МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

DOI 10.17223/20710410/16/5

УДК 519.178

FPT-АЛГОРИТМЫ НА ГРАФАХ ОГРАНИЧЕННОЙ ДРЕВОВИДНОЙ ШИРИНЫ

В. В. Быкова

Институт математики Сибирского федерального университета, г. Красноярск

E-mail: bykvalen@mail.ru

Исследован специальный метод конструирования FPT-алгоритмов — метод динамического программирования на основе дерева декомпозиции. Выявлены проблемы, ограничивающие применение этого метода на практике. Предложено проблему памяти решать с помощью бинарного сепараторного дерева декомпозиции, снижающего теоретические и реальные размеры таблиц динамического программирования. Табличная техника описана на языке реляционной алгебры.

Ключевые слова: *алгоритмы на графах, дерево декомпозиции, динамическое программирование.*

Введение

Параметризированный подход к оценке сложности вычислений — естественный способ справиться с трудноразрешимостью задач, которые охарактеризованы как NP-трудные в соответствии с классической дихотомией **P** против **NP** [1]. Основная идея параметризированного подхода состоит в том, чтобы с помощью некоторого числового параметра учесть структуру исходных данных и выделить основной источник трудноразрешимости задачи. Параметризированный подход позволяет исследовать различные параметризации одной и той же задачи, каждая из которых может приводить или не приводить к FPT-алгоритмам. Такие алгоритмы представляют интерес для алгоритмической практики, так как при ограниченном значении параметра время их выполнения полиномиально зависит от длины входа алгоритма [2].

В настоящее время уже известны некоторые специальные методы проектирования FPT-алгоритмов. Наиболее известный из них — метод динамического программирования на основе дерева декомпозиции, автором которого считается Г. Бодлаендер [3]. Этот метод ориентирован на класс задач, решение которых ищется в конечной области. При этом поиск решения подразумевает нахождение одного из допустимых решений (в задачах разрешения и удовлетворения ограничений) или оптимального решения (в задачах комбинаторной оптимизации). Такие задачи в литературе часто называют задачами выбора [4]. Параметризация задачи в данном случае осуществляется по древовидной ширине входного графа. Несмотря на теоретическую привлекательность метода динамического программирования по дереву декомпозиции, практическое его применение ограничивается двумя проблемами, первая из них связана с высокими потребностями в памяти получаемых FPT-алгоритмов, а вторая — с вычислением древовидной ширины и построением дерева декомпозиции. В данной работе предложены

средства решения первой из указанных проблем. Методы решения второй проблемы рассмотрены в работе автора [5].

1. Дерево декомпозиции и древовидная ширина графа

Динамическое программирование по дереву декомпозиции — это сочетание декомпозиционного подхода к решению задачи выбора с декомпозиционным представлением входного графа, когда выделение подзадач и построение их решений осуществляется исходя из этого представления. Дерево декомпозиции графа — это специальная конструкция, которая описывает структуру графа «с точностью до клик и сепараторов» и определяется следующим образом. Пусть задан связный граф $G = (V, E)$, $n = |V| \geqslant 1$ и $m = |E| \geqslant 1$. Дерево декомпозиции графа G — пара (M, T), задающая разбиение множества вершин и множества рёбер этого графа, где $M = \{X_i : i \in I\}$ — семейство подмножеств множества V, называемых «мешками», а $T = (I, W)$ — дерево, узлам которого сопоставлены эти «мешки». Вершины дерева T принято называть узлами, чтобы избежать путаницы с вершинами графа G. Семейство $M = \{X_i : i \in I\}$ и множество W рёбер дерева $T = (I, W)$ такие, что справедливы следующие условия [3]:

1) объединение всех подмножеств, образующих «мешки», даёт множество V;
2) для всякого ребра графа G существует хотя бы один «мешок», содержащий обе вершины этого ребра;
3) для любой вершины v графа G множество узлов $\{i \in I : v \in X_i\}$, «мешки» которых содержат эту вершину, индуцирует связный подграф, являющийся поддеревом дерева T.

Ширина дерева декомпозиции (M, T) равна $\max\limits_{i \in I}\{|X_i| - 1\}$. Древовидная ширина (*treewidth*) графа G определяется как наименьшая ширина всех допустимых его деревьев декомпозиции и обозначается через $\mathrm{tw}(G)$. Дерево декомпозиции ширины $\mathrm{tw}(G)$ называется оптимальным, а без кратных и вложенных «мешков» — фундаментальным. В фундаментальном дереве декомпозиции всегда $\mathrm{O}(n)$ узлов и к нему можно перейти за время $\mathrm{O}(|I|)$. Для $\mathrm{tw}(G)$ верны естественные границы: $1 \leqslant \mathrm{tw}(G) \leqslant n-1$. Считается, что граф G обладает ограниченной древовидной шириной, если $\mathrm{tw}(G) \leqslant k$ и k есть целая положительная константа, не зависящая от n.

Заметим, что всякое дерево декомпозиции (M, T) графа $G = (V, E)$ есть не что иное, как дерево соединений ациклического гиперграфа $H = (V, M)$, рёбрами которого выступают «мешки» этого дерева [6]. При этом граф смежности вершин гиперграфа H является некоторой (не обязательно минимальной) триангуляцией графа G.

2. Описание метода

Пусть задана некоторая параметризированная задача П, которую надо решить для графа $G = (V, E)$. Полагаем, что в качестве параметра задачи взята древовидная ширина $\mathrm{tw}(G)$ и $\mathrm{tw}(G) \leqslant k$, k — целая положительная константа. Кроме того, считаем, что задача П сформулирована в оптимизационной конструктивной постановке, то есть задана некоторая числовая характеристика (целевая функция) графа $G = (V, E)$, которую надо оптимизировать и указать решение, отвечающее оптимальному значению этой характеристики.

Пусть для исходного графа $G = (V, E)$ известно корневое дерево декомпозиции (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, ширины k и с узлом r в роли корня. Семейство подзадач в данном случае можно задать следующим образом. Определим для всякого узла $i \in I$ множество вершин:

$$Y_i = \{v \in X_j : j = i \text{ или } j \text{ — потомок для } i \text{ в } T\}. \quad (1)$$

Множество Y_i индуцирует в G подграф $G_i = G(Y_i)$, а в T — поддерево с корневым узлом i. Примечательно, что $Y_r = V$ и $G_r = G$. Тогда в качестве подзадачи Π_i можно рассматривать решение задачи Π применительно к G_i, $i \in I$. Понятно, что для каждой конкретной задачи Π специфичны характеристика решения и рекуррентные процедуры, связывающие решения подзадач. Между тем сценарий действия алгоритмов, основанных на динамическом программировании по дереву декомпозиции, общий.

