УДК 519.1

АНАЛИЗ ИЗВЕСТНЫХ ПОДХОДОВ К РЕШЕНИЮ ЗАДАЧИ ПОИСКА ТРАНЗИТИВНО - РЕФЛЕКСИВНЫХ ЗАМЫКАНИЙ

В.Н.Бацамут
(представил д.ф.-м.н. С.В. Смеляков)

В статье проводится анализ и сравнительная оценка функциональных возможностей существующих методов решения задачи выделения транзитивности и рефлексивности на объектах с сетевой организацией.

Быстрое и точное определение наличия пути между различными элементами системы распределенного типа имеет большое прикладное значение в задачах топологического контроля данных объектов, что в свою очередь, влияет на устойчивость функционирования системы в целом, ее пропускную способность, оперативность доведения информации до абонентов, степень использования каналов и т.д.

В том случае, когда необходимо установить лишь факт наличия пути между различными узлами системы, возникает задача построения транзитивного замыкания (ТЗ).

В настоящее время разработано большое количество методов поиска ТЗ в распределенных системах. Однако проблема создания эффективных алгоритмов данного класса остается актуальной. В литературе не представлены подходы, которые бы, наряду с поиском ТЗ, однозначно определяли и наличие рефлексивных замыканий (РЗ) для узлов таких систем. Также нет четкой классификации существующих алгоритмов поиска ТЗ. Как правило, все они опираются на матричные преобразования, т.е. данные в них представляются в виде различных матриц (смежности, достижимостей, контрдостижимостей, весов и т.д.), различаются по своим функциональным возможностям и вычислительной сложности. При этом под вычислительной сложностью алгоритма понимается число элементарных операций выполняемых в том или ином алгоритме в асимптотически худшем случае [5].

Рассмотрим наиболее известные и эффективные алгоритмы, решающие задачу построения ТЗ на графовых структурах. С проблемой поиска ТЗ тесно связана задача определения кратчайших путей на графе. Алгоритмы решающие эту задачу можно условно отнести к алгоритмам поиска ТЗ на графах. Рассмотрим основные.

Алгоритм Дейкстры [3]. Он позволяет находить в графе кратчайший путь между двумя выделенными вершинами **s** и **t** при положительных длинах дуг и считается одним из наиболее эффективных алгоритмов

решения данной задачи. Основная идея алгоритма состоит в следующем. Предположим, что известны **m** вершин, ближайших к вершине **s** (близость любой вершины **x** к вершине **s** определяется длиной кратчайшего пути, ведущего из **s** в **x**). Пусть также известны сами кратчайшие пути, соединяющие вершину **s** с выделенными **m** вершинами. Окрашивается вершина **s** и **m** бли-жайших к ней вершина. Для каждой неокрашенной вершины **y** строятся пути, непосредственно соединяющие с помощью дуг **(x, y)** каждую окрашенную вершину **x** с неокрашенной вершиной **y**. Из полученных путей выбирается кратчайший и считается условно кратчайшим путем из вершины **s** в вершину **y**. Таким образом, определяется **(m+1)** - я ближайшая к **s** вершина. Для каждой неокрашенной вершины **x** пересчет осуществляется следующим образом:

$$\mathbf{d(x) = min\{d(x), d(y) + a(y, x)\}}. \qquad (1)$$

Начиная с $\mathbf{m = 0}$, описанная процедура может повторяться до тех пор, пока не будет получен кратчайший путь, ведущий из вершины **s** к вершине **t**.

Проанализируем вычислительную сложность алгоритма Дейкстры. На первой итерации алгоритма должны быть просмотрены **(N - 1)** неокрашенных вершин. Поскольку просмотр вершин осуществляется на основе (1), то на первой итерации выполняется **(N - 1)** операций сложения, **(N - 1)** операций сравнения, а также производится выбор наименьшего из **(N - 1)** чисел. Общее число операций в алгоритме определяется соотношением

$$\sum_{i=1}^{N} 3(N-1) = 3N(N-1)/2 \approx 1.5N^2 . \qquad (2)$$

Таким образом, вычислительная сложность алгоритма Дейкстры может быть оценена как $\mathbf{O\ (1.5N^2)}$.

Достоинства алгоритма Дейкстры:

- сравнительная простота реализации;
- невысокая полиномиальная вычислительная сложность;
- отыскивает кратчайшие пути и их величины, что является основной информацией на графовых структурах при поиске ТЗ;
- возможность машинной реализации.

