# О новом подходе к решению задачи мягкого ML-декодирования линейных кодов

Д. Н. Цымжитов

В работе представлен метод декодирования для двоичных линейных кодов, основанный на использовании векторов, которые мы называем неприводимыми. Неприводимый вектор представляет собой минимальную структуру определенного вида и является обобщением понятия минимального кодового слова. Нами предложены два алгоритма, реализующие этот подход применительно к двум классам кодов, характеризуемых некоторыми предположениями относительно структуры графа Таннера проверочной матрицы. В первом случае рассматриваются коды с ациклическим графом Таннера, а во втором случае — коды, у которых все символьные вершины в графе Таннера имеют степень не более 2. Если $n$ — длина кода, то первый алгоритм выполняется за время $O(n^2)$ при $n \to \infty$, а второй — за время $O(n^4)$ при $n \to \infty$.

**Ключевые слова:** мягкое декодирование, декодирование по максимуму правдоподобия, линейные коды, сложность декодирования, минимальные кодовые слова.

## 1. Введение

Известно, что метод декодирования по максимуму правдоподобия или просто ML-декодирование является самым сильным методом в том смысле, что при его использовании вероятность ошибки в принятой последовательности символов достигает минимума. Этот минимум зависит только от свойств канала связи и кода, характеризуя,

таким образом, действительную эффективность последнего вне зависимости от какого-то конкретного алгоритма декодирования. Поэтому в теории помехоустойчивых кодов корректирующую способность кода оценивают в предположении, что на приемном конце осуществляется ML-декодирование.

Декодирование по максимуму правдоподобия является важнейшей и наиболее сложной алгоритмической проблемой в теории кодирования. Известно, что, к примеру, для двоичного симметричного канала связи и произвольных линейных кодов эта проблема в общем случае является NP-полной [1]. Более того, она остается таковой даже тогда, когда допускается сколь угодно долгая предобработка кода [2]. К настоящему времени для случая произвольных линейных кодов разработано большое множество алгоритмов, позволяющих уменьшить асимптотическую сложность ML-декодирования. Все они имеют сложность, зависящую экспоненциально от длины кода, но в отличие от переборного метода — с меньшим показателем экспоненты. Неплохой обзор этих методов представлен в статье [3].

В настоящей работе предложен алгоритм, выполняющий декодирование по максимуму правдоподобия за определенное число итераций, в ходе которых производится постепенное уточнение решения. На каждой итерации алгоритма выполняется корректировка текущего приближения посредством прибавления к нему специально подобранного вектора, обладающего рядом свойств. Во-первых, корректирующий вектор должен изменить текущее приближение таким образом, чтобы на некоторой позиции его синдрома — например, с индексом $i_0$ — единица сменилась на ноль, а на всех позициях с индексами из некоторого множества $K$, где стоят нули, значение не поменялось. Другими словами, синдром текущего приближения должен улучшиться. Во-вторых, корректирующий вектор должен быть неприводимым. Это означает, что его носитель не должен содержать носителей ненулевых кодовых слов в качестве собственного подмножества. Наконец, в-третьих, корректирующий вектор должен быть оптимальным решением подзадачи определенного вида в множестве всех векторов, удовлетворяющих первым двум требованиям. Мы обозначим это множество $\mathcal{I}_{K,i_0}$. Подробное описание и обоснование этого метода представлено в разделе 3.

Понятие неприводимого вектора является простым обобщением понятия минимального кодового слова. На самом деле единственное их отличие состоит в том, что неприводимые векторы не обязаны быть кодовыми. Минимальные кодовые слова были впервые рассмотрены в статье [4], где был предложен основанный на их использовании итеративный алгоритм декодирования для произвольных линейных кодов, относящийся к семейству градиентоподобных алгоритмов. Кроме того, свойства множества минимальных кодовых слов имеют непосредственное отношение к структуре областей Вороного, изучаемых в контексте задачи мягкого декодирования линейных кодов [5]. Наконец, интерес к минимальным кодовым словам возник и со стороны криптографии в серии работ, посвященных линейным схемам разделения секрета [6, 7]. Изучению свойств минимальных кодовых слов и основанного на их использовании метода декодирования целиком или частично посвящены работы [3, 4, 8, 9].

Что же касается предлагаемого нами метода, то требование неприводимости объясняется двумя соображениями. Во-первых, оно позволяет существенно уменьшить множество кандидатов, среди которых производится поиск корректирующего вектора, без ущерба для локальной оптимальности последнего (см. предложение 8). Во-вторых, в некоторых случаях предположение о неприводимости позволяет получать наилучший корректирующий вектор, не прибегая к перебору множества $\mathcal{I}_{K,i_0}$.