Работа подобных алгоритмов включает в себя два этапа. На первом этапе для исходного графа $G = (V, E)$ строится корневое дерево декомпозиции (M, T) ширины k. На втором этапе решается задача Π с помощью (M, T): сначала обход всех узлов дерева T снизу вверх (от листьев к корню r) с целью вычисления необходимой информации и нахождения значений характеристик подзадач; затем обход всех узлов дерева T сверху вниз (от корня r к листьям) с целью конструирования оптимального решения задачи Π для исходного графа G. При этом вся нужная информация вычисляется и хранится в виде таблиц. Каждому узлу $i \in I$ дерева T соответствует таблица A_i, которая содержит информацию по задаче Π_i.

Процесс формирования таблиц и решений отвечает следующим необходимым требованиям. При любом $i \in I$ решение задачи Π_i находится исключительно из таблицы A_i. Для всякого листа $i \in I$ дерева T соответствующая таблица вычисляется непосредственно из $G(X_i)$. Для каждого внутреннего узла $i \in I$ таблица создается из информации о $G(X_i)$ и таблиц, которые отвечают прямым потомкам узла i в T. Дерево декомпозиции гарантирует выполнение указанных требований, поскольку справедливо

Утверждение 1 [3]. Пусть $i \in I$ — произвольный внутренний узел дерева T. Вершины графа $G_i = G(Y_i)$, которые смежны с вершинами, находящимися вне G_i, обязательно содержатся в «мешке» X_i (и таких вершин не более $k + 1$). Иными словами, графы $G(Y_i)$ и $G(V \backslash Y_i)$ «связаны» между собой в G только с помощью вершин из X_i.

В общем случае поиск точного решения задачи Π_i для каждого узла i дерева T осуществляется полным перебором в A_i, где представляются различные подмножества множества вершин как претенденты на оптимальное или допустимое решение. Очевидно, что время подъёма и спуска по дереву декомпозиции зависит от числа и размера создаваемых таблиц динамического программирования, а также от времени обработки каждой строки такой таблицы.

3. Достаточные условия FPT-разрешимости относительно древовидной ширины

Оказывается, что описанный выше метод динамического программирования на основе дерева декомпозиции приводит к FPT-алгоритму относительно древовидной ширины, если задача выбора может быть сформулирована в монадической логике второго порядка. Монадическая логика второго порядка (*Monadic Second Order Logic*, или MSOL) для графа $G = (V, E)$ — язык исчисления предикатов выражения его свойств. Предикатные формулы в MSOL, описывающие свойства графа G, состоят из следующих символов [7]:

— предметных переменных v_i и e_j, при этом каждой переменной v_i соответствует отдельная вершина, а переменной e_j — отдельное ребро графа G ($1 \leqslant i \leqslant n = |V|$, $1 \leqslant j \leqslant m = |E|$);

— предметных переменных V_i и E_j, где переменной V_i отвечает некоторое подмножество множества вершин V, а переменной E_j — некоторое подмножество множества рёбер графа G ($0 \leqslant i \leqslant 2^n$, $0 \leqslant j \leqslant 2^m$);
— предикатных переменных и логических связок $\&$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$;
— кванторов $\forall$ и $\exists$, запятых и круглых скобок.

В состав атомарных формул входят:

— предикат совпадения ($x = x'$). Принимает значение «истина», если вершины (рёбра) x и x' графа G совпадают;
— предикат принадлежности ($x \in X$), где предметная переменная x отвечает вершине (ребру) графа G, а переменная X — подмножеству вершин (рёбер) этого графа;
— предикат инцидентности $\mathrm{Incident}(v_i, e_j)$. Принимает значение «истина», если вершина v_i и ребро e_j инцидентны в G.

Все другие формулы MSOL формируются из атомарных формул MSOL с помощью логических связок, кванторов $\exists x$ и $\forall x$ (x — предметная переменная, соответствующая вершине или ребру графа G) и кванторов $\exists X$ и $\forall X$ (X — предметная переменная, отвечающая подмножеству вершин или ребер графа G). Заметим, что монадическая логика первого порядка (*Monadic First Order Logic*, или MFOL) отличается от MSOL лишь тем, что в ней не допускаются кванторы $\exists X$ и $\forall X$.

Всякое свойство графа G может быть задано некоторым предикатом P. Предикат P для графа G (обозначается через $P(G)$) принимает значение «истина», если рассматриваемое свойство выполняется для этого графа, и «ложь» в противном случае. Некоторые популярные свойства графа, выраженные в виде формул MSOL, представлены в таблице. Из всех указанных в этой таблице свойств только свойства связности и гамильтоновости графа не могут быть выражены в MFOL. Заметим, что длина всех формул, приведённых в таблице, не зависит от размера (числа вершин и рёбер) графа G. Такие формулы MSOL называются конечными.

Пусть граф G имеет ограниченную древовидную ширину, то есть $\mathrm{tw}(G) \leqslant k$, где k — некоторая целая положительная константа. Пусть проверяемое свойство графа G представлено в виде MSOL-формулы P и $|P|$ — длина этой формулы.

Теорема 1 (теорема Курселя [7]). Для графа $G = (V, E)$ с $\mathrm{tw}(G) \leqslant k$ существует функция $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ и алгоритм, который проверяет истинность $P(G)$ за время $\mathrm{O}(n \cdot f(|P|, k))$.

Очевидно, что если величина $|P|$ не зависит от $n = |V|$ и $m = |E|$, например, является постоянной или зависит только от k, то по теореме Курселя для графа, удовлетворяющего условиям теоремы 1, существует алгоритм со временем работы $\mathrm{O}(n \cdot f(k))$, то есть FPT-алгоритм относительно древовидной ширины графа G. Таким образом, всякое свойство графа $G = (V, E)$, выраженное в виде конечной формулы MSOL, может быть проверено за полиномиальное время при условии, что граф G имеет ограниченную древовидную ширину. Теорема Курселя демонстрирует широту класса FPT-разрешимых графовых задач, но не даёт точного ответа, как устроен FPT-алгоритм для задачи, отвечающей условиям этой теоремы. Единственное, что известно, — это общая схема такого алгоритма, описанная выше в п. 2.