Недостатки алгоритма:

- работает только на графовых структурах имеющих неотрицательные длины дуг;
- отыскивает кратчайшие пути и их величины только от одной вершины;
- для определения ТЗ между всеми парами вершин, требует многократного повтора для всех пар вершин исходного графа, что соответственно в **N** раз увеличивает вычислительные затраты в целом;
- не отыскивает рефлексивные замыкания;
- имеет достаточно сложное в сравнении с матричными методами программное представление.

Алгоритм Флойда [3]. Он позволяет находить в графе все кратчайшие пути. В качестве исходной выступает матрица $\mathbf{D^0}$ – матрица весов дуг исходного графа. Из нее вычисляется матрица $\mathbf{D^1}$. Затем по $\mathbf{D^1}$ рассчитывается матрица $\mathbf{D^2}$ и т.д. Процесс повторяется до тех пор, пока по матрице $\mathbf{D^{N-1}}$ не будет получена матрица $\mathbf{D^N}$, где **N** – число вершин в исходном графе.

Рассмотрим основную идею, лежащую в основе алгоритма Флойда. Предположим, что известны:

а) кратчайший путь из вершины **i** в вершину **m**, в котором в качестве промежуточных допускается использование только первых **(m - 1)** вершин;

б) кратчайший путь из вершины **m** в вершину **j**, в котором в качестве промежуточных допускается использование только первых **(m - 1)** вершин;

в) кратчайший путь из вершины **i** в вершину **j**, в котором в качестве промежуточных допускается использование только первых **(m - 1)** вершин.

По предположению исходный граф не может содержать контуров отрицательной длины, один из двух путей – путь совпадающий с представленным в п.в, или путь, являющийся объединением путей из пп.а, б, - должен быть кратчайшим путем из вершины **i** в вершину **j**, в котором в качестве промежуточных допускается использование только первых **m** вершин. Таким образом

$$\mathbf{d_{ij}^{m} = \min\left\{d_{im}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\right\}}. \qquad (3)$$

Из соотношения (3) видно, что для вычисления элементов матрицы $\mathbf{D^m}$ необходимо располагать лишь элементами матрицы $\mathbf{D^{m-1}}$. Более того, соответствующие вычисления могут быть проведены без обращения к исходному графу.

Проанализируем сложность алгоритма Флойда. В нем вычисляются **N** матриц $\mathbf{D^1}$, $\mathbf{D^2}$, …, $\mathbf{D^N}$, каждая из которых состоит из $\mathbf{N^2}$ элементов. Следовательно, в целом необходимо получить $\mathbf{N^3}$ элементов. Каждое такое вычисление осуществляется с помощью соотношения (3) и требует выполнение одной операции сложения и одной операции сравнения. Следовательно, в алгоритме выполняется $\mathbf{N^3}$ сложений и $\mathbf{N^3}$ сравнений. Общее количество операций, определяющих сложность алгоритма Флойда, пропорционально $\mathbf{2N^3}$.

Достоинства алгоритма Флойда:

- возможность определения наличия пути, его ранга и величины между всеми парами вершин;
- полиномиальность функции вычислительной сложности;
- возможность и сравнительная простота программной реализации.

Недостатки алгоритма:

- не допускается наличие контуров отрицательной длины;
- может определять только ТЗ между парами вершин;
- для задач большей размерности такой способ формирования кратчайших путей является весьма трудоемким;

– не определяет рефлексивные замыкания и топологию кратчайших путей.

Алгоритм Данцига [3]. Он весьма близок к алгоритму Флойда и отличается от последнего лишь иным порядком выполнения тех же самых операций. Идея алгоритма состоит в следующем:

новая вычисляемая матрица $\mathbf{D^m}$ содержит на одну строку и на один столбец больше, чем ее предшественница, матрица $\mathbf{D^{m-1}}$. Элементы матрицы $\mathbf{D^m}$, не входящие в последние столбец и строку (число таких элементов равно $\mathbf{(m-1)^2}$, представляется точно так же, как в алгоритме Флойда). Остальные элементы $\mathbf{d_{ij}^m}$, где $\mathbf{i = m}$ или $\mathbf{j = m}$ определяются с учетом приводимых ниже аргументов. Кратчайший путь из вершины **i** в вершину **m** (или наоборот), в котором допускается использование в качестве промежуточных только первых **m** вершин графа, не может иметь среди промежуточных вершину **m**, поскольку любой контур в исходном графе имеет неотрицательную длину. В силу этого такой кратчайший путь из вершины **i** в вершину **m** должен иметь своей первой частью кратчайший путь из вершины **i** в некоторую **j (j < m)**, который допускает использование в качестве промежуточных только **(m - 1)** первых вершин графа, а второй частью – кратчайшую дугу, ведущую из вершины **j** в вершину **m**.

