*Вестник МГТУ, том 14, №4, 2011 г. стр.799-810*

УДК 519.711 : 519.712

# Алгоритмы оптимизации сетевых структур на базе графов кодовых пересечений при объединении произвольных графов

**Л.Ф. Борисова**

*Судоводительский факультет МА МГТУ, кафедра радиотехники и радиотелекоммуникационных систем*

**Аннотация.** Предлагаются комбинаторная алгоритмическая модель и алгоритмы ее реализации, позволяющие строить оптимальный по критерию однородности структуры граф, объединяющий произвольное число произвольных графов, причем произвольные графы сохраняют свои конфигурации в объединенном графе. Алгоритмы разработаны на основе стратегии систематического приоритетного порождения множества конфигураций объединенных матриц специального вида, отображающих структуры графов, и эвристического поиска оптимальной с точки зрения однородности объединенной матрицы. Предложены три варианта алгоритмов реализации оптимизационной задачи, которые позволяют гибко приспосабливать решение к условиям конкретной прикладной задачи.

**Abstract.** The combinatory algorithmic model and algorithms of its realization have been proposed. The algorithms allow to build the optimum (according to the criterion of structure uniformity) graph joining arbitrary numbers of arbitrary graphs. These arbitrary graphs keep configurations in the incorporated graph. The algorithms have been worked out on the basis of strategy of regular priority generation of configurations set of joined matrixes of the special kind displaying structures of graphs, and the heuristic search optimum from the point of view of uniformity of the joined matrix.

**Ключевые слова:** граф кодовых пересечений, матрица мощностей пересечений номеров вершин графа, систематическое приоритетное порождение, эвристический поиск, алгоритм, оптимизация
**Key words:** code intersections' graph, matrix of capacities of intersections of graph tops' numbers, regular priority generation, heuristic search, algorithm, optimization

## 1. Введение

Множество разнообразных прикладных задач, использующих сетевые структуры, естественно формулируются в терминах теории графов и их матричных представлений. В реальных задачах соответствующие графы часто так велики, а расчеты так трудоемки, что получение решения неосуществимо без использования ЭВМ и методов динамического программирования. Среди множества прикладных задач, решаемых с применением теории графов, большую актуальность имеют задачи, связанные с построением макросистем на основе объединения нескольких микросистем (подсистем). Такие задачи можно рассматривать как алгоритмические на дискретных конечных математических объектах. Применение комбинаторных методов при построении оптимальных алгоритмов поиска решений на дискретных конечных математических объектах позволяет получать по сравнению с естественным подходом на основе полного перебора более эффективные решения, которые могут давать существенный выигрыш во времени работы и требуемой памяти ЭВМ. Учитывая большую актуальность алгоритмических задач, к настоящему времени разработано достаточно большое количество комбинаторных алгоритмов для различных областей применений (*Варфоломеев*, 2000; *Ноден, Ките*, 1999; *Рейнгольд и др.*, 1980). Большинство практических комбинаторных алгоритмов ориентировано на выполнение вычислений над элементарными математическими объектами – числами. Значительно реже в качестве математических объектов используются векторы, и практически нет алгоритмов, которые работают с матричными объектами. Объясняется это вычислительной сложностью таких алгоритмов, которая при работе с матрицами становится неприемлемо высокой и требует слишком большого объема памяти ЭВМ.

В настоящей работе предлагается использовать свойства графов кодовых пересечений (ГКП) для алгоритмического моделирования сложных сетей, получаемых путем объединения произвольного числа подсетей, имеющих произвольные топологии и размеры. Использование свойств ГКП для представления сетевых топологий позволяет формализовать решение топологических задач и является удобным и экономичным для компьютерной обработки. Для представления структур с помощью ГКП необходимо специальным образом закодировать номера вершин соответствующих графов. Принципиальная математическая модель объединения подграфов ГКП в единый надграф разработана в (*Борисова*, 2007). При этом задача кодирования вершин графов сводится к построению единого связного надграфа, который является порожденным подграфом графа кодовых пересечений (ПП ГКП), изоморфным заданным произвольным исходным графом, и присвоению специальным образом вершинам графов

кодированных номеров из множества номеров вершин ПП ГКП. При построении ПП ГКП используются специального типа матрицы, отображающие структуры графа, которые названы матрицами мощностей пересечений номеров вершин графа (ММП) (*Борисова*, 2007). Разработка комбинаторных алгоритмов, математическими объектами которых являются указанные матричные объекты, есть цель данной работы.

Ниже предлагаются комбинаторная алгоритмическая модель и алгоритмы ее реализации, позволяющие строить оптимальный с точки зрения однородности структуры граф, объединяющий произвольное число произвольных графов, причем произвольные графы сохраняют свои конфигурации в объединенном графе. Алгоритмы разработаны на основе стратегии систематического приоритетного порождения множества конфигураций объединенных матриц вида МПН (матриц пересечений кодированных номеров вершин ГКП (*Борисова*, 2006)), которые соответствуют заданному набору из локальных матриц вида МПН фиксированной структуры, и эвристического поиска типа "восхождение в гору" оптимальной с точки зрения максимальной однородности (в функции энтропии) объединенной МПН, удовлетворяющей критерию оптимальности при заданных ограничениях окна поиска. В общем случае стратегия поиска оптимального решения включает следующую последовательность действий:

(1) выбор локальной матрицы для сдвига, сдвиг локальной матрицы на одну позицию вдоль главной диагонали рабочей матрицы;

(2) порождение – формирование новой матрицы МПН путем объединения элементов в каждой позиции рабочей матрицы в вектор, определение параметров ПП ГКП($n$, $k$, $r$);

(3) определение значения оценочной функции и его анализ – значение оценочной функции удовлетворяет критерию оптимальности?: ДА $\rightarrow$ 5, НЕТ $\rightarrow$ 4;

(4) отбрасывание решения, $\rightarrow$ 1;

(5) принятие решения, прекращение процесса.

Алгоритмы, построенные на базе выбранной стратегии, для краткости называются алгоритмами систематического приоритетного порождения (СПП).

Разработанная комбинаторная алгоритмическая модель является базовой для построения различных алгоритмов СПП, которые позволяют получать оптимальную структуру единой сети, отличающуюся высокой структурной надежностью и экономичностью в отношении кратчайших путей в этой сети. При этом объединяемые сети сохраняют свою автономность, способы и параметры функционирования, а при технологическом взаимодействии сети работают как единое целое. Программная реализация алгоритмов достаточно компактна и не критична к системным требованиям ЭВМ и времени вычисления.

**2. Формулировка оптимизационной задачи и способы представления и хранения структур данных при реализации алгоритмов оптимизации**

Требования к единой сети при объединении произвольного числа произвольных сетей в наиболее общем случае могут быть сформулированы в следующем виде:

1) Единая сеть (ассоциация) гибко включает в себя произвольное число произвольных сетей (автономных, локальных), имеющих различные топологии и связанные с ними различные физические параметры (скорости, физические среды, виды и размеры объектов перемещения (транспортных единиц), тарифные политики и др.).