Для применения теоремы Курселя на практике необходимо: найти MSOL-формулу проверяемого свойства графа; убедиться, что заданный граф G имеет древовидную ширину $\mathrm{tw}(G) \leqslant k$; построить для G дерево декомпозиции ширины k. Относительно последних действий известен следующий результат.

Формулы MSOL, определяющие некоторые свойства графа

Свойство графа	Предикат	Формула MSOL, определяющая предикат
Совпадение рёбер e_i и e_j	$e_i = e_j$	$(\forall v_1)(\text{Incident}(v_1, e_i)) \Leftrightarrow (\text{Incident}(v_1, e_j))$
Смежность вершин v_i и v_j	$\text{Adjacent}(v_i, v_j)$	$\neg(v_i = v_j) \,\&\, (\exists e_1)\ (\text{Incident}(v_i, e_1)\ \&$ $\&\ \text{Incident}(v_j, e_1))$
V_1 — независимое множество вершин	$\text{IndependentSet}(V_1)$	$(\forall v_2)(\forall v_3)((v_2 \in V_1 \,\&\, v_3 \in V_1) \Rightarrow$ $\neg\text{Adjacent}(v_2, v_3))$
V_1 — вершинное покрытие	$\text{VertexCover}(V_1)$	$(\forall e_2)(\exists v_3)(v_3 \in V_1 \,\&\ \text{Incident}(v_3, e_2))$
V_1 — клика	$\text{Clique}(V_1)$	$(\forall v_2)(\forall v_3)((v_2 \in V_1 \,\&\, v_3 \in V_1) \Rightarrow$ $\text{Adjacent}(v_2, v_3))$
V_1 — доминирующее множество	$\text{DominatingSet}(V_1)$	$(\forall v_2)(v_2 \in V_1 \vee (\exists v_3)(v_3 \in V_1 \,\&$ $\&\ \text{Adjacent}(v_2, v_3)))$
Вершинная k-раскраска	$\text{VertexColorable}(V_1, \ldots, V_k)$	$(\forall v_0)(v_0 \in V_1 \vee \ldots \vee v_0 \in V_k) \,\&$ $\text{IndependentSet}(V_1) \,\&\, \ldots$ $\&\, \text{IndependentSet}(V_k)$
E_1 — связность	$\text{Connected}(E_1)$	$(\forall V_2)(\forall V_3)(\neg(\exists v_4)(v_4 \in V_2)) \vee$ $(\neg(\exists v_5)(v_5 \in V_3)) \vee$ $(\exists v_6)(\neg(v_6 \in V_2) \,\&\, \neg(v_6 \in V_3)) \vee$ $(\exists e_7)(\exists v_8)(\exists v_9)(e_7 \in E_1 \,\&\, v_8 \in V_2 \,\&$ $v_9 \in V_3 \,\&\ \text{Incident}(v_8, e_7) \,\&$ $\text{Incident}(v_9, e_7))$
E_1 — гамильтонов цикл	$\text{HamCycle}(E_1)$	$\text{Connected}(E_1) \,\&\, (\forall v_2)(\exists e_3)(\exists e_4)$ $(e_3 \in E_1 \,\&\, e_4 \in E_1 \,\&\, \neg(e_3 = e_4) \,\&$ $\text{Incident}(v_2, e_3) \,\&\ \text{Incident}(v_2, e_4) \,\&$ $(\forall e_5)((e_5 \in E_1 \,\&\, \text{Incident}(v_2, e_5)) \Rightarrow$ $(e_5 = e_3 \vee e_5 = e_4)))$

Теорема 2 (Г. Бодлаендер, Т. Клокс [8]). Существует функция $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ и алгоритм, который за время $O(n \cdot f(k))$ проверяет, имеет ли граф G древовидную ширину $\text{tw}(G) \leqslant k$, и если да, то строит для G дерево декомпозиции ширины k.

Данная теорема свидетельствует о том, что условия применимости теоремы Курселя, касающиеся древовидной ширины графа, могут быть проверены соответствующим FPT-алгоритмом за полиномиальное время.

4. Вычислительная сложность метода и её преодоление с помощью B&S-дерева

Динамическое программирование — метод, рекурсивный по своей природе. Однако рекурсивная реализация данного метода при числе подзадач больше единицы неминуемо влечёт неполиномиальное время вычислений относительно длины входных данных [9]. Согласно соотношению (1), число подзадач на каждом шаге динамического программирования по дереву декомпозиции определяется арностью данного дерева. Очевидно, что в общем случае эта арность больше единицы. Поэтому для получения FPT-алгоритма необходимо динамическое программирование осуществлять итерационно с помощью табличной техники. Между тем эта техника порождает проблему памяти для размещения создаваемых таблиц, поскольку размер всякой таблицы экспоненциально зависит от арности и древовидной ширины применяемого дерева декомпозиции.

Пусть для задачи Π, решаемой на графе $G = (V, E)$ с $\text{tw}(G) \leqslant k$, построено корневое дерево декомпозиции (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, ширины k и с узлом r в роли корня. Если дерево T содержит $|I| = O(n)$ узлов, то при динамическом программировании придется хранить $O(n)$ таблиц A_i, $i \in I$. Предположим, что всякая вершина

из X_i может находиться по отношению к возможному решению в q состояниях. Например, в задаче о вершинном покрытии $q = 2$ («принадлежит вершинному покрытию», «не принадлежит вершинному покрытию»), а в задаче о доминирующем множестве $q = 3$ («входит в доминирующее множество», «не входит в доминирующее множество, но доминируется», «не входит в доминирующее множество и не доминируется»). Тогда таблица A_i узла i, являющегося листом в T, имеет размер $\mathrm{O}(kq^k)$, так как $|X_i| \leqslant k+1$. Размер таблицы A_i для внутреннего узла i с двумя прямыми потомками может достигать $\mathrm{O}(kq^{3k})$. Ясно, что чем больше арность дерева T (число прямых потомков для узлов этого дерева) и чем больше ширина (M, T), тем большего размера таблицы могут возникать в процессе вычислений и тем больше времени потребуется для их обработки. Таким образом, проблема памяти — серьезное препятствие практического применения FPT-алгоритмов, основанных на динамическом программировании по дереву декомпозиции. Эта проблема является предметом теоретических исследований многих авторов [3, 8, 10, 11]. В работах практической направленности преимущественно предлагаются различные улучшения дерева декомпозиции. Наиболее известное в этом отношении — «хорошее» дерево декомпозиции (*nice tree decomposition*), описанное Т. Клоксом [12]. Его принято называть деревом декомпозиции Kloks-вида (или просто Kloks-деревом).