Аналогично кратчайший путь из вершины **m** в вершину **i**, в котором допускается использование в качестве промежуточных только **m** первых вершин графа, должен иметь своей первой частью кратчайшую дугу, ведущую из вершины **m** в некоторую вершину **j (j < m)**, а второй частью – кратчайший путь из вершины **j** в вершину **i**, который допускает использование в качестве промежуточных только **(m - 1)** первых вершин.

Элементы $\mathbf{d_{ij}^m}$, где **m=1**, **2**, **…**, **N** матрицы $\mathbf{D^m}$ определяются следующим образом:

$$\mathbf{d_{mj}^m = \min_{i=1,2,...,m-1}\left\{d_{mi}^0 + d_{ij}^{m-1}\right\}, \quad j = 1,2,...,m-1;} \tag{4}$$

$$\mathbf{d_{im}^m = \min_{j=1,2,...,m-1}\left\{d_{ij}^{m-1} + d_{jm}^0\right\}, \quad i = 1,2,...,m-1;} \tag{5}$$

$$\mathbf{d_{ij}^m = \min\left\{d_{ij}^m + d_{mj}^m, d_{ij}^{m-1}\right\}, \quad i,j = 1,2,...,m-1.} \tag{6}$$

Кроме того, $\forall$ **i** и **m** необходимо положить $\mathbf{d_{ii}^m = 0}$.

Алгоритм Данцига имеет такую же вычислительную сложность, как и алгоритм Флойда – $\mathbf{O\,(2N^3)}$. Ему также присущи все недостатки и достоинства данного алгоритма.

Рассмотрим ***алгоритм*** построения ТЗ, предложенный ***Мунро*** [1].

Построение алгоритма основано на переходе от орграфа к его графу Герца. В качестве начального шага отыскивается множество сильных компонент (СК) и строится граф Герца. Затем строится ТЗ графа Герца,

которое дополняется до ТЗ исходного графа дугами, инцидентными каждой вершине внутри СК. При построении ТЗ исходного графа используется быстрое перемножение матриц (метод Штрассена).

Основная идея алгоритма Мунро состоит в следующем. Если известны ТЗ двух "смежных" подграфов с $\mathbf{2^i}$ вершинами каждый, то может быть построено ТЗ подграфа с $\mathbf{2^{i+1}}$ вершинами, получающегося объединением исходных $\mathbf{2^i}$ – вершинных подграфов.

Если имеется два подграфа, то вершина в подграфе 2 достижима из некоторой вершины подграфа 1 самое большое в три приема. Вначале необходимо попасть в некоторую вершину в подграфе 1, затем в некоторую вершину в подграфе 2 и только потом в нужную вершину из подграфа 2. Иногда первый или третий шаги или оба могут отсутствовать.

Построение ТЗ производится путем последовательного отыскания степеней матрицы с заменой на каждом шаге ненулевых элементов на 1. При этом все используемые матрицы треугольные и все вычисляемые элементы лежат в правом верхнем углу.

Вычислительная сложность алгоритма составляет $\mathbf{O\left(N_G^k \log_2 N_G\right)}$, где $\mathbf{N_G}$ – число вершин в графе Герца, а **k** берется из интервала $\mathbf{[2,\ \log_2 7]}$.

Достоинства алгоритма Мунро - невысокая вычислительная сложность.

Недостатки алгоритма:

- матрица смежности графа Герца должна обязательно быть верхней треугольной матрицей;
- используется довольно сложный метод Штрассена быстрого перемножения матриц;
- довольно сложная возможность машинной реализации
- не определяет рефлексивные замыкания.

Рассмотрим ***алгоритм*** поиска ТЗ, предложенный ***Уоршоллом*** [5].

Пусть **G** – ориентированный граф на **n** вершинах, которые обозначены целыми числами **1**, 2, … , **n**. Пусть $\mathbf{G^0 = G}$. Алгоритм Уоршолла строит последовательность таких графов, что **G** является подграфом графа $\mathbf{G^{i+1}}$, и $\mathbf{G^n}$ является ТЗ графа **G**. Граф $\mathbf{G^i}$ **(i > 1)** получается из графа $\mathbf{G^{i-1}}$ после обработки вершины **i** в графе $\mathbf{G^{i-1}}$. Обработка вершины **i** в графе $\mathbf{G^{i-1}}$ включает добавление новых ребер в $\mathbf{G^{i-1}}$ следующим образом.