2) Локальная (автономная) сеть, входящая в состав единой сети (ассоциации), работает автономно и независимо при транспортировке объектов перемещения в пределах этой сети.

3) Транспортировка объектов перемещения из узла одной локальной сети в узел, расположенный в пределах другой локальной сети, обеспечивается по кратчайшему пути транзитом через любые другие локальные сети, входящие в состав единой сети (ассоциации).

4) Единая сеть (ассоциация) имеет единое информационное виртуальное бизнес-пространство, обеспечивающее выработку (и доставку) решений по технологическому взаимодействию различных объектов перемещения с учетом требований по надежности (или безопасности перемещения).

Оптимизационную задачу сформулируем в терминах теории графов кодовых пересечений. Используем понятия и термины теории графов (*Оре*, 1980), теории матриц (*Ланкастер*, 1982), теоретические положения графов кодовых пересечений (*Борисова*, 2006), а также способ информационно-графического отображения сетевых топологий при их объединении (*Борисова*, 2007). Задача формулируется и решается относительно ориентированных графов. Аналогичное решение для неориентированных графов можно получить, применив при нумерации вершин графов наряду с прямым зацеплением кодовых комбинаций номеров вершин также обратное зацепление (*Борисова*, 2006).

Пусть $T$ ориентированных графов $G(X_t, Y_t)$, имеющих множества вершин $X_t$, мощностями $|X_t|$, и множества дуг $Y_t$, мощностями $|Y_t|$, $t = 1, 2, \ldots T$, и не имеющих истоков и стоков, представлены своими $L_t \times L_t$ матрицами пересечений номеров вершин графов МПН $\mathrm{M}_t = \|A^t_{i,j}\|$, которые отображаются в

матрицы мощностей пересечений номеров вершин графов ММП $M_t = ||w^t_{ij}||$, $w^t_{ij} = |A^t_{ij}|$ и соответствующие идентификаторы мощности. Решаем задачу построения для любых $T$ ориентированных графов $G(X_t,Y_t)$, $t = 1, \ldots, T$, оптимального с точки зрения однородности структуры связного изоморфного надграфа $G(A,V)$ с множеством вершин $A = \bigcup_{t=1}^{T} X_t$ и множеством дуг $V \in \bigcup_{t=1}^{T} Y_t$, являющегося порожденным подграфом ГКП($n$, $k$, $r$) с параметрами: $n$ – длиной кодовой комбинации номера вершины, $k$ – основанием кода, $r$ – мощностью пересечения кодовых комбинации номеров вершин, т.е.

$$\bigcup_{t=1}^{T} G(X_t, Y_t) \bigcup \overline{V} \sim G(A, V) \cap \text{ГКП}(n, k, r),$$

где $\overline{V} = V / \bigcup_{t=1}^{T} Y_t$ – дополнение, множество дуг, соединяющих вершины непересекающихся графов, которые названы мостами, причем множество $\overline{V}$ может быть пустым.

**Задача линейного программирования.** Построить для заданного множества $T$ неоднородных и несоразмерных квадратных $L_t \times L_t$ ММП-матриц $M_t = ||w^t_{ij}||$, $i, j = 1, 2, \ldots L_t$, $t = 1, 2, \ldots T$, объединенную квадратную $L \times L$ матрицу вида ММП

$$M = || w_{ij} || = \Sigma^{T}_{t=1} M_t, \quad (1)$$
$$w_{ij} = \Sigma_{t \in T} \Sigma_{i,j \in [1,l]} w^t_{ij},$$
$$L = k \wedge (n_{opt} - r_{opt}),$$
$$r_{opt} = |\log_k W_{opt}| + |\log_k l_{opt}|, \quad n_{opt} = r_{opt} + |\log_k l_{opt}|,$$

которая соответствует псевдографу кодовых пересечений ПП ГКП($n$, $k$, $r$) с параметрами $n$ и $r$ при заданном параметре $k$ и удовлетворяет критерию оптимальности

$$W_{opt} = \max_{\substack{i,j \in [1,l] \\ l \in \Im}} w_{ij} \le W_0, \quad (2)$$
$$l_{opt} = \max_{l \in \Im} l \le L_0,$$

где $\Im$ – множество значений $l$, полученных при различных вариантах объединения $T$ графов, $\varepsilon x \upsilon$ – наименьшее целое, не меньшее $x$, "^" – операция возведения в степень, $\{W_0, L_0\}$ – окно поиска.

Размеры окна поиска определяются величинами оптимальных параметров $n$ и $r$ при заданном параметре $k$, поэтому окно должно быть как можно меньшим. Минимальные размеры окна поиска определяются в соответствии с выражениями:

$$W_0 = k \wedge \varepsilon \log_k \max_{i,j \in [1, L_l]} w_{ij} \upsilon,$$
$$L_0 = k \wedge \varepsilon \log_k \max_{t \in T} L_t \upsilon. \quad (3)$$

Варьируя размеры $W_0$ и $L_0$ окна поиска, можно изменять величины параметров $n$ и $r$ при заданном параметре $k$.

Эффективность реализации алгоритмов оптимизации в большой степени зависит от способов хранения данных в ячейках памяти ЭВМ. При объединении локальных матриц в единую рабочую матрицу возникают две проблемы:

1) эффективное размещение элементов различных типов матриц в ячейках памяти;

2) гибкое обращение к памяти при реализации динамических процедур включения и исключения элементов рабочей матрицы в ходе выполнения операции сдвига.

Минимизировать требуемый объем оперативной памяти при хранении сильно разреженных матриц позволяют специальные структуры хранения массивов данных – дуплеты и триплеты.

*Дуплеты* используют 2 информационных поля I и J для хранения пары индексов ($i$, $j$) по горизонтали и вертикали матрицы смежности, которые определяют позицию элемента матрицы со значением, равным "1". Линейный список индексов дуплетов хорошо приспособлен для хранения матриц смежностей исходных графов, элементами которых являются нули и единицы.

*Триплеты* предназначены для хранения матриц, элементами которых являются произвольные значения. Триплеты содержат 3 информационных поля: поля I и J для хранения пары индексов ($i$ = I, $j$ = J), матрицы по горизонтали и вертикали, определяющие позиции с ненулевыми значениями, а также поле ID (Идентификатор вершины) для хранения значений ячеек матрицы в этих позициях. Этот способ хорошо приспособлен для хранения ММП – матриц мощностей пересечений кодированных номеров вершин ГКП исходных графов, элементами которых являются произвольные значения (*Борисова*, 2007).

Другой проблемой при работе с рабочей матрицей в ходе оптимизации ее структуры является необходимость выполнения на каждом шаге процедуры оптимизации включения и исключения элементов объединяемых векторов в общий рабочий вектор при реализации сдвига. Удобство выполнения этих операций зависит от способа размещения элементов векторов в памяти.