Корневое дерево декомпозиции (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, отвечает Kloks-виду, если оно удовлетворяет следующим условиям:

1) каждый узел дерева имеет не более двух узлов прямых потомков;
2) если узлу i непосредственно подчинены два узла l и j, то $X_i = X_l = X_j$ (i — узел-соединение);
3) если узел i располагает одним прямым потомком j, то
   - либо $|X_i| = |X_j| + 1$ и $X_j \subset X_i$ (i — узел-вставка),
   - либо $|X_i| = |X_j| - 1$ и $X_i \subset X_j$ (i — узел-удаление).

В работе [12] доказано, что всякое фундаментальное дерево декомпозиции (M, T), имеющее ширину k и $\mathrm{O}(n)$ узлов, может быть преобразовано за время $\mathrm{O}(k^2 n)$ в Kloks-дерево, которое обладает той же шириной и $\mathrm{O}(kn)$ узлами. Заметим, что мощности «мешков» Kloks-дерева, соответствующие смежным узлам этого дерева, различаются не более чем на единицу. Такая сглаженность и бинарность Kloks-дерева даёт возможность размер создаваемых таблиц удерживать в пределах оценки $\mathrm{O}(kq^k)$. Понятно, что это достигается за счет увеличения (возможно в k раз) числа узлов дерева и таблиц динамического программирования.

Желательно иметь такой вид дерева декомпозиции, который бы обеспечивал компромисс между размером и числом таблиц динамического программирования. Для достижения этой цели предлагается использовать бинарное сепараторное дерево декомпозиции (или просто B&S-дерево). Для него характерны следующие особенности: переход от бинарного корневого дерева декомпозиции к B&S-дереву увеличивает число таблиц не более чем в два раза; оно, как и Kloks-дерево, позволяет удерживать размер всякой таблицы динамического программирования в пределах оценки $\mathrm{O}(kq^k)$; для вычисления таблиц динамического программирования по B&S-дереву возможно привлечение аппарата реляционной алгебры и свойств ациклических баз данных. Дадим пояснение и обоснование сказанному.

Корневое дерево декомпозиции (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, графа G назовём B&S-деревом, если в нём каждый узел имеет не более двух прямых потомков и относится к одному из четырёх типов узлов:

1) узел-лист — узел, у которого нет потомков;
2) узел-сепаратор — узел s с одним прямым потомком j и узлом i в роли родителя. Всегда X_s — сепаратор графа G и $X_s = X_i \cap X_j \neq \varnothing$, $X_s \subseteq X_i$, $X_s \subseteq X_j$;
3) узел-расширение — узел i с одним прямым потомком s. В данном случае $X_s \subseteq X_i$;
4) узел-соединение — узел i с двумя прямыми потомками l и j. Здесь $X_l \cup X_j \subseteq X_i$.

Утверждение 2. Всякое фундаментальное дерево декомпозиции (M, T) графа G, где $M = \{X_i : i \in I\}$, $T = (I, W)$, имеющее ширину k и $|I| = \mathrm{O}(n)$ узлов, может быть преобразовано за время $\mathrm{O}(n)$ в B&S-дерево, которое обладает той же шириной и $\mathrm{O}(n)$ узлами.

Доказательство. Преобразование заданного дерева декомпозиции (M, T) в B&S-дерево можно выполнить за два шага.

Ш а г 1. Сначала выбрать любой узел $r \in I$ в роли корня. Затем снизить арность дерева до двух следующим образом. Если внутренний узел $i \in I$ обладает одним или двумя прямыми потомками, то ничего не делать. Если внутренний узел $i \in I$ имеет в качестве прямых потомков узлы $j_1, \ldots, j_d$, $d \geqslant 3$, то выполнить «клонирование» узла i в узлы $i_1, \ldots, i_{d-1}$ и приписать каждому из них «мешок» X_i. Отношение подчинённости между узлами установить так, как показано на рис. 1. В результате будет добавлено $\mathrm{O}(n)$ узлов.

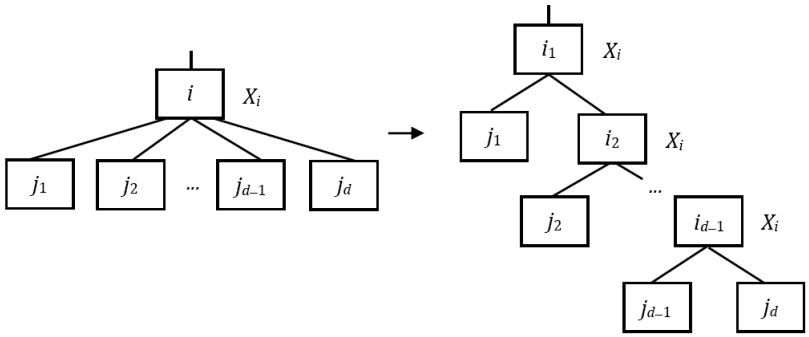

Рис. 1. Схема преобразования дерева декомпозиции к бинарному виду

Ш а г 2. Для любых двух узлов $i, j \in I$, смежных в T и имеющих «мешки» X_i и X_j соответственно, добавить промежуточный узел s и сопоставить ему в качестве «мешка» множество вершин $X_s = X_i \cap X_j \neq \varnothing$ (рис. 2). Исходя из известных свойств дерева декомпозиции [5, с. 67, утверждение 1], множество X_s — сепаратор графа G и, следовательно, s есть узел-сепаратор. Таких узлов будет добавлено $\mathrm{O}(n)$.

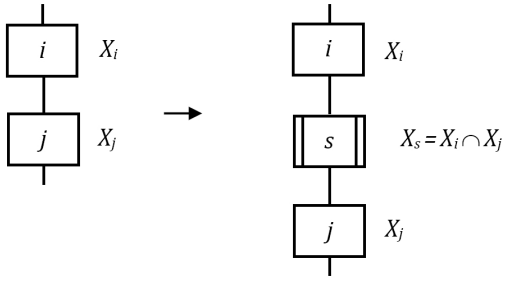

Рис. 2. Схема формирования узла-сепаратора

Общее число узлов в B&S-дереве составит $O(n)$. Время реализации каждого шага — также $O(n)$. Ни одно из действий, предусмотренных шагами 1 и 2, не увеличивает вместимость «мешков» для (M, T), поэтому значение k не изменится. Построенное B&S-дерево — дерево декомпозиции графа G, так как оно очевидным образом отвечает условиям 1–3 определения дерева декомпозиции. ■

Чтобы убедиться, что B&S-дерево сохраняет размер всех создаваемых в процессе динамического программирования таблиц в пределах оценки $O(kq^k)$, сначала рассмотрим, как это достигается при применении Kloks-дерева. Здесь целесообразно прибегнуть к языку реляционной алгебры (алгебры таблиц) [6].