Пусть в графе $\mathbf{G^{i-1}}$ присутствуют ребра **(i, k)**, **(i, l)**, **(i, m)**, …, исходящие из вершины **i**. Тогда для каждого ребра **(j, i)** заходящего в вершину **i**, добавляем в графе $\mathbf{G^{i-1}}$ ребра **(i, k)**, **(i, l)**, **(i, m)**, …, если эти ребра еще не присутствуют в графе $\mathbf{G^{i-1}}$.

Граф, получаемый после обработки вершины **i**, обозначается через $\mathbf{G^1}$. Пример работы алгоритма Уоршолла представлен на рис.1. Ясно, что $\mathbf{G^i \subseteq G^{i+1}}$ при $\mathbf{i \geq 0}$.

Сделаем ряд замечаний.

1. Алгоритм Уоршолла преобразует матрицу смежности **M** графа **G** в матрицу смежности его ТЗ соответствующим переписыванием матрицы **M**. Поэтому говорят, что алгоритм работает «на месте».

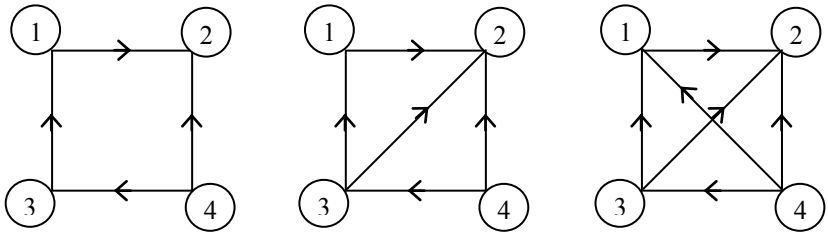

Рис.1. Работа алгоритма Уоршолла

2. Алгоритм обрабатывает все ребра, заходящие в вершину, перед тем, как начинает обрабатывать следующую вершину. Другими словами, он обрабатывает матрицу **M** по столбцам. Следовательно, описывается алгоритм Уоршолла как ориентированный по столбцам.
3. При обработке вершины никакие новые ребра (т. е. ребра, которые не присутствовали в графе, когда началась обработка данного узла), заходящие в эту вершину, не добавляются. Это означает, что при обработке вершины можно выбирать ребра, заходящие в нее в произвольном порядке.
4. Алгоритм Уоршолла называется работающим в один проход, так как каждая вершина обрабатывается точно один раз.
5. Алгоритм имеет вычислительную сложность порядка **O** ($2N^3$).

Недостаток алгоритма - алгоритм не определяет рефлексивные замыкания.

Рассмотрим ***алгоритм*** поиска ТЗ, предложенный ***Рейнгольдом*** [4].

Предполагается, что орграф **G=(N, E)** представлен матрицей смежности $\mathbf{A}=[\mathbf{a_{ij}}]$ размера $|\mathbf{N}|\times|\mathbf{N}|$, где $\mathbf{a_{ij}}=\mathbf{1}$, если имеется ребро $\mathbf{(i, j)}\in\mathbf{E}$, и $\mathbf{a_{ij}}=\mathbf{0}$ в противном случае, $\mathbf{i, j}\in\mathbf{E}$. Необходимо получить матрицу связности $\mathbf{A}^*=\left[\mathbf{a}_{\mathbf{ij}}^*\right]$, определяемую условием вида $\mathbf{a}_{\mathbf{ij}}^*=\mathbf{1}$, если в графе **G** имеется путь из вершины **i** в вершину **j**, и $\mathbf{a}_{\mathbf{ij}}^*=\mathbf{0}$, если такого пути нет. Если множество **E** ребер графа **G** рассматривать как бинарное отношение на множестве **N** вершин графа **G**, то $\mathbf{A}^*$ будет матрицей смежности для графа $\mathbf{G}^*=\left(\mathbf{N},\mathbf{E}^*\right)$, в котором $\mathbf{E}^*$ - ТЗ бинарного отношения **E**.

Матрица $\mathbf{A}^*$ формируется по матрице **A**, определяя последовательность матриц $\mathbf{A}^{(0)} = \left[\mathbf{a}_{ij}^{(0)}\right], \mathbf{A}^{(1)} = \left[\mathbf{a}_{ij}^{(1)}\right], \ldots, \mathbf{A}^{(|N|)} = \left[\mathbf{a}_{ij}^{(|N|)}\right]$, так

$$\mathbf{a}_{ij}^{(0)} = \mathbf{a}_{ij}, \mathbf{a}_{ij}^{(e)} = \mathbf{a}_{ij}^{(e-1)} \vee \left(\mathbf{a}_{ie}^{e-1} \wedge \mathbf{a}_{ej}^{e-1}\right). \quad (7)$$