Простейшим вариантом хранения элементов в памяти является *последовательное распределение* (*Рейнгольд и др.*, 1980). При нем порядок следования элементов в векторе задается неявно требованием, чтобы смежные элементы последовательности находились в смежных ячейках памяти. В результате элементы последовательности во время включения или исключения должны передвигаться. Учитывая, что включения и исключения элементов в рабочей матрице выполняются на каждом шаге процедуры оптимизации, последовательное распределение при большом числе объединяемых матриц представляет существенную трудность.

Обойтись без сдвига элементов в ячейках памяти при их включении и исключении позволяет использование *связанного распределения* (*Рейнгольд и др.*, 1980; *Трубецкой*, 2009). Связанный список – это разновидность линейных структур данных, представляющая собой последовательность элементов, обычно отсортированную в соответствии с заданным правилом. Последовательность может содержать любое количество элементов, поскольку при создании списка используется динамическое распределение памяти. В этом случае информация о способе упорядочения последовательности элементов хранится явным образом. В общем случае при представлении последовательности $s_1, s_2, \ldots, s_n$ в виде связанного списка каждый элемент списка состоит из поля INFO, содержащего элемент $s_i = \mathrm{INFO}(l_i)$ последовательности, и поля LINK, содержащего указатель $P_i = \mathrm{LINK}(l_i)$, отмечающий ячейку, в которой записаны $s_{i+1}$ и $P_{i+1}$ следующего элемента. Указатель $P_0$ указывает начальную ячейку с символами $s_1$ и $P_1$, $P_n = \mathrm{LINK}(l_n) = \Lambda$, где $\Lambda$ – пустой или нулевой указатель. Связанное представление локальных векторов облегчает операции включения элемента после некоторого элемента $s_i$ и исключения элемента $s_{i+1}$, если ячейка для элемента $s_i$ известна. Для этого необходимо лишь изменить значения некоторых указателей. Так же легко осуществляется сцепление последовательностей и разбиение последовательности на подпоследовательности. Использование связанных распределений позволяет добиться большей гибкости при реализации процедуры оптимизации объединенной матрицы системы, хотя приходится тратить память на указатели $P_i$.

Еще большая гибкость достигается при использовании *дважды связанного (двусвязного) списка*, в котором каждый элемент $s_i$ имеет два связанных с ним указателя (*Рейнгольд и др.*, 1980; *Трубецкой*, 2009). Неудобство включения и исключения элементов локальных векторов в рабочий вектор в ходе объединения локальных матриц при использовании последовательного распределения требует хранения элементов векторов в виде связанного списка. Они указывают на элементы $s_{i+1}$ и $s_{i-1}$. В таком списке для любого элемента имеется мгновенный прямой доступ к предыдущему и последующему элементам. Благодаря этому облегчаются операции включения нового элемента перед некоторым элементом $s_i$ и исключение элемента $s_i$ без предварительного знания его предшественника.

Связанный список, в котором память для элементов выделяется динамически, называется *динамическим*. Динамический массив способен изменять свой размер. Оболочка для данного массива отвечает за выделение и освобождение памяти под массив, а также обеспечивает доступ к элементам массива. Когда пользователь создает объект класса-оболочки, конструктор класса выделяет память под массив, который имеет либо указанный пользователем размер, либо размер, заданный по умолчанию. Если по мере заполнения массива вся выделенная память окажется занятой, то при добавлении очередного элемента выделенная ранее память освобождается, все хранящиеся в массиве значения сохраняются во временном массиве. Затем выделяется память под массив большего размера, и в него помещаются сохраненные значения. Таким образом, изменение размера массива происходит автоматически, невидимо для пользователя.

В настоящее время подобные динамические структуры данных (контейнеры) в большом ассортименте представлены в стандартной библиотеке шаблонов C++ – Standard template library (STL), которая поддерживается практически всеми компиляторами. Наиболее типичным примером является класс vector, который содержит все необходимые функции и итераторы. Также используются следующие классы-контейнеры:

map – ассоциативный массив, который содержит пары "ключ – значение" и обеспечивает доступ к значению по ключу;

multymap – ассоциативный массив, в котором могут встречаться одинаковые по значению ключи;

queue – очередь, т.е. массив, организованный по принципу FIFO ("first in first out");

deque – очень напоминает класс vector, он также как и vector, поддерживает произвольный доступ к элементам, но не поддерживает некоторые функции, которые присутствуют в классе vector;

list – список, который не поддерживает доступ к элементу по индексу, вместо этого осуществляется поиск элемента по значению;

set – множество (или простой ассоциативный массив), которое отличается от ассоциативного массива тем, что в нем ключ является одновременно и значением (имеет интерфейс, напоминающий интерфейс map, что позволяет безболезненно их чередовать);

multyset – множество, в котором могут встречаться одинаковые по значению ключи.

При разработке и анализе алгоритмов СПП используем понятия и термины теории комбинаторного счисления (*Виленкин*, 1969) и комбинаторных алгоритмов (*Рейнгольд и др.*, 1980). Порождение различных матричных объектов производим путем сдвига отдельных локальных матриц вдоль главной диагонали общей рабочей матрицы с последующим объединением на каждом шаге процедуры сдвига *ij*-ых элементов локальных матриц в единый вектор, находящихся в *ij*-ой позиции рабочей матрицы ММП. Для обеспечения линейности и эффективности алгоритма СПП сдвиг локальных матриц производим только в одном направлении (вниз вдоль главной диагонали рабочей матрицы ММП) и только на одну позицию рабочей матрицы ММП. Локальные матрицы ММП ранжируем по размеру и для сдвига выбираем ту из них, которая имеет минимальный размер $L_t$.

В соответствии с принятой стратегией возможны два подхода к решению оптимизационной задачи:

1) вначале все локальные матрицы объединяются в единую рабочую матрицу вида ММП, затем на каждом шаге процедуры оптимизации выполняется систематическое комбинаторное порождение объединенной матрицы ММП путем сдвига одной из локальных матриц ММП, после этого выполняется проверка соответствия новой конфигурации матрицы ММП условиям оптимальности;

2) объединенная матрица вида ММП строится путем добавления к рабочей матрице на каждом шаге процедуры оптимизации одной из локальных матриц ММП из списка в некотором систематическом комбинаторном порядке, а затем выполняется проверка соответствия новой порожденной конфигурации ММП условиям оптимальности.

В результате структура суммарной рабочей матрицы в обоих вариантах объединения постепенно приближается к некоторой конфигурации ММП, удовлетворяющей критерию оптимальности. Выбор той или иной локальной матрицы для сдвига при соблюдении принятых правил оптимизации неоднозначен, т.к. условию сдвига могут удовлетворять одновременно несколько локальных матриц из числа возможных, поэтому найденное оптимальное решение не обязательно является глобальным оптимумом. Целью оптимизации является получение конфигурации ММП, для которой значение оценочной функции удовлетворяет критерию оптимальности в заданном окне поиска.

