

АЛГОРИТМ ВЫЧИСЛЕНИЯ ТРАНЗИТИВНОГО ЗАМЫКАНИЯ СЕТЕВОГО ОБЪЕКТА МЕТОДОМ ДИЗЬЮНКТИВНОГО ВЛОЖЕНИЯ СТРОК МАССИВА СМЕЖНОСТИ

В.Н. Бацамут

(представил д.ф.-м.н. С.В. Смеляков)

Рассматривается полиномиальная алгоритмическая модель построения транзитивного замыкания объектов с сетевой организацией структуры.

В [1, 2] был предложен алгоритм поиска транзитивно - рефлексивного замыкания (ТРЗ) на орграфовых и неорграфовых структурах соответственно. Алгоритм базировался на построении матрицы достижимости R_G из матрицы смежности S_G исходного графа G . Там же указывалось, что такое построение может быть получено с вычислительными затратами порядка $O(N^3)$, где N - размерность G .

Алгоритм вычисления массива R_G , предлагаемый в данной статье, использует идею дизьюнктивного вложения строк исходной матрицы S_G [3]. Покажем, что такое вложение может быть выполнено на базе одной матрицы, оптимизировано с точки зрения количества выполняемых арифметических операций, что приводит к уменьшению вычислительной сложности подобного алгоритма, а также затрачиваемых аппаратных средств.

1. Оптимизация алгоритма. Алгоритм, предложенный в [3], в процессе работы использует две матрицы S^1 и S^2 размером $|N| \times |N|$. При этом в начале S^1 и S^2 являются матрицами смежности исходного графа G . Осуществляется построчный обход S^1 , и если некоторый ее $s_{ij}^1 \neq 0$, то осуществляется поэлементное дизьюнктивное вложение j -й строки в i -ую, а именно:

$$s_{ik}^2 := s_{ik}^1 \vee s_{jk}^1, \text{ где } k = \overline{1, |N|}. \quad (1)$$

Другими словами при обработке некоторой вершины обрабатываются все дуги исходящие из нее. Обработка дуги (i, j) заключается в добавлении в состав G дуги (i, k) для каждой дуги (j, k) исходящей из вершины j . Это очевидно из самой операции вложения строк. Обозначим ее через L .

Результаты анализа элементов массива S^1 заносятся в S^2 . После такого обхода производится поэлементное сравнение массивов S^1 и S^2 . При их несоответствии элементы S^2 переписываются в S^1 и обход повто-

ряется. Признаком останова алгоритма является идентичность S^1 и S^2 или же выполнение $N-1$ обходов массива S^1 . При этом $S^1=S^2=R_G$. Вычислительную сложность подобного алгоритма можно оценить порядком $O(2N^4)$, так как на каждом обходе массива S^1 осуществляется N^3 операций сложения и N^3 операций присваивания (1) и таких обходов в асимптотике необходимо сделать $N-1$ раз.

Оптимизацию на этапе дизъюнктивного вложения строк можно осуществить, если перед дизъюнктивным сложением анализировать слагаемые. Очевидно, что, если в i - й строке некоторый ее элемент $s_{ik}^1 = 1$ или в j - й строке некоторый $s_{jk}^1 = 0$, то операция (1) избыточна.

Учитывая тот факт, что операция сравнения выполняется быстрее, чем операция дизъюнкции и присваивания, то на больших N можно получить значительный выигрыш по времени выполнения алгоритма.

Результаты дизъюнктивного вложения строк будем заносить не в S^2 , а накапливать сразу в единственном массиве S^1 . Учитывая сказанное, сформулируем и докажем следующую теорему.

Теорема. Матрица достижимости R_G исходного произвольного графа G может быть получена из его матрицы смежности S^1 путем дизъюнктивного вложения соответствующих строк за m обходов текущего массива, где $m = \overline{1,2}$.

Доказательство: Покажем, что данная задача сводится к доказательству двух более простых. Для приведения доказательства рассмотрим отдельные участки некоторого графа G .

Предположим, что две произвольные вершины s и t исходного графа G транзитивно замкнуты. Значит между ними существует некоторое множество путей, соединяющих их. Тогда, используя обработку L , на первом обходе массива смежности, для любой i в G может быть добавлена некоторая дуга (i,k) . При этом возможны две ситуации: 1) $i > k$ или $k = t$; 2) $i < k$ или $k = t$.