В качестве демонстрации того, как можно получить оптимальный корректирующий вектор без перебора, в разделах 4 и 5 предлагаются алгоритмы, реализующие данный подход применительно к двум достаточно простым классам линейных кодов. Эти коды характеризуются некоторыми предположениями относительно структуры графа Таннера проверочной матрицы. В первом случае рассматриваются коды с ациклическим графом Таннера, а во втором случае — коды, у которых все символьные вершины в графе Таннера имеют степень не более 2. В обоих случаях получено описание множества $\mathcal{I}_{K,i_0}$ в терминах подграфов графа Таннера: в первом случае векторы множества $\mathcal{I}_{K,i_0}$ порождают в графе Таннера поддеревья определенного вида (см. предложение 10); во втором случае векторы множества $\mathcal{I}_{K,i_0}$ порождают в нем простые цепи определенного вида (см. предложе-

ние 16). Если $n$ — длина кода, то первый алгоритм выполняется за время $O(n^2)$ при $n \to \infty$, а второй — за время $O(n^4)$ при $n \to \infty$.

Доказано, что в случае графа Таннера с символьными вершинами степени не более 2 нахождение оптимального корректирующего вектора сводится к неориентированной задаче о кратчайшей простой цепи во взвешенном графе без циклов отрицательного веса, которая является частным случаем задачи о $T$-соединении минимального веса. Последняя, как известно, сводится к задаче о совершенном паросочетании минимального веса. В конце раздела 5 мы рассматриваем случай, когда граф Таннера содержит символьные вершины, степень которых равна в точности двум. Нетрудно доказать (см. лемму 23), что в этой ситуации задача декодирования по максимуму правдоподобия полностью эквивалентна задаче о $T$-соединении минимального веса, и, следовательно, решается за время $O(n^3)$ при $n \to \infty$. Что касается кодов с ациклическим графом Таннера, следует отметить, что алгоритм min-sum (sum-product) в этом случае также предоставляет максимально правдоподобное решение, имея сложность всего лишь $O(n)$ при $n \to \infty$ [10].

Автор выражает благодарность сотрудникам кафедры математической теории интеллектуальных систем механико-математического факультета МГУ профессору Э. Э. Гасанову и к.ф.-м.н. П. А. Пантелееву за постановку задачи и помощь в работе.

## 2. Постановка задачи и основные результаты

**Постановка задачи.** Детальное изложение фактов и понятий, составляющих содержание этого раздела, можно найти в [11, гл. 1], [12, гл. 1] и [13, гл. 1].

Пусть имеется дискретный канал связи без памяти с двоичным входным алфавитом $\mathbb{F}_2$ и вещественным выходным алфавитом $\Omega \subset \mathbb{R}$. Если множество $\Omega$ не более чем счетно, то поведение канала связи определяется дискретной условной вероятностью $\mathrm{p}(r \mid a)$ того, что на выходе канала связи появится символ $r \in \Omega$, если на его вход был подан символ $a \in \mathbb{F}_2$. Если множество $\Omega$ несчетно, мы будем предполагать существование функции плотности условной вероятно-

сти $\mathrm{p}(r \mid a)$, похожим образом определяющей поведение канала связи. Как в дискретном, так и недискретном случае мы используем единое обозначение $\mathrm{p}(r \mid a)$. Будем считать, что $\mathrm{p}(r \mid a) \neq 0$ для всех $r \in \Omega$ и $a \in \mathbb{F}_2$.

Если данные передаются блоками длины $n$, то коль скоро канал связи не имеет памяти, для любых слов $\mathbf{r} \in \Omega^n$ и $\mathbf{a} \in \mathbb{F}_2^n$ будем иметь

$$\mathrm{p}(\mathbf{r} \mid \mathbf{a}) = \prod_{j=1}^{n} \mathrm{p}(r_j \mid a_j).$$

Здесь $\mathrm{p}(\mathbf{r} \mid \mathbf{a})$ обозначает саму условную вероятность, либо плотность условной вероятности того, что на приемном конце будет принято слово $\mathbf{r} \in \Omega^n$, если было передано слово $\mathbf{a} \in \mathbb{F}_2^n$. Частными случаями модели двоичного дискретного канала без памяти являются, например, двоичный симметричный канал, двоичный канал с аддитивным белым гауссовским шумом, двоичный канал с лапласовским шумом, двоичный канал с шумом Коши и другие модели.