Выбор того или другого подхода к объединению зависит от числа объединяемых матриц и их размеров. Первый подход теоретически может дать решение за один шаг, но при большом количестве локальных матриц и их размеров этот вариант может оказаться излишне расточительным в отношении памяти вычислителя. Второй подход более экономичен, однако требует больше времени для объединения всех локальных матриц.

В соответствии с первым подходом разработан алгоритм "линейный список", который обеспечивает объединение локальных матриц по типу "звезда". На втором подходе основаны два других алгоритма – "расширенное бинарное дерево, минимальное по ширине" (конфигурация типа "шина") и "расширенное бинарное дерево, минимальное по высоте" (конфигурация типа "дерево"). Выбор в пользу того или иного алгоритма связан с компромиссом между различными желаемыми свойствами. В общем случае, чем лучше алгоритм с одной точки зрения, тем он хуже с другой. Выбор того или иного алгоритма из трех зависит от условий конкретной задачи.

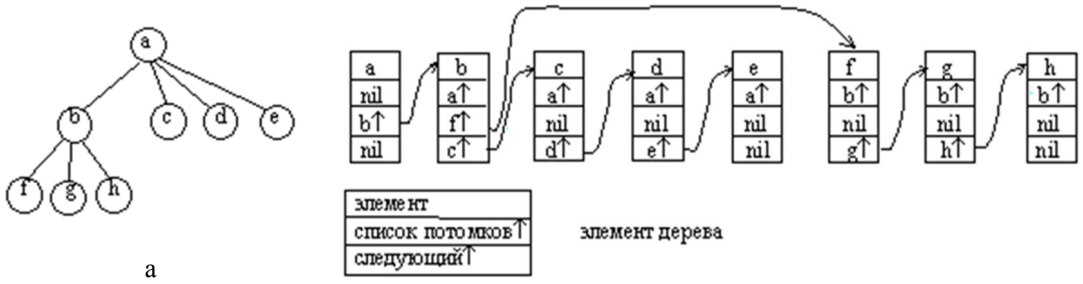
**3. Разработка алгоритма оптимизации "линейный список"**

Алгоритм "линейный список (звезда)" использует поисковую таблицу, построенную прямым объединением всех локальных ММП, и теоретически получает результат за одну операцию (рис. 1). Структура этого алгоритма по схеме объединения листьев в корневой узел напоминает по форме звезду (рис. 1а). Систематический приоритетный поиск оптимального решения осуществляется по дереву состояний (рис. 2), листьями которого являются объединяемые локальные матрицы. Для представления таких деревьев используются элементы, аналогичные элементам списковой структуры бинарного дерева. Элемент такой структуры содержит минимум три поля: значение узла, указатель на начало списка потомков узла, указатель на следующий элемент в списке потомков текущего уровня. Также необходимо хранить указатель на корень дерева. При этом дерево представляется в виде структуры, связывающей списки потомков различных вершин. Представление деревьев с произвольной структурой в виде массивов основано на матричных способах описания графов.

Дерево состояний в алгоритме "линейный список" является сильноветвящимся. Причем число потомков различных узлов не ограничено и заранее не известно. Дерево состояний строится путем сортировки с пошаговым сдвигом листьев, объединенных в корневой вершине и хранящихся в поисковой таблице. Элементы рабочей матрицы, образующие поисковую таблицу, определяются последовательным суммированием элементов локальных матриц, находящихся в одинаковых позициях рабочей матрицы, с одновременной проверкой каждой суммы по критерию оптимальности. Для ускорения процедуры поиска оптимума матрица для сдвига выбирается на приоритетной основе – матрица с меньшими размерами имеет более высокий приоритет.


а б

Рис. 1. Алгоритм "линейный список": а – схема объединения локальных ММП в рабочую матрицу; б – структура рабочей ММП




а б

Рис. 2. Схема систематического приоритетного поиска по алгоритму "линейный список": а – дерево состояний рабочей ММП; б – представление сильноветвящегося дерева в виде списка

Процедура оптимизации состоит из следующих последовательных этапов. (1) Находится первый элемент рабочей матрицы, полученный путем суммирования, который не удовлетворяет условию оптимальности. (2) Среди всех матриц, элементы которых входят в найденную сумму, выбирается локальная матрица для сдвига. (3) Выбранная локальная матрица сдвигается на одну позицию вниз вдоль главной диагонали рабочей матрицы. Далее процедура повторяется сначала.

Процедура суммирования продолжается, начиная с той позиции рабочей матрицы, которая определяется позицией первого элемента сдвинутой локальной матрицы до выполнения сдвига и заканчивается в позиции последнего элемента сдвинутой локальной матрицы после выполнения сдвига. Процедура оптимизации заканчивается, когда все элементы рабочей матрицы удовлетворяют условию оптимальности.

Таким образом, поиск направлен на определение нераскрытой вершины дерева, которую следует раскрыть первой, чтобы уменьшить общее число перебираемых вариантов. Раскрыть вершину – значит построить все возможные переходы из данного состояния. При этом для каждой вершины определяется значение оценочной функции $W_{i,j}$, и раскрывается та вершина, которая имеет минимальное значение. Оценочная функция $W_{i,j} = \{w^t_{i,j}, L_t, \mathrm{H}\{p_{i,j}\}\}$ показывает количество ресурса, необходимого для достижения финиша.

**Пример.** Пусть формируется единая ММП путем объединения матриц ММП 1 – ММП 7 (рис. 3). Пусть на некотором шаге процедуры формирования рабочей матрицы при суммировании $(i, j)$-ых элементов матриц ММП 1, ММП 2, ММП 5, ММП 6, ММП 7 получено максимальное значение элемента рабочей матрицы в позиции, помеченной на рисунке круговой линией. Пусть проверка показала, что условие (2) не выполняется:

$$\max w\{t = [1, 2, 5, 7]\} = \max_{i,j \in [1, L_0]} w_{ij} = \sum_{t=1,2,5.7} w^t_{ij} > W_0.$$

Тогда в соответствии с процедурой оптимизации требуется уменьшить значение этого элемента. Для уменьшения данного суммарного элемента выбираем среди матриц ММП 1, ММП 2, ММП 5, ММП 6, ММП 7, элементы которых составляют сумму $\max w\{\mathrm{t} = [1, 2, 5, 6, 7]\}$ матрицу ММП 7, имеющую минимальные размеры.

Сместив эту матрицу вдоль главной диагонали рабочей матрицы вниз на одну позицию, можно добиться уменьшения суммы $\max w_{i,j}$ за счет вычитания из нее элемента $w^7_{i,j}$. Последовательное повторение этой процедуры ведет к уравновешиванию рабочей матрицы и максимизации функции ее энтропии. Процедура поиска оптимальной матрицы в заданных границах $L_0 \times L_0$ рабочей матрицы заканчивается, когда получена рабочая матрица, все элементы которой удовлетворяют условию (2) или когда все локальные матрицы достигнут нижней границы рабочей матрицы.