В первом случае в графе G образуется некоторый путь $M_{st} = \{(s, i_1), (i_1, i_2), (i_2, i_3), \dots, (i_m, t)\}$, у которого $s > i_1 > i_2 > i_3 > \dots > i_m$. Например, путь

$$M_{st} = \{(4,2), (2,1)\} \text{ (рис.1).}$$

Во втором случае данный путь будет обладать свойством $s < i_1 < i_2 < i_3 < \dots < i_m$: путь

$$M_{st} = \{(1,3), (3,5)\} \text{ (рис.2).}$$

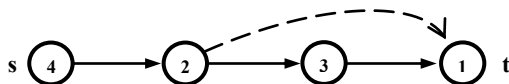


Рис.1. Образование пути с монотонно убывающими номерами вершин

Таким образом, после завершения первого обхода массива смежности G , в структуре промежуточного графа транзитивного замыкания между двумя произвольными вершинами s и t можно выделить участ-

ки пути с монотонно убывающими или возрастающими номерами вершин. Рассмотрим поведение алгоритма на таких структурах.

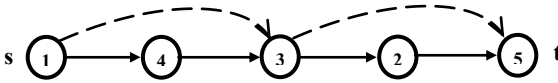


Рис.2. Образование пути с монотонно возрастающими номерами вершин

случае при обработке каждой последующей дуги будут приняты во

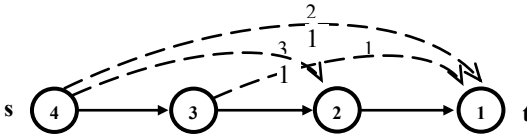


Рис.3. Построение транзитивного замыкания на участке пути с монотонно убывающими номерами вершин

обозначена очередность добавления в G^* дуг.

Во втором случае дуга $(1,2)$ будет встречаться первой (рис.4). Ее

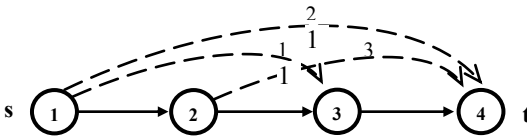


Рис.4. Построение транзитивного замыкания на участке пути с монотонно возрастающими номерами вершин

руемом массиве R_G ляжет правее от элемента отвечающего за дугу $(1,2)$, что приведет к появлению дуги $(1,4)$ и т.д.

Таким образом, в общем случае подобному алгоритму для вычисления искомого G^* требуется два обхода текущего массива. Теорема доказана.

Такой подход, основанный на обработке только одной матрицы, освобождает от необходимости осуществлять процедуру перезаписи элементов массива S^2 в S^1 , вычислительные затраты которой оцениваются, как $O(N^2)$. Это в свою очередь ускоряет предлагаемый в данной статье алгоритм. Нет необходимости также вкладывать строку саму в

Так как обработка дуг (просмотр строк текущего массива) происходит в порядке возрастания номеров вершин - истоков, то на втором обходе в первом внимании все предыдущие (рис.3). Обработка последней дуги $(4,3)$ будет учитывать как $(3,2)$, так и $(3,1)$, что добавит в структуру искомого G^* дуги $(4,2)$ и $(4,1)$ соответственно. Цифрами

обработка вызовет появление дуги $(1,3)$, так как в графе присутствует дуга $(2,3)$. Поскольку $3 > 2$, то дуга $(1,3)$ также будет обработана (соответствующий ей единичный элемент в форми-

себя $(r_{ij}^m = 1, i = j)$. На всей матрице подобные операции могут занять $O(N^2)$ вычислительных затрат. Если предположить, что один элемент массива хранится в одной ячейке памяти, то для предлагаемого алгоритма достаточно N^2 ячеек. Для алгоритма, описанного в [3], это количество в два раза больше и составляет $2N^2$.

Учитывая все выше сказанное, предлагаемый в данной статье алгоритм A1 можно реализовать в виде процедуры, представленной на рис.5.

```

procedure TForm1.TZ_Algorithm;
var i, j, jj, m: Integer;
begin
    for m:=1 to 2 do
    for i:=0 to N-1 do
    for j:=0 to N-1 do
        If (S[i,j]=1) and (i<>j) Then
            begin
                for jj:=0 to N-1 do
                    If (S[i,jj]=0) and (S[j,jj]=1) Then S[i,jj]:=1;
            end;
    end;

```

Рис.5. Программная реализация алгоритма A1

Очевидно, что сложность данного алгоритма можно оценить как $2N^3$.

2. Сравнение с известными подходами. В качестве критерия для проведения сравнительного анализа была выбрана вычислительная сложность алгоритма. Алгоритм сравнивался с известными подходами Флойда и Уоршалла, вычислительная сложность которых также ограничивается полиномом третьей степени. В [4] отмечено, что на практике обычно самым эффективным оказывается алгоритм Уоршалла, соответствующим образом запрограммированный. Там же приведены программные реализации указанных алгоритмов.

Для чистоты эксперимента отметим, что все три алгоритма были реализованы одним и тем же программистом, эксперимент для каждого алгоритма выполнялся на одном и том же компьютере с конфигурацией: Intel Celeron-500/i440BX/64Мб ОЗУ.

