к нему же приходит информация о

появлении *i+1*-ой петли, но без конца. Поэтому корень ждёт, пока появятся все *m* петель, т.е. ***O(m)*** тактов.

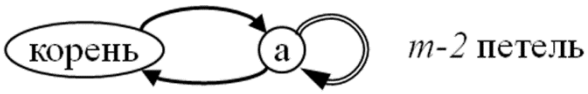

Рис. 5. Пример ожидания регистрации всех *m* дуг.

Заметим также, что для разметки динамического графа можно обойтись без нумерованности графа, если использовать идентификацию вершин с помощью динамически создаваемых векторов вершин, как это делалось для разметки статического графа. Для этого используется сообщение ***Старт***, которое накапливает в себе вектор маршрута, который оно проходит. Корень получает ***Старт*** извне с пустым вектором. Автомат каждой вершины, получив первый раз сообщение ***Старт***, запоминает его вектор маршрута как вектор вершины и посылает ***Старт*** по всем выходящим дугам. Регистрация выходящих дуг прекращается. Повторные сообщения ***Старт*** игнорируются. Начальные дуги гарантируют, что каждая вершина получит сообщение ***Старт*** и, тем самым, получит свой вектор. Поскольку по каждой дуге посылается не более одного сообщения ***Старт***, вектор вершины будет уникальным в графе.

Только после этого автомат вершины начинает выполнять 1-ый этап разметки, т.е. работать с сообщением ***Прямое***: каждый раз, когда дуга освобождается или появляется, по ней посылается сообщение ***Прямое*** с накопленным множеством описаний дуг. При получении сообщения ***Прямое*** множество описаний дуг из сообщения объединяется с множеством описаний дуг, хранящимся в памяти автомата. При этом вектор вершины-получателя вставляется в описание дуги, по которой принято сообщение, как вектор конца дуги.

Когда автомату корня становятся известны все вершины, он создаёт описание виртуального «веника», которое рассылается веером по всему графу в сообщении ***Обратное***, начинается 2-ой этап разметки. Описание «веника» – это

множество описаний вершин. Для каждой вершины по её номеру указывается её индекс в «венике» и признак: листовая или не листовая. Когда автомат вершины первый раз получает сообщение с описанием «веника», он запоминает отдельно описание своей вершины, удаляет его из описания «веника» и такое уменьшенное описание «веника» сохраняет и рассылает дальше. При получении повторного сообщения автомат вершины строит пересечение множества описаний вершин, которое хранится в нём, и которое содержится в сообщении. Это пересечение сохраняется и рассылается дальше. Рано или поздно сообщение становится пустым и через какое-то время описание «веника» в автомате корня тоже становится пустым. Это и означает завершение 2-го этапа разметки. Далее система готова к параллельным вычислениям.

8.2. Алгоритм пульсации

Как и в случае статического графа для параллельных вычислений на динамическом графе применяется алгоритм пульсации вопросов и ответов. Однако по правилу одного сообщения вся нужная информация должна рассылаться по графу в одном сообщении, распространяемом веером с размножением в каждой вершине. Поэтому на этапе вычисления функции распространяется сообщение ***Вопрос-Ответ***, которое содержит как вопрос, так и ответы. При этом для каждой ветви «веника» в сообщении не больше одного ответа, а именно ответа от автомата вершины с наименьшим номером по ветви. Ответ задаётся вместе с индексом отправителя.

После завершения вычисления по графу может продолжать циркулировать такое сообщение ***Вопрос-Ответ***. Оно должно вытесняться сообщением для следующего вычисления. Для этого автомат корня нумерует вопросы в порядке их поступления извне и номер вопроса помещается в сообщение.

Итак, сообщение ***Вопрос-Ответ*** содержит: номер вопроса, вопрос (функции g и e), множество пар <ответ (значение функции g), индекс вершины-отправителя ответа>.

8.3. Оценки

Время разметки графа $O(n)$, время вычисления функции $O(n^2/w)$. Память автомата и размер сообщения: на этапе разметки $O(m\log ns_{out})$, на этапе вычисления: $O(y+\log N+wz+w\log n)$. Здесь n – число вершин, m – число дуг, s_{out} – ограничение на полустепень исхода вершин, y – размер вопроса (описаний функций h, g и e), z – размер ответа (значения функции), N – максимальное число вопросов.