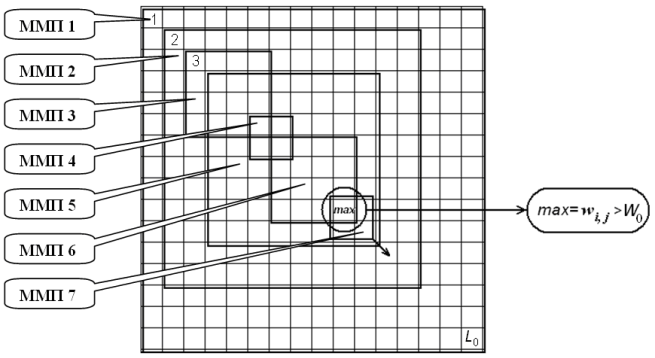



Рис. 3. Схема процедуры сдвига матриц при максимизации функции энтропии

Если в результате сдвигов локальных матриц все они достигли границы рабочей матрицы, а условие (2) не выполнено, то выбирается один из двух возможных вариантов изменения размеров окна поиска:

1) увеличивается значение параметра $L_0$, и процедура поиска продолжается;
2) увеличивается значение параметра $W_0$, и процедура поиска начинается сначала.

Процедура выбора локальной матрицы с минимальным значением оценочной функции (по числу диагональных элементов) может быть проведена одним из методов частичной сортировки (*Кнут*, 2007; *Кормен и др.*, 2006; *Седжвик*, 2003), например, типа турнира с выбыванием. Имеется более совершенный алгоритм, который принято называть пирамидальной сортировкой (Heapsort) (*Кузнецов*, 2009; *Трубецкой*, 2009). Его идея состоит в том, что вместо полного дерева сравнения исходный массив $a[1]$, $a[2]$, ..., $a[n]$ преобразуется в пирамиду, обладающую тем свойством, что для каждого $a[i]$ выполняются условия $a[i] <= a[2i]$ и $a[i] <= a[2i+1]$. Затем пирамида используется для сортировки.

Несмотря на кажущуюся сложность, алгоритм "линейный список" является самым быстрым алгоритмом объединения локальных матриц из всех возможных алгоритмов, так как он, используя поисковую таблицу, в принципе может получить оптимальный результат за одну операцию. Платой за такую скорость является расточительное использование объема памяти. Алгоритм требует хранения одновременно всех объединяемых локальных матриц ММП, что при больших значениях $L_t$ и $T$ делает алгоритм неприемлемым для практического использования. Однако, если число локальных матриц не слишком велико, размеры их не слишком большие, а мощности вычислительного средства достаточно, то алгоритм "линейный список" может быть весьма эффективен.

**4. Разработка алгоритма оптимизации "расширенное бинарное дерево, минимальное по ширине"**

Стратегия оптимизации "линейный список" требует значительного объема памяти ЭВМ для реализации и может быть применена при ограниченном числе объединяемых матриц, помещающихся целиком в основной памяти. Снизить требования к объему основной памяти позволяют алгоритмы, построенные на основе бинарных деревьев и использующие методы сортировки со слиянием. Такой подход к оптимизации информационной структуры снимает ограничение в требовании размещения исходных локальных матриц в основной памяти и позволяет использовать внешнюю память.

Каждая вершина *бинарного (двоичного) дерева (binary tree)* имеет не более двух поддеревьев. Причем для каждого узла выполняется правило, в соответствии с которым в левом поддереве содержатся только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве содержатся только ключи, имеющие значения, большие, чем значение данного узла. Двоичное дерево состоит из узлов (вершин) – записей вида (data, l, r), где data – некоторые данные, привязанные к узлу, l, r – ссылки на узлы, являющиеся детьми данного узла. Узел l называется левым ребёнком (сыном), а узел r – правым. Бинарное дерево является рекурсивной структурой, поскольку каждое его поддерево само является бинарным деревом и, следовательно, каждый его узел в свою очередь является корнем дерева. Существует следующее рекурсивное определение двоичного дерева:

<дерево> ::= <данные> <дерево> <дерево> | null.

Таким образом, двоичное дерево либо является пустым, либо состоит из данных и двух поддеревьев (каждое из которых может быть пустым). Каждое поддерево, в свою очередь, тоже является деревом. Если у некоторого узла оба поддерева пустые, то он называется листовым узлом (листовой вершиной). Бинарное дерево также может представлять собой пустое множество. Расширенное бинарное дерево получается путем присоединения к бинарному дереву внешних узлов (листьев) (рис. 4а).

Бинарные деревья достаточно просто могут быть представлены в виде списков или массивов. Списочное представление позволяет представить и хранить "расширенное бинарное дерево, минимальное по ширине". Списочное представление бинарных деревьев основано на использовании элементов, соответствующих узлам дерева. Каждый элемент имеет поле данных и два поля указателей. Один указатель используется для связывания элемента с правым потомком, а другой указатель – с левым. Листья имеют пустые указатели потомков.

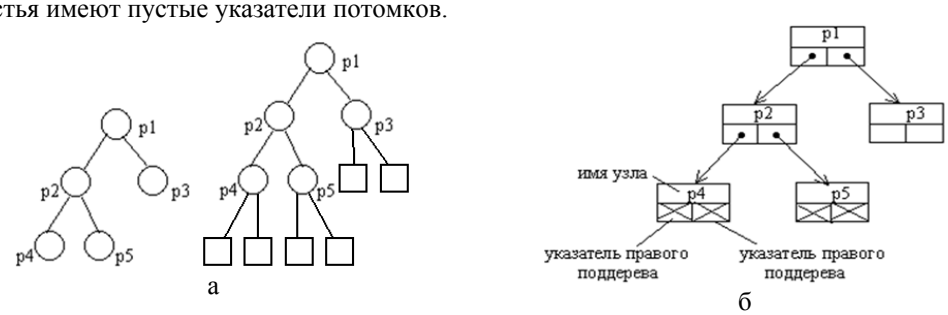



Рис. 4. Представление бинарного дерева в виде списковой структуры:
а – структуры бинарного дерева и расширенного бинарного дерева;
б – дерево состояний поисковой таблицы

Указатель на узел, являющийся корнем дерева, следует обязательно сохранять (рис. 4б). Такой способ представления бинарного дерева имеет сходство с простым линейным списком, разновидностью мультисписка, образованного комбинацией множества линейных списков, в нем каждый линейный список объединяет узлы, входящие в путь от корня дерева к одному из листьев.