Пусть (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, — некоторое корневое бинарное дерево декомпозиции графа $G = (V, E)$ ширины k. Решение подзадачи Π_i вычисляется из таблицы A_i, в которой сформированы все допустимые решения для подграфа $G(Y_i)$, где Y_i задается формулой (1). В общем случае данная таблица может содержать $|Y_i|$ столбцов (по одному столбцу на каждую вершину из Y_i) и $q^{|Y_i|}$ строк (по одной на каждое допустимое решение) при условии, что q — число состояний вершины графа по отношению к допустимому решению. Тот факт, что в таблице A_i столбцами выступают все вершины из Y_i, обозначим через $A_i(Y_i)$. Аналогичным образом обозначим через $A'_i(X_i)$ таблицу, содержащую допустимые решения для подграфа $G(X_i)$, то есть подграфа, индуцированного вершинами только одного «мешка» X_i, и назовем её локальной таблицей. Если узел i является листом, то $A'_i(X_i)$ совпадает с $A_i(Y_i)$. Если i — внутренний узел или корень, то таблица $A_i(Y_i)$ формируется из таблицы $A'_i(X_i)$ и таблиц узлов прямых потомков узла i путем их согласования по строкам. Такое согласование таблиц в реляционной алгебре выполняет естественное соединение $\bowtie$ (далее просто соединение). Данная операция бинарная и отвечает законам коммутативности и ассоциативности. Поэтому общая схема формирования таблицы $A_i(Y_i)$ для внутреннего или корневого узла i, имеющего узлы j и l в качестве прямых потомков, может быть представлена рис. 3. Так как $|X_i| \leqslant k+1$ при всех $i \in I$, то число столбцов всякой таблицы $A_i(Y_i)$, соответствующей узлу i с двумя прямыми потомками, может достигать $3(k+1)$, а число строк в худшем случае (когда операция соединения вырождается в декартово произведение) — значения q^{3k}.

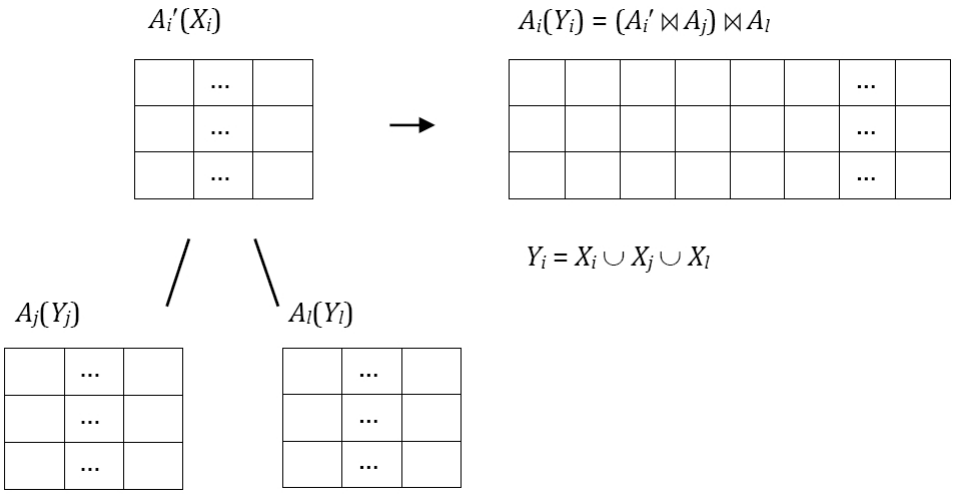

Рис. 3. Создание таблицы для внутреннего узла путем соединения

Пусть (M,T) — Kloks-дерево ширины k. Тогда для всех типов узлов $|Y_i| \leqslant k+1$. Действительно, если узлу i непосредственно подчинены два узла l и j, то

$$X_i = X_l = X_j, \quad Y_i = X_i \cup X_j \cup X_l = X_i, \quad |Y_i| \leqslant k+1.$$

Если i — узел-вставка с прямым потомком j, то

$$X_j \subset X_i, \quad |X_i| = |X_j| + 1, \quad Y_i = X_i \cup X_j = X_i, \quad |Y_i| \leqslant k+1.$$

Если i — узел-удаление с прямым потомком j, то

$$X_i \subset X_j, \ |X_i| = |X_j| - 1, \ Y_i = X_i \cup X_j = X_j, \ |Y_i| \leqslant k+1.$$

Пусть теперь (M,T) — B&S-дерево ширины k. В данном случае также для всех типов узлов $|Y_i| \leqslant k+1$. Для узлов-листьев это очевидно. Рассмотрим внутренние узлы. Заметим, что по построению B&S-дерева узел-сепаратор s всегда имеет только одного родителя (согласно рис. 2 это узел i), одного прямого потомка (это узел j) и $X_s = X_i \cap X_j \neq \varnothing$, $X_s \subseteq X_i$, $X_s \subseteq X_j$. Поэтому для таблицы узла-сепаратора s выполняются отношения $Y_s = X_s \cup X_j = X_j$, $|Y_s| \leqslant k+1$.

Для узла i, играющего роль родителя узла s, справедливы подобные отношения: $Y_i = X_i \cup X_s = X_i$, $|Y_i| \leqslant k+1$. В B&S-дереве любой внутренний узел i является либо узлом-сепаратором, либо родителем одного или двух узлов-сепараторов. Поэтому для него всегда $|Y_i| \leqslant k+1$. Значит, с точки зрения числа столбцов B&S-дерево сопоставимо с Kloks-деревом, хотя имеет в большинстве случаев меньшее число таблиц.

Для Kloks-дерева и B&S-дерева число строк создаваемых таблиц динамического программирования в худшем случае составляет $O(q^k)$. Это теоретическая оценка, и она слишком пессимистичная. На самом деле, как было ранее замечено, всякое дерево декомпозиции (M,T), $M = \{X_i : i \in I\}$, $T = (I,W)$, графа $G = (V,E)$ есть не что иное, как дерево соединений ациклического гиперграфа $H = (V,M)$. Исходя из свойств ациклических гиперграфов, совокупность таблиц A'_i, $i \in I$, обладает монотонным планом соединения. Монотонный план — это такой порядок соединения таблиц, когда число строк всякой промежуточной таблицы не превышает числа строк результирующей таблицы. Монотонный план соединения отвечает обходу дерева соединения ациклического гиперграфа $H = (V,M)$ снизу вверх (от листьев к корню), при этом в роли корня может выступать произвольный узел дерева соединений [6]. Таким образом, при реализации метода динамического программирования по Kloks-дереву или B&S-дереву ширины k в большинстве случаев лишь для таблицы корневого узла число строк достигает значения q^k.