Утверждается, что $\mathbf{A}^* = \mathbf{A}^{(|N|)}$. Хотя соотношение (7) полезно для понимания способа вычисления $\mathbf{A}^*$, оно неудобно для фактического выполнения вычислений. Поэтому (7) представляют в другой форме. Заметим, что если задана матрица $\mathbf{A}^{(e-1)}$, то ее можно преобразовать в $\mathbf{A}^{(e)}$ следующим образом. Если $\mathbf{a}_{ie}^{(e-1)} = \mathbf{0}$, то просто имеем $\mathbf{a}_{ij}^{(e)} = \mathbf{a}_{ij}^{(e-1)}$, и если $\mathbf{a}_{ie}^{(e-1)} = \mathbf{1}$, то имеем $\mathbf{a}_{ij}^{(e)} = \mathbf{a}_{ij}^{(e-1)} \vee \mathbf{a}_{ej}^{e-1}$. Обозначив **i** - ю строку матрицы **A** через $\mathbf{a}_{i,*}$, получают:

$$\mathbf{a}_{i,*}^{(e)} = \begin{cases} \mathbf{a}_{i,*}^{(e-1)}, & \text{если } \mathbf{a}_{ie}^{(e-1)} = \mathbf{0}; \\ \mathbf{a}_{i,*}^{(e-1)} \vee \mathbf{a}_{e,*}^{(e-1)}, & \text{если } \mathbf{a}_{ie}^{(e-1)} = \mathbf{1}. \end{cases} \quad (8)$$

Для $\mathbf{i} \neq \mathbf{1}$ значение $\mathbf{a}_{i,*}^{(e-1)}$ используется только при вычислении $\mathbf{a}_{i,*}^{(e-1)}$, поэтому его значения могут меняться, не влияя на вычисления $\mathbf{a}_{k,*}^{(e)}$ для $\mathbf{k} \neq \mathbf{i}$. Для преобразования **A** в $\mathbf{A}^*$ в алгоритме используется соотношение (8). Его вычислительная сложность равна $\mathbf{O}\left(|\mathbf{N}|^3\right)$.

Достоинства алгоритма:

- простота реализации;
- полиномиальная зависимость вычислительной сложности.

Недостатки - алгоритм не определяет рефлексивные замыкания.

Краткий сравнительный анализ алгоритмов поиска ТЗ в распределенных системах приведен в табл.1.

Как видно из вышеизложенного анализа, в настоящее время нет алгоритма, который бы на системах с распределенной сетевой структурой, одновременно строил как транзитивные, так и рефлексивные замыкания. При этом имел бы высокую производительность и невысокую вычислительную сложность. Таким образом, создание высокоэффективного алгоритма построения транзитивно - рефлексивных замыканий (ТРЗ) на объектах с сетевой организацией, в настоящее время представляется актуальной научной задачей.

Таблица 1

Сравнительный анализ алгоритмов поиска ТЗ

	Алгоритм Дейкстры	Алгоритм Флойда	Алгоритм Данцига	Алгоритм Мунро	Алгоритм Уоршолла	Алгоритм Рейнгольда
Ф-ные возможности	ТЗ	ТЗ	ТЗ	ТЗ	ТЗ	ТЗ
Вычислительная ложность	$\mathbf{O(1.5N^2)}$	$\mathbf{O(2N^3)}$	$\mathbf{O(2N^3)}$	$\mathbf{O(N_G^k \log_2 N_G)}$	$\mathbf{O(2N^3)}$	$\mathbf{O(N^3)}$
Простота реализации	+	+	+	-	+	+
Наличие контуров отрицательной длины	не допускается	не допускается	не допускается	допускается	допускается	не допускается
Необходимость многократного выполнения для всех пар вершин	+	-	-	-	-	-

ЛИТЕРАТУРА

1. Евстигнеев В.А. Применение теории графов в программировании. – М.: Наука, 1985. – 351 с.

2. Лекции по теории графов / Емеличев В.А. и др. – М.: Наука, 1990. – 384 с.

3. Minieka E., Optimization Algoritms for Networks and Graphs. marcel Dekker, Inc., New York and Basel, 1978. – 323 p.

4.Рейнгольд Э., Нивергельт Ю., Део Н., Комбинаторные алгоритмы. Теория и практика. – М.: Мир, 1980. – 602 с.

5. Свами Н., Тхуласираман К. Графы, сети и алгоритмы. – М.: Мир, 1984. – 454 с.

Поступила в редколлегию 28.08.2000