Структура типа "расширенное бинарное дерево, минимальное по ширине", подобна "шине" – это дерево, у которого внутренние узлы имеют в качестве потомка не более одного листа, и только последний узел в качестве потомков имеет два листа. При построении такого дерева исходные МПН помещаются в дерево в качестве листьев. Главное отличие алгоритма "расширенное бинарное дерево, минимальное по ширине" от алгоритма "линейный список" состоит в том, что исходные МПН (листья) в "шине" объединяются не все сразу, как в "линейном списке", а постепенно, путем добавления по одной локальной матрице к суммарной (рабочей) матрице. В этом случае определение максимума оценочной функции для $T$ массивов можно свести к рекуррентной процедуре определения максимума для каждой пары из $T$.

Листья в дереве должны быть отсортированы по критерию размерности МПН от минимальной к максимальной. Для этого можно использовать различные алгоритмы сортировки (*Кнут*, 2007; *Кормен и др.*, 2006; *Седжвик*, 2003). Причем, в случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, является *ключом сортировки*. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Алгоритмы сортировки оцениваются по скорости выполнения (вычислительной или временной сложности) и эффективности использования памяти (емкостной сложности). *Время* – основной параметр, характеризующий быстродействие алгоритма. Пусть на вход алгоритму подается множество $A$, обозначим $n = | A |$. Для алгоритма $A$ функция *временной сложности* $A(n)$ дает верхнюю границу для максимального времени его работы, т.е. максимальное число основных операций (сложения, сравнения и т.д.), которые должен выполнить алгоритм $A$ при решении задачи на входных данных размером $n$. Для упорядочения важны *худшая*, *средняя* и *лучшая* скорость (вычислительная сложность) алгоритма в терминах мощности входного множества $A$. Типичный алгоритм имеет хорошую скорость $O(n \log n)$, плохую – $O(n^2)$, идеальную скорость для упорядочения – $O(n)$. Алгоритмы сортировки, использующие только абстрактную операцию сравнения ключей, всегда нуждаются по меньшей мере в $O(n \log n)$ сравнениях. Существуют более эффективные алгоритмы, например, алгоритм сортировки Хана (Yijie Han) с вычислительной сложностью $O(n \log \log n \ \log \log \log n)$, использующий тот факт, что пространство ключей ограничено, однако он чрезвычайно сложен, а за O-обозначением скрывается весьма большой коэффициент, что делает невозможным его применение в повседневной практике. Также существует понятие сортирующих сетей. Предполагая, что можно одновременно (например, при параллельном вычислении) проводить несколько сравнений, можно отсортировать $n$ чисел за $O(n \log^2 n)$ операций. При этом число $n$ должно быть заранее известно. Временная (вычислительная) сложность некоторых алгоритмов сортировки исследуется в работах (*Левитин*, 2006; *Кнут*, 2007).

*Память* характеризует функцию *емкостной сложности* алгоритма $A(n)$, дающую границу для максимального числа одновременно существующих скалярных значений при выполнении $A$ на входных данных размером $n$. Ряд алгоритмов требует выделения дополнительной памяти под временное хранение

данных. Как правило, эти алгоритмы требуют O(log $n$) памяти. При оценке не учитывается место, которое занимает исходный массив, и независящие от входной последовательности затраты, например, на хранение кода программы (так как всё это потребляет O(1)).

В простейшем случае реализации алгоритма "расширенное бинарное дерево, минимальное по ширине" при использовании пузырьковой сортировки вначале проверяются размерности двух соседних листьев (МПН) и, если предыдущий лист больше последующего, то листья меняются местами. Затем начинается процесс формирования рабочей матрицы путем объединения двух соседних листьев в дереве. Для этого вначале выбираются параметры $\{W_0, L_0\}$ окна поиска, которые задают размерность $L_0$ рабочей матрицы и вес (мощность) $W_0$ ячейки рабочей матрицы. Заметим, что величины параметров $W_0$ и $L_0$ кратны степени $k$. Далее начинается процедура вложения матриц МПН в рабочую матрицу. Эта процедура продолжается до тех пор, пока вес какой-либо ячейки рабочей матрицы не достигнет (или не превысит) своего максимума – $W_0$. Если вложенная матрица не вышла за пределы рабочей матрицы, т.е. если не превышен размер $L_0$, то производится диагональное смещение текущей МПН и процесс вложения повторяется заново. Если размер $L_0$ превышен в результате сдвига, то производится корректировка окна поиска. Параметры окна поиска в процессе выполнения программы могут изменяться динамически, т.к. их начальные значения не обязательно оптимальны.

Алгоритм, построенный в форме бинарного дерева решений, позволяет резко сократить требования к объему памяти. В простейшем случае бинарное дерево решений представляет собой линейный список с указателями массивов ММП, который просматривается последовательно на каждом шаге итерации. Организация данных с помощью бинарных деревьев часто позволяет значительно сократить время поиска нужного элемента. Поиск элемента в линейных структурах данных обычно осуществляется путем последовательного перебора всех элементов, присутствующих в данной структуре. Поиск по дереву не требует перебора всех элементов, поэтому занимает значительно меньше времени. Максимальное число шагов при поиске по дереву равно высоте данного дерева, т.е. количеству уровней в иерархической структуре дерева.

Оценим сложность алгоритма "расширенное бинарное дерево, минимальное по ширине" в зависимости от $T$ – числа объединяемых ММП. В работе (*Рейнгольд и др.*, 1980) показано, что высота дерева решений применительно к числовым объектам может соответствовать числу решений, требуемых алгоритмом в худшем случае, а среднее число сравнений, требуемых алгоритмом для решения задачи, соответствует сумме уровней, взятых по всем исходам, поделенной на число исходов. Если считать в идеальном случае все $T$ ММП равными и однородными, то применительно к матричным объектам имеем

$$E(T)/(N+1),$$

где $N$ – число внутренних узлов, равное $T$-1.

В линейном алгоритме высота дерева равна $T$ – числу объединяемых ММП, а диапазон значений функции $E(T)$ определяется выражением

$$E(T) = 0{,}5(T-1)(T+2).$$

Тогда среднее расстояние до внешнего узла определяется выражением

$$E(T) / T = 0{,}5\,[(T-1)(T+2)] / T.$$

Следовательно, средняя сложность данного алгоритма растет линейно с ростом $T$.

Таким образом, к достоинствам данного алгоритма по сравнению с алгоритмом "линейный список" относятся следующие:

- простота и компактность – объем памяти, требуемый для реализации алгоритма, пропорционален числу объединяемых графов;
- хорошая скорость получения решения – с ростом числа $T$ средняя скорость получения решения растет не быстрее $T$/2, т.е.

$$E(T) / T = 0{,}5\,[(T-1)(T+2)] / T = T/2 + \mathrm{O}(1) \text{ при } T \to \infty,$$

где выражение O( ) означает асимптотическое соотношение между функциями $f(x)$ и $g(x)$, такое, что $f(x)$ = O($g(x)$) при $x \to x_0$, если и только если существует константа $c$, такая, что

$$\lim_{x \to x_0} \sup |f(x) / g(x)| = c,$$

в этом случае $f(x)$ растет не быстрее $g(x)$.