Исключительная особенность B&S-дерева состоит в том, что к нему применимо еще одно свойство ациклических гиперграфов — существование программы полусоединений для совокупности таблиц A'_i, $i \in I$. Представим себе, что мы имеем реляционную базу данных, состоящую из множества таблиц A'_i, $i \in I$. Пусть эта база данных распределена по узлам сети так, что в каждом узле находится только одна таблица, а конфигурация сети задана деревом T. Требуется выполнить запрос к базе данных, в котором надо соединить все таблицы A'_i, $i \in I$. Пусть запрос сформулирован в корневом узле. Тогда процесс вычисления запроса сводится к последовательному соединению таблиц снизу вверх по дереву T с передачей по каналам связи промежуточных таблиц на вышестоящий уровень узлов этой сети. Предположим, что большая часть времени выполнения запроса приходится на время передачи данных. В этом случае

эффективность вычисления запроса зависит от минимизации объёма передаваемых данных. В ациклических реляционных базах данных это достигается за счёт применения программы полусоединений, когда вверх по дереву соединений передаются только те данные, которые участвуют в соединении далее. Очевидно, что процесс выполнения запроса по дереву соединений (M, T) идентичен процессу формирования таблиц динамического программирования по дереву декомпозиции (M, T), а минимизации объёма передаваемых данных — сокращение числа строк промежуточных таблиц динамического программирования.

Поясним действия, выполняемые операцией полусоединения. Пусть заданы две таблицы $A'_i(X_i)$ и $A'_j(X_j)$, отвечающие узлам i и j, где узел j — прямой потомок узла i. Тогда полусоединением $A'_i(X_i)$ и $A'_j(X_j)$ называется таблица, определяемая равенством

$$A'_j \ltimes A'_i = \pi_{X_i}(A'_j \bowtie A'_i), \tag{2}$$

то есть $A'_j \ltimes A'_i$ — эта часть строк таблицы A'_j, которая участвует в соединении со строками из A'_i. По законам реляционной алгебры [6, с. 348]

$$\pi_{X_i}(A'_j \bowtie A'_i) = \pi_{X_i \cap X_j}(A'_j) \bowtie A'_i; \tag{3}$$

$$A'_j \bowtie A'_i = (A'_j \bowtie A'_i) \bowtie A'_i. \tag{4}$$

Если $X_s = X_i \cap X_j$, то из (2)–(4) следует, что

$$A'_j \bowtie A'_i = \pi_{X_s}(A'_j) \bowtie A'_i, \tag{5}$$

где $\pi_{X_s}(A'_j)$ — таблица, которая получается из A_j извлечением столбцов, входящих в X_s, и удалением строк-дубликатов. Так действует операция π, называемая в реляционной алгебре проекцией. Эта операция неизменно уменьшает число строк и столбцов таблицы-операнда. Заметим, что множество $X_s = X_i \cap X_j$ соответствует узлу-сепаратору s в B&S-дереве (рис. 2). Если таблицу для узла-сепаратора получать из таблицы прямого его потомка, применяя операцию проекции, то, согласно (5), операцию соединения таблиц можно осуществить более эффективно с точки зрения времени и памяти. Узел-сепаратор играет в этом случае роль транзита, передающего вверх по дереву только те данные, которые используются для формирования таблицы A_r. Такой эффект от узла-сепаратора гарантируется ацикличностью гиперграфа, ассоциированного с B&S-деревом.

5. Решение задачи о вершинном покрытии методом динамического программирования по B&S-дереву

Задача о вершинном покрытии является типичной NP-трудной задачей на графах, к которой сводятся многие подобные задачи. Напомним, что подмножество $V_1 \subseteq V$ образует вершинное покрытие графа $G = (V, E)$, если каждое ребро из E инцидентно хотя бы одной вершине из V_1. Покрытие называется минимальным, если оно не содержит покрытия с меньшим числом вершин, и наименьшим, если число вершин в нём наименьшее среди всех покрытий графа G. Число вершин в наименьшем покрытии графа G называется числом вершинного покрытия этого графа и обозначается $\beta_0(G)$. В оптимизационной постановке данная задача формулируется следующим образом. Задан граф $G = (V, E)$. Требуется найти число вершинного покрытия $\beta_0(G)$ и наименьшее вершинное покрытие графа G, отвечающее $\beta_0(G)$.

Свойство подмножества $V_1 \subseteq V$ быть вершинным покрытием графа $G = (V, E)$ выражается конечной MSOL-формулой (см. таблицу):

$$\mathrm{VertexCover}(V_1) \Leftrightarrow (\forall e_2)(\exists v_3)(v_3 \in V_1 \,\&\, \mathrm{Incident}(v_3, e_2)). \tag{6}$$

Согласно теореме Курселя, применение к данной задаче метода динамического программирования по дереву декомпозиции приведёт к FPT-алгоритму при условии, что входному графу свойственна ограниченная древовидная ширина. Убедимся в этом. Пусть заданы граф G с $\mathrm{tw}(G) \leqslant k$, его B&S-дерево (M, T), $M = \{X_i : i \in I\}$, $T = (I, W)$, ширины k и с узлом r в роли корня. Реализация метода динамического программирования по B&S-дереву (M, T) предполагает конкретизацию рекуррентной процедуры формирования таблиц $A_i(Y_i)$, то есть определения для каждого из четырёх типов узлов правил, по которым вычисляются все допустимые решения подзадачи Π_i $(i \in I)$. Дадим описание этих правил для задачи о вершинном покрытии, используя язык реляционной алгебры.