9. Заключение

В серии из [1,2] и данной статьи представлены результаты по исследованию графов с помощью автоматов. В [1] изучались возможности одного автомата, в том числе конечного (робота), а в следующих статьях – коллектива автоматов. В [1,2] автоматы двигались по дугам графа и, если их было несколько, обменивались между собой сообщениями по независимой от графа сети связи, гарантирующей доставку сообщение «от точки к точке» по адресу получателя. В данной статье автоматы ассоциированы с вершинами графа и никуда не двигаются, но обмениваются сообщениями, пересылаемыми по дугам графа.

В частности, в [2] рассматривалась работа коллектива двигающихся автоматов на недетерминированном графе, а в данной статье – работа коллектива неподвижных автоматов на динамически меняющемся графе. Темой дальнейших исследований могла бы стать, наоборот, работа двигающихся автоматов на динамическом графе и неподвижных автоматов – на недетерминированном графе. К числу нерешённых задач можно отнести также Δ-обход недетерминированного графа коллективом достаточно большого числа двигающихся полуроботов.

Для конечного автомата (робота) не решена основная задача «объяснения» разрыва между длиной минимального обхода ориентированного графа $O(nm)$ и обхода наилучшим из известных роботов с оценкой $O(nm+n^2\log\log n)$. Прежде

всего, не известно, существует ли робот, выполняющий обход с минимальной оценкой ***O(nm)***, и если не существует, то какова наилучшая возможная оценка для робота. Для двух взаимодействующих роботов достигается оценка ***O(nm)***, но неизвестно, может ли она быть улучшена и насколько при наличии большого числа роботов.

Если формально «инвертировать» модель неподвижных автоматов из данной статьи, то получится модель двигающихся автоматов, которая отличается от модели в [2] двумя свойствами. Во-первых, автоматы не обмениваются между собой сообщениями, а только через память вершин. Предполагается, что срабатывание автомата, при котором он читает из вершины, меняет своё состояние, записывает в вершину и указывает номер выходящей дуги, по которой будет двигаться, является атомарным действием. Таким образом осуществляется синхронизация по доступу к памяти вершин. Во-вторых, автоматы могут исчезать и генерироваться другими автоматами в любой вершине, а не только в корне (в [2] регуляторы, генерируемые в любой вершине, теряют возможность перемещаться по графу). Такая модель может стать темой дальнейших исследований.

Следует отметить, что рассматриваемые модели и полученные результаты в этих трёх статьях не исчерпывают всей области автоматов на графах.

Литература

1. И.Б.Бурдонов, А.С.Косачев. Исследование графа автоматом. // Программная инженерия. — год. — Т.???, №???. — С. ???-???.
2. И.Б.Бурдонов, А.С.Косачев. Исследование графов коллективом двигающих автоматов. // Программная инженерия. — год. — Т.???, №???. — С. ???-???.
3. И.Б.Бурдонов, А.С. Косачев. Построение прямого и обратного остовов автоматами на графе. // Труды Института системного программирования РАН. — 2014. — Том 26, №6. — С. 57-62.

4. И.Б.Бурдонов, А.С.Косачев, В.В. Кулямин. Параллельные вычисления на графе. // Программирование. — 2015. — №1. — С. 3-20.
5. А.Г.Кушнеренко, Г.В.Лебедев. Программирование для математиков. — М. : Наука, Главная редакция физико-математической литературы, 1988.
6. И.Б.Бурдонов, А.С.Косачев, В.В.Кулямин. Параллельные вычисления автоматами на прямом и обратном остовах графа. // Труды Института системного программирования РАН. — 2014. — Том 26, №6. — С. 63-66.
7. И.Б.Бурдонов, А.С.Косачев. Мониторинг динамически меняющегося графа. // Труды Института системного программирования РАН. — 2015. — Том 27, №1. — С. 69-96.
8. И.Б.Бурдонов, А.С.Косачев. Параллельные вычисления на динамически меняющемся графе. // Труды Института системного программирования РАН. — 2015. — Том 27, №2. — С. 189-220.

И. Бурдонов, А. Косачев.
Исследование ориентированного графа коллективом неподвижных автоматов.
Программная инженерия. т.8, №1, 2017, стр. 16-25.
10 стр.