Алгоритм "расширенное бинарное дерево, минимальное по ширине" является самым простым в реализации из всех трех алгоритмов бинарных деревьев. Однако используемое в линейном алгоритме бинарное дерево не является оптимальным, по форме оно напоминает шину и имеет максимальную высоту среди всех деревьев.

**5. Разработка алгоритма оптимизации "расширенное бинарное дерево, минимальное по высоте"**

Расширенное бинарное дерево, минимальное по высоте (остовное дерево) получается из линейного бинарного дерева путем перестановки внешних узлов по схеме, изображенной на рис. 5. Построенное таким образом расширенное бинарное дерево сбалансировано по высоте и является оптимальным по длине внешних путей. Подобная модификация бинарного дерева сохраняет число внешних и внутренних узлов, но уменьшает высоту дерева и длину внешних путей. Оценим высоту дерева при объединении $T$ ММП.

В работе (*Рейнгольд и др.*, 1980) доказано, что минимальная длина внешних путей расширенного бинарного дерева с $n$ внутренними узлами равна

$$l = (n + 1)\log_2(n + 1) + (n + 1)(2 - Q - 2^{1-Q}),$$
$$Q = \log_2(n + 1) - \lfloor \log_2(n + 1) \rfloor,\ 0 \le Q < 1.$$

Учитываем, что число внутренних узлов дерева равно $T$-1. Тогда минимальная длина внешних путей, а, следовательно, и высота расширенного сбалансированного по высоте бинарного дерева, построенного в результате объединения $T$ локальных матриц, составляет:

$$h = T\log_2 T + T(2 - Q - 2^{1-Q}),\ 0 \le \log_2 T - \varepsilon\log_2 T\upsilon < 1.$$

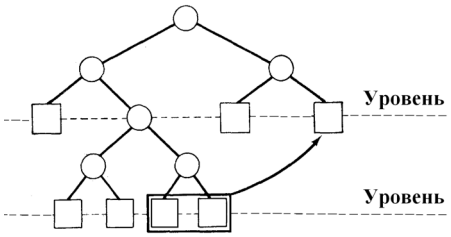



Рис. 5. Схема уменьшения длины внешних путей

Высота дерева решений соответствует числу решений, требуемых алгоритмом в худшем случае, следовательно, расширенное бинарное дерево, минимальное по высоте, имеет самый короткий путь для получения результата среди всех рассматриваемых в данной работе бинарных деревьев.

Расширенные бинарные деревья, минимальные по высоте, удобно представлять в виде массивов (рис. 6). Проще всего представляется полное бинарное дерево, так как оно всегда имеет строго определенное число вершин на каждом уровне. Вершины можно пронумеровать слева направо последовательно по уровням и использовать эти номера в качестве индексов в одномерном массиве. Если число уровней дерева в процессе обработки не будет существенно изменяться, то способ представления полного бинарного дерева значительно экономичнее, чем любая списковая структура.

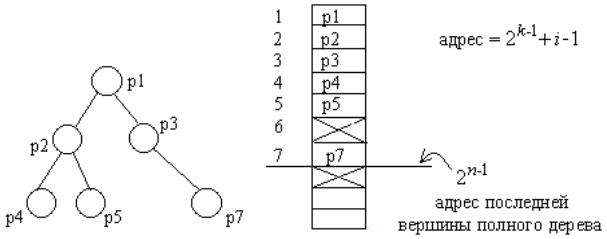



Рис. 6. Представление бинарного дерева в виде массива

Однако далеко не все бинарные деревья являются полными. Для неполных бинарных деревьев применяют следующий способ представления. Бинарное дерево дополняется до полного дерева, вершины последовательно нумеруются. В массив заносятся только те вершины, которые были в исходном неполном дереве. При таком представлении элемент массива выделяется независимо от того, будет ли он содержать узел исходного дерева. Следовательно, необходимо отметить неиспользуемые элементы массива. Это можно сделать путем занесения специального значения в соответствующие элементы массива. В результате структура дерева переносится в одномерный массив. Адрес любой вершины в массиве вычисляется как

$$\text{адрес} = 2k - 1 + i - 1,$$

где $k$ – номер уровня вершины, $i$ – номер на уровне $k$ в полном бинарном дереве. Адрес корня равен единице.

Для любой вершины можно вычислить адреса $L$ левого и $R$ правого потомков соответственно

$$\text{адрес\_}L = 2k + 2(i - 1);$$
$$\text{адрес\_}R = 2k + 2(i - 1) + 1.$$

Недостатком рассмотренного способа представления бинарного дерева является то, что структура данных является статической. Размер массива выбирается исходя из максимально возможного количества уровней бинарного дерева. Причем чем менее полным является дерево, тем менее рационально используется память.

Существуют несколько способов прохождения бинарных деревьев. Под прохождением бинарного дерева понимают определенный порядок обхода всех вершин дерева. Прямой порядок прохождения бинарного дерева можно определить следующим образом (рис. 7а): (1) попасть в корень; (2) пройти в прямом порядке левое поддерево; (3) пройти в прямом порядке правое поддерево. Прохождение бинарного дерева в обратном порядке можно определить в аналогичной форме (рис. 7б): (1) пройти в обратном порядке левое поддерево; (2) пройти в обратном порядке правое поддерево; (3) попасть в корень. Симметричный порядок прохождения бинарного дерева, можно определить в следующем виде (рис. 7в): (1) пройти в симметричном порядке левое поддерево; (2) попасть в корень; (3) пройти в симметричном порядке правое поддерево. Порядок обхода бинарного дерева можно хранить непосредственно в структуре данных. Для этого достаточно ввести дополнительное поле указателя в элементе списковой структуры и хранить в нем указатель на вершину, следующую за данной вершиной при обходе дерева. Представление деревьев в виде массивов также допускает хранение порядка прохождения дерева. Для этого вводится дополнительный массив, в который записывается адрес вершины в основном массиве, следующей за данной вершиной (рис. 7в). Такие структуры данных получили название прошитых бинарных деревьев. Указатели или адреса, определяющие порядок обхода, называют нитями. При этом в соответствии с порядком прохождения вершин различают право прошитые, лево прошитые и симметрично прошитые бинарные деревья.