Пусть имеем узел-лист i с «мешком» X_i. Для графа $G(X_i)$ задача решается полным перебором. Для этого создается таблица $A_i(Y_i)$ как $A'_i(X_i)$ по следующим правилам. В этой таблице $|X_i| + 1$ столбцов и $2^{|X_i|}$ строк, отдельная строка для каждого из $2^{|X_i|}$ различных подмножеств $Z \subseteq X_i$. Состав вершин, входящих в Z, представляется битовой шкалой b: $b(v) = 1$, если вершина $v \in Z$, и $b(v) = 0$ в противном случае. Столбец $c(Z)$ таблицы $A'_i(X_i)$ содержит характеристику решения: $c(Z) = |Z|$, если Z — вершинное покрытие для $G(X_i)$, иначе $c(Z) = +\infty$. Является ли подмножество Z допустимым решением для Π_i, устанавливается путём проверки на истинность значения предиката (6). Если да, то $c(Z)$ вычисляется по формуле $c(Z) = |Z| = \sum\limits_{v \in X_i} b(v)$.

Рассмотрим узел-сепаратор s с «мешком» X_s и одним прямым потомком j, который имеет «мешок» X_j. В данном случае $X_s \subseteq X_j \subseteq Y_j$, где Y_j — множество вершин исходного графа G, которые принадлежат X_j и всем «мешкам» узлов, являющихся в T потомками узла j. Пусть для узла j ранее уже была построена таблица $A_j(Y_j)$. Тогда таблица $A_s(Y_s)$ вычисляется по формуле

$$A_s = \pi_Q(A_j), \tag{7}$$

где π — операция проекции, которая выбирает из таблицы $A_j(Y_j)$ подмножество столбцов $Q = X_s \cup \{c(Z)\}$. В полученной таблице возможны строки, совпадающие по битовым шкалам и имеющие разные значения $c_1(Z), c_2(Z), \ldots, c_h(Z), h \geqslant 1$. Для всех таких строк полагается

$$c(Z) = \min\{c_1(Z), c_2(Z), \ldots, c_h(Z)\} \tag{8}$$

и в $A_s(Y_s)$ оставляется только одна из них. После этого результирующая таблица $A_s(Y_s)$ содержит только ту информацию из $A_j(Y_j)$, которая необходима для согласования её с таблицей вышестоящего узла. Справедливость формул (7) и (8) вытекает из наследственности свойства, представленного формулой (6): если Z — вершинное покрытие графа $G_i = G(Y_i)$, то множество $Z \cap X_s \subseteq Y_j$ является вершинным покрытием графа $G(X_s)$.

Пусть i — узел-расширение с одним прямым потомком s, для которого $X_s \subseteq X_i$. Предположим, что для узла s ранее уже была создана таблица $A_s(Y_s)$. В данной ситуации вначале формируется локальная таблица $A'_i(X_i)$, которая затем связывается с таблицей $A_s(Y_s)$, и результат записывается в $A_i(Y_i)$:

$$A_i = A'_i \bowtie A_s. \tag{9}$$

Поскольку $X_s \subseteq X_i$, то, согласно (9), в $A_i(Y_i)$ войдут все столбцы из $A'_i(X_i)$ и добавится ещё дополнительный столбец со значениями $c(Z)$ из $A_s(Y_s)$. Пусть $A'_i.c(Z)$ и $A_s.c(Z)$ — характеристики $Z \subseteq X_i$, включённые в результирующую таблицу $A_i(Y_i)$ из A'_i и A_s соответственно. В этих обозначениях формула пересчёта значений $c(Z)$ для A_i имеет вид

$$c(Z) = A'_i.c(Z) + A_s.c(Z) - \sum_{v \in X_s} b(v). \tag{10}$$

Данная формула реализует принцип включения и исключения: мощности допустимых и согласованных между собой решений складываются и вычитается число вершин, учтённых дважды (это вершины из «мешка» узла-сепаратора). Справедливость этой формулы вытекает из наследственности свойства, представленного формулой (6).

Пусть узел i имеет в качестве прямых потомков узлы-сепараторы l и j с «мешками» X_l и X_j соответственно. Допустим, что этим узлам отвечают таблицы $A_l(Y_l)$ и $A_j(Y_j)$, а узлу i — локальная таблица $A'_i(X_i)$. Сначала таблица $A'_i(X_i)$ связывается с таблицей $A_l(Y_l)$ по формулам (9), (10), а затем полученная таблица — с таблицей $A_j(Y_j)$. Таким образом, обработка узла-соединения сводится к двукратному повторению действий, определённых для узла-расширения.

Как только достигается корневой узел r и вычисляется таблица $A_r(V)$, значение числа вершинного покрытия графа G находится по формуле

$$\beta_0(G) = \min\{c(Z) : Z \subseteq Y_r\}. \tag{11}$$

Построение самого вершинного покрытия выполняется при обходе всех узлов дерева T сверху вниз (от корня к листьям) с использованием созданных ранее таблиц динамического программирования. Для хранения $O(n)$ таблиц, каждая из которых имеет размер $O(k2^k)$, необходима память $O(k2^k n)$. Обход дерева T с $O(n)$ узлами можно реализовать за время $O(n)$. Время обработки каждого узла сопоставимо с размером создаваемых таблиц. Поэтому для задачи о вершинном покрытии время работы FPT-алгоритма, реализующего метод динамического программирования по B&S-дереву ширины k, составляет $O(k2^k n)$. Это полностью согласуется с оценкой, приведенной в теореме Курселя. На практике эта оценка может быть более оптимистичной за счет применения B&S-дерева.

Рассмотрим конкретный граф G вместе с его деревом декомпозиции (рис. 4).

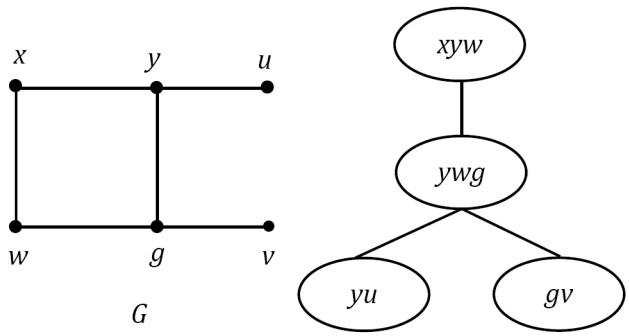

Рис. 4. Граф G и его оптимальное дерево декомпозиции ширины 2

Соответствующее B&S-дерево графа G изображено на рис. 5. Найдём наименьшее вершинное покрытие заданного графа с помощью описанного выше FPT-алгоритма. Процесс подъёма по B&S-дереву начнём с узла 1. Это узел-лист. Узел 2 является узлом-