Схема эксперимента. Размерность тестовой задачи изменялась в диапазоне $10 \div 100$ с шагом 10. С помощью таймера отсчитывались интервалы времени T . В качестве интервала была выбрана 0.01 сек. Количество интервалов, затрачиваемое каждым алгоритмом на решение 1000

задач, заносилось в таблицу. Для формирования исходных данных было выбрано два варианта:

а) на каждом шаге использовалась одна и та же тестовая задача соответствующей размерности, решение которой заранее известно – стационарная задача;

б) для формирования исходной матрицы смежности использовался датчик случайных чисел из множества $\{0,1\}$ – динамические задачи.

Кривые были построены с помощью математического пакета MathCAD 2.52 по точкам, полученным в результате проведения исследования.

Преимущества предлагаемого алгоритма над алгоритмом Уоршалла при схеме а), на размерности $N=30$ достигает 3%, а при $N=100$ уже 15% (рис.6). Преимущества над алгоритмом Флойда в среднем составляет 48-50% на всех размерностях. При схеме б) предлагаемый подход опережает алгоритм Уоршалла в среднем на 45-60% в зависимости от размерности тестовой задачи. Примерно такие же показатели сохраняются и в сравнении с алгоритмом Флойда.

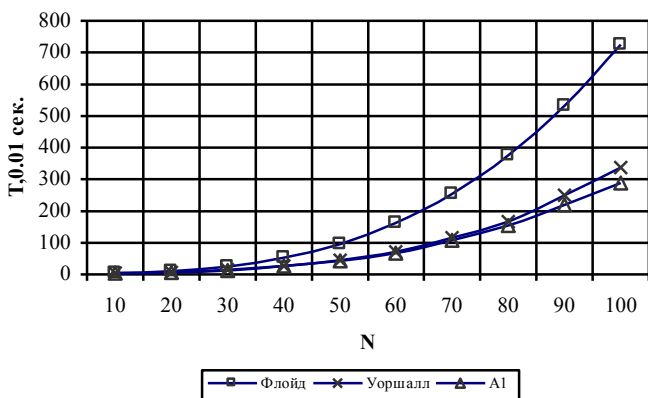


Рис.6. Рост вычислительной сложности от размерности при решении стационарной задачи

При решении динамических задач (вариант б) (рис.7) вычислительная сложность всех алгоритмов возросла. Резкий рост наблюдается у алгоритма Уоршалла. Очевидно, что при варианте б) исходные массивы были более плотными и характеризовались 45-55% содержанием единичных элементов, на анализ которых алгоритму Уоршалла приходилось затрачивать большее время [4]. В условиях а) и б) постоянством отличается алгоритм Флойда, так как он не содержит условных операторов, а значит, не анализирует значения каждого элемента массива и четко выполняет три вложенных цикла.

Рост вычислительной сложности, для различной размерности задач для алгоритма А1, при варианте б) составил 14-28% в сравнении с вариантом а), в то время как у алгоритма Уоршалла этот показатель составляет 42-60%.

Предлагаемый в данной статье алгоритм может найти свое применение в задачах динамического контроля связности сетевых объектов, а

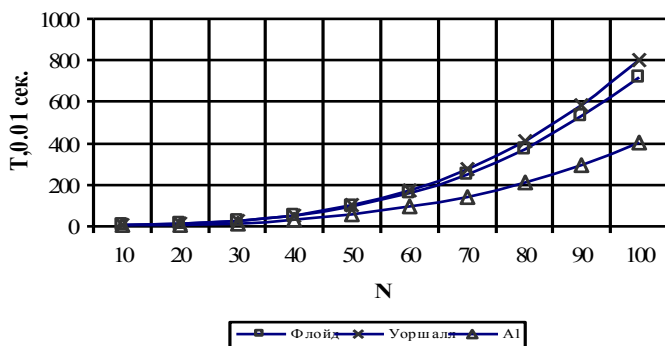


Рис. 7. Рост вычислительной сложности от размерности при решении динамических задач

также при управлении информационными потоками в них там, где время, затрачиваемое на обработку и передачу информации соизмеримо со временем выполнения управляющих алгоритмов. Понятно, что алгоритмы, реализующие подобные функции, должны обладать более низкими вычислительными затратами.

ЛИТЕРАТУРА

1. Бацамут В.Н., Спорышев К.А. Алгоритм структурно-топологического синтеза транзитивно-рефлексивных замыканий на произвольно ориентированных графах // Системи обробки інформації. – Харків: НАНУ, ПАНМ, ХВУ. – 2000. – Вип. 2(8). – С. 112 - 114.
2. Кузнецов А.В., Бацамут В.Н. Построение транзитивно-рефлексивных замыканий бинарно - унарных отношений произвольных неорграфов // Системи обробки інформації. – Харків: НАНУ, ПАНМ, ХВУ. – 1999. – Вип. 1(5). – С. 27 - 30.
3. Кузнецов А.В. Математическое моделирование объектов проектирования в САПР. – Харьков : ХВУ, 1994. – 92 с.

4. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 213 с.

Поступила в редколлегию 30.04.2001