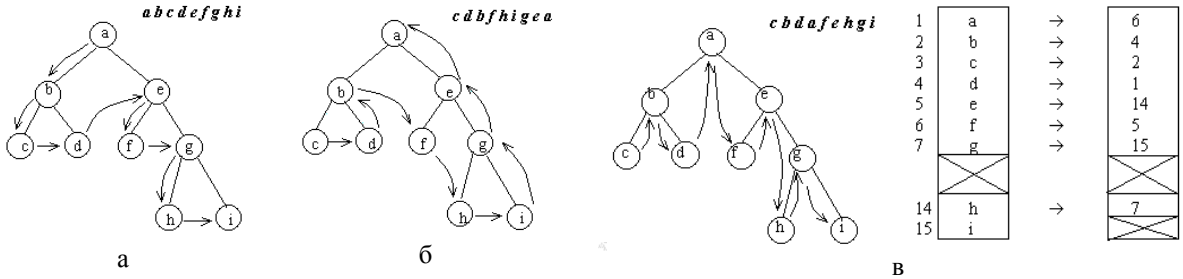



Рис. 7. Прохождение бинарного дерева: а – в прямом порядке; б – в обратном порядке; в – симметричное прошитое бинарное дерево и его представление в виде массивов

Таким образом, алгоритм "расширенное бинарное дерево, минимальное по высоте" обеспечивает достаточно быстрое получение результата, даже для больших значений $L$ и $T$. Более сложная реализация по сравнению с алгоритмом "расширенное бинарное дерево, минимальное по ширине", компенсируется более высокой скоростью.

## 6. Сравнительная характеристика и практические рекомендации по выбору алгоритмов оптимизации

В зависимости от масштабов решаемой задачи, требований к программной сложности, скорости и эффективности можно использовать один из трех разработанных алгоритмов построения оптимальной объединенной матрицы ММП, отображающей исходные графы.

Алгоритм "линейный список" (звезда) использует поисковую таблицу, построенную прямым объединением всех локальных ММП, и теоретически получает результат за одну операцию. Это самый быстрый алгоритм объединения локальных матриц из всех алгоритмов, однако он самый расточительный в отношении использования объема памяти. Алгоритм "линейный список" эффективен, если число локальных матриц не слишком велико, размеры их не слишком большие, а вычислительное средство имеет достаточно мощности.

Снизить требования к объему основной памяти позволяют алгоритмы, построенные на основе бинарных деревьев и использующие методы сортировки со слиянием. Такой подход к оптимизации информационной структуры позволяет использовать внешнюю память. Организация данных с помощью бинарных деревьев часто позволяет значительно сократить время поиска нужного элемента.

Алгоритм "расширенное бинарное дерево, минимальное по ширине" (шина) является более сложным в реализации и требует больше времени для получения результирующей объединенной матрицы, чем "линейный список". Однако он самый простой в реализации среди всех алгоритмов и не слишком критичный в отношении числа и размеров локальных матриц.

Алгоритм "расширенное бинарное дерево, минимальное по высоте" (остовное дерево) обеспечивает достаточно быстрое получение результата, даже при больших значениях размеров и большом количестве локальных матриц. Более сложная реализация по сравнению с алгоритмом "расширенное бинарное дерево, минимальное по ширине" компенсируется более высокой скоростью.

"Расширенное бинарное дерево, минимальное по высоте" имеет самый короткий путь для получения результата среди всех бинарных деревьев. Время для получения результирующей объединенной матрицы требуется больше, чем в "линейном списке", но меньше, чем в "расширенном бинарном дереве, минимальном по ширине". В реализации алгоритм "расширенное бинарное дерево, минимальное по высоте" более сложен, чем "линейный список" и "расширенное бинарное дерево, минимальное по ширине".

**7. Заключение**

1. Разработанные методы и алгоритмы реализации систематического приоритетного порождения: алгоритм "линейный список" со схемой объединения листьев в корневой узел типа "звезда", алгоритм "расширенное бинарное дерево, минимальное по ширине", реализующий схему объединения листьев в корневой узел типа "шина", и алгоритм "расширенное бинарное дерево, минимальное по высоте", имеющий схему объединения листьев в корневой узел типа "остовное дерево", – позволяют эффективно строить оптимальные сетевые структуры при объединении различных произвольных графов, причем в качестве математических объектов используются матрицы. Предложенные три варианта алгоритмов реализации оптимизационной задачи позволяют гибко приспосабливать решение к условиям конкретной прикладной задачи.

2. Комбинаторные алгоритмы используют в качестве математических объектов матрицы специального вида. Обоснованные способы оптимизации представления и хранения структур данных при реализации алгоритмов систематического приоритетного порождения позволяют минимизировать время вычисления и объем требуемой памяти вычислителя.

3. Выработанные практические рекомендации по выбору алгоритмов оптимизации учитывают масштабы решаемой задачи, требования к программной сложности, скорости и эффективности. При этом алгоритм "линейный список" является самым быстрым, но расточительным в отношении памяти, практически он эффективен, если число локальных ММП не слишком велико, размеры их не слишком большие, а вычислительное средство имеет достаточно мощности. Алгоритм "расширенное бинарное дерево, минимальное по ширине" – самый простой в реализации среди всех алгоритмов бинарных деревьев, практически он эффективен при использовании достаточно большого числа локальных ММП, имеющих средние или не слишком большие размеры. Алгоритм "расширенное бинарное дерево, минимальное по высоте" является самым сложным в реализации и обеспечивает достаточно быстрое получение результата, даже при больших значениях размеров и большом количестве локальных ММП.

**Литература**

**Борисова Л.Ф.** Прикладные вопросы теории графов кодовых пересечений. *Вестник МГТУ*, т.9, № 2, с.291-300, 2006.

**Борисова Л.Ф.** Метод информационно-графического отображения сетевых топологий для решения транспортных задач. *Вестник МГТУ*, т.10, № 4, с.581-589, 2007.

**Варфоломеев В.И.** Алгоритмическое моделирование элементов экономических систем. Практикум. *М., Финансы и статистика*, 208 с., 2000.

**Виленкин Н.Я.** Комбинаторика. *М., Наука, гл. редакция физ.-мат. литературы*, 328 с., 1969.

**Кнут Д.** Искусство программирования. *М., Вильямс*, 1824 с., 2007.

**Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л., Штайн К.** Алгоритмы: построение и анализ. *М., Вильямс*, 1296 с., 2006.

**Кузнецов С.Д.** Методы сортировки и поиска. *ИСП РАН, Центр Информационных Технологий*: *http://www.citforum.ru/programming/theory/sorting/sorting1.shtml*, 2009.

**Ланкастер П.** Теория матриц. *М., Наука*, 336 с., 1982.

**Левитин А.В.** Алгоритмы: введение в разработку и анализ. Метод грубой силы: Пузырьковая сортировка. *М., Вильямс*, с.144-146, 2006.

**Ноден П., Ките К.** Алгебраическая алгоритмика. *М., Мир*, 720 с., 1999.

**Оре О.** Теория графов. *М., Наука*, 336 с., 1980.

**Рейнгольд Э., Нивергельт Ю., Део Н.** Комбинаторные алгоритмы. Теория и практика. *М., Мир*, 476 с., 1980.

**Седжвик Р.** Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск. *СПб., ДиаСофтЮП*, с.672, 2003.

**Трубецкой А.** С++. Программирование. URL: *http://trubetskoy1.narod.ru/index.htm*, 2009.