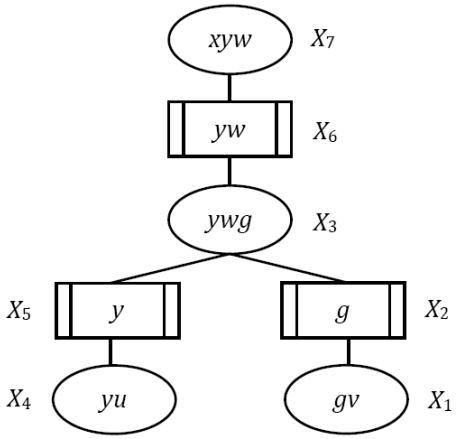

Рис. 5. B&S-дерево ширины 2. Узлу с номером i отвечает «мешок» X_i, $i = 1, 2, \ldots, 7$

сепаратором и родителем для узла 1. Таблицы A_1 и A_2 отвечают узлам 1 и 2 (рис. 6,*а* и *б*). Исходя из (7), (8), $A_2 = \pi_Q(A_1)$, $Q = \{g\} \cup \{c(Z)\}$. Аналогично узлам 4 и 5 соответствуют таблицы A_4, A_5 (рис. 6,*в* и *г*) и $A_5 = \pi_Q(A_4)$, $Q = \{y\} \cup \{c(Z)\}$. Узел 3 — это узел-объединение по отношению к узлам 2 и 5. Для него таблица A'_3 (рис. 6,*д*) содержит локальную информацию для подграфа $G(X_3)$. Соединение A'_3 с таблицей A_2, а затем с таблицей A_5 по формулам (9) и (10) даёт A_3 (рис. 6,*е*). Узел 6 — узел-сепаратор. Для него $A_6 = \pi_Q(A_3)$, $Q = \{y, w\} \cup \{c(Z)\}$ (рис. 6,*ж*). Корневой узел 7 — это узел-расширение по отношению к узлу 6. В соответствии с формулами (9) и (10) таблица A_7 имеет вид, представленный на рис. 6,*з*. Таблица A_7 отвечает корневому узлу. Применение формулы (11) к A_7 даёт $\beta_0(G) = 3$.

A_1

g	v	$c(Z)$
0	0	∞
0	1	1
1	0	1
1	1	2

а

A_2

g	$c(Z)$
0	1
1	1

б

A_4

y	u	$c(Z)$
0	0	∞
0	1	1
1	0	1
1	1	2

в

A_5

y	$c(Z)$
0	1
1	1

г

A_3'

y	w	g	$c(Z)$
0	0	0	∞
0	0	1	1
0	1	0	∞
0	1	1	2
1	0	0	∞
1	0	1	2
1	1	0	2
1	1	1	3

д

A_3

y	w	g	$c(Z)$
0	0	0	∞
0	0	1	2
0	1	0	∞
0	1	1	3
1	0	0	∞
1	0	1	2
1	1	0	3
1	1	1	3

е

A_6

y	w	$c(Z)$
0	0	2
0	1	3
1	0	2
1	1	3

ж

A_7

x	y	w	$c(Z)$
0	0	0	∞
0	0	1	∞
0	1	0	∞
0	1	1	3
1	0	0	3
1	0	1	4
1	1	0	3
1	1	1	4

з

Рис. 6. Таблицы динамического программирования для вычисления вершинного покрытия графа с рис. 4

Спуск по B&S-дереву от корня к листьям определяет следующие наименьшие вершинные покрытия графа G : $\{x, y, g\}, \{x, g, u\}, \{y, w, g\}, \{y, w, v\}$. Заметим, что для исходного графа допустимы деревья декомпозиции, отличные от дерева, указанного на рис. 4, а значит, и B&S-деревья, несхожие с рассмотренным выше B&S-деревом. Решение задачи о вершинном покрытии на их основе приведёт, возможно, к другим оптимальным решениям, хотя неизменным будет значение $\beta_0(G) = 3$.

Очевидно, что приведённый выше FPT-алгоритм можно применять для взвешенной версии задачи о вершинном покрытии, а его идеи относительно правил формирования таблиц динамического программирования для различных типов узлов B&S-дерева — для решения других подобных графовых задач: нахождения наибольшего независимого множества вершин, поиска наибольшей клики, отыскания наименьшего доминирующего множества графа. Конечно, для этих задач необходимо уточнение формул вычисления числовых характеристик решения.

Заключение

Метод динамического программирования на основе дерева декомпозиции — один из современных способов преодоления вычислительной сложности NP-трудных задач выбора с помощью FPT-алгоритмов. В работе предложено бинарное сепараторное дерево декомпозиции, которое обеспечивает компромисс между размером и числом таблиц динамического программирования и расширяет область практического применения данного подхода. Реляционный взгляд на процесс динамического программирования (как процесс соединения таблиц базы данных, распределённой по узлам сети, конфигурация которой задаётся деревом декомпозиции) позволяет применять при решении задач выбора свойства ациклических гиперграфов и способствует алгоритмической конкретизации метода.

ЛИТЕРАТУРА

1. *Downey R. and Fellows M.* Parameterized complexity. New York: Springer Verlag, 1999.
2. *Быкова В. В.* FPT-алгоритмы и их классификация на основе эластичности // Прикладная дискретная математика. 2011. №2(12). С. 40–48.
3. *Bodlaender H. L.* Treewidth: Algorithmic techniques and results // LNCS. 1997. V. 1295. P. 19–36.
4. Компьютер и задачи выбора. М.: Наука, 1989.
5. *Быкова В. В.* Вычислительные аспекты древовидной ширины графа // Прикладная дискретная математика. 2011. №3(13). С. 65–79.
6. *Мейер Д.* Теория реляционных баз данных. М.: Мир, 1987.
7. *Courcelle B.* The monadic second-order logic of graphs. III. Tree decompositions, minors and complexity issues // RAIRO Inform. Theor. Appl. 1992. V. 26(3). P. 257–286.
8. *Bodlaender H. L. and Kloks T.* Efficient and constructive algorithms for the pathwidth and treewidth of graphs // J. Algorithms. 1996. V. 21. P. 358–402.
9. *Быкова В. В.* Математические методы анализа рекурсивных алгоритмов // Журнал СФУ. Математика и физика. 2008. № 1(3). С. 236–246.
10. *Aspvall B., Proskurowski A., and Telle J. A.* Memory requirements for table computations in partial k-tree algorithms // Algorithmica. 2000. V. 27(3). P. 382–394.
11. *Bodlaender H. L. and Fomin F. V.* Tree decompositions with small cost // Discr. Appl. Math. 2005. V. 145(2). P. 143–154.
12. *Kloks T.* Treewidth. Computations and Approximations. Springer Verlag. LNCS. 1994. V. 842.