Будем считать, что при передаче данных используется линейный двоичный код $\mathcal{C}$ длины $n$. Если на приемном конце наблюдается некоторый вектор $\mathbf{r} \in \Omega^n$, то декодировать его — значит восстановить по нему тот кодовый вектор $\mathbf{c} \in \mathcal{C}$, который был подан на вход канала связи. Блоковая ошибка декодирования — это исход, при котором декодер выдает кодовую последовательность, не совпадающую с той, что была передана на самом деле. Известно, что наиболее сильным методом декодирования, минимизирующим вероятность блоковой ошибки, является *декодирование по максимуму апостериорной вероятности* или просто *MAP-декодирование* (англ. *maximum a posteriori decoding*). Обозначим через $\mathrm{P}(\mathbf{c} \mid \mathbf{r})$ вероятность того, что был передан вектор $\mathbf{c} \in \mathcal{C}$, при условии, что на выходе канала связи принят вектор $\mathbf{r} \in \Omega^n$. Задача MAP-декодирования наблюдаемой $\mathbf{r} \in \Omega^n$ состоит в нахождении вектора $\mathbf{c} \in \mathcal{C}$, максимизирующего апостериорную вероятность $\mathrm{P}(\mathbf{c} \mid \mathbf{r})$.

Мы будем предполагать, что источник распределен равномерно, то есть все слова $\mathbf{c} \in \mathcal{C}$ передаются с одинаковой вероятностью $\mathrm{P}(\mathbf{c}) = |\mathcal{C}|^{-1}$. Легко видеть, что в этом случае задача MAP-декодирования становится эквивалентной задаче *декодирования по максимуму прав-*

*доподобия* или просто *ML-декодирования* (англ. *maximum likelihood decoding*), которая состоит в максимизации величины $\mathrm{p}(\mathbf{r} \mid \mathbf{c})$.

Определим на множестве $\Omega$ функцию

$$\lambda(r) = \ln \frac{\mathrm{p}(r \mid 0)}{\mathrm{p}(r \mid 1)},$$

которая называется *логарифмическим отношением правдоподобия*. Для любой пары векторов $\boldsymbol{\alpha} \in \mathbb{R}^n$ и $\mathbf{a} \in \mathbb{F}_2^n$ определим их «скалярное произведение» как

$$\langle \boldsymbol{\alpha}, \mathbf{a} \rangle = \sum_{j\colon a_j \neq 0} \alpha_j.$$

Нетрудно проверить, что для всех $\mathbf{r} \in \Omega^n$ и $\mathbf{a} \in \mathbb{F}_2^n$ верно соотношение

$$\mathrm{p}(\mathbf{r} \mid \mathbf{a}) = \mathrm{p}(\mathbf{r} \mid \mathbf{0}) e^{-\langle \boldsymbol{\lambda}, \mathbf{a} \rangle}, \quad \text{где } \boldsymbol{\lambda} = (\lambda(r_1), \ldots, \lambda(r_n)).$$

Так что задача ML-декодирования может быть записана в виде

$$\langle \boldsymbol{\lambda}, \mathbf{c} \rangle \to \min, \quad \mathbf{c} \in \mathcal{C}. \tag{1}$$

Мы видим, что вектор $\boldsymbol{\lambda}$ содержит в себе всю информацию, необходимую для ML-декодирования, и в этом смысле является достаточной статистикой.

Пусть код $\mathcal{C}$ задан проверочной матрицей $\mathbf{H} = (h_{ij})$ размера $m \times n$. Тогда условие $\mathbf{c} \in \mathcal{C}$ можно представить в виде равенства $\mathbf{Hc} = \mathbf{0}$ или системы

$$h_i(\mathbf{c}) = \sum_{j=1}^{n} h_{ij} c_j = 0, \quad i = 1, \ldots, m.$$

Все наши дальнейшие рассуждения будут проводиться в контексте задачи (1), так что мы будем считать вектор $\boldsymbol{\lambda} \in \mathbb{R}^n$ фиксированным.

**Основные результаты.** Здесь мы коротко изложим суть нашего подхода к решению задачи (1) и сформулируем основные результаты.

Рассмотрим последовательность кодов

$$\mathcal{C}_t = \{\mathbf{c} \in \mathbb{F}_2^n \mid h_i(\mathbf{c}) = 0 \text{ для всех } i \leqslant t\}, \quad t = 0, \ldots, m,$$

образующих убывающую цепочку

$$\mathbb{F}_2^n = \mathcal{C}_0 \supseteq \mathcal{C}_1 \supseteq \ldots \supseteq \mathcal{C}_m = \mathcal{C}.$$

И рассмотрим последовательность векторов

$$\mathbf{c}_t = \arg\min_{\mathbf{c}\in\mathcal{C}_t} \langle \boldsymbol{\lambda}, \mathbf{c}\rangle, \quad t = 0, \ldots, m, \tag{2}$$

так что вектор $\mathbf{c}_m \in \mathcal{C}$ является оптимальным решением задачи (1), и имеет место цепочка неравенств

$$\langle \boldsymbol{\lambda}, \mathbf{c}_0\rangle \leqslant \langle \boldsymbol{\lambda}, \mathbf{c}_1\rangle \leqslant \ldots \leqslant \langle \boldsymbol{\lambda}, \mathbf{c}_m\rangle.$$

Предлагаемый нами подход заключается в том, чтобы получить решение $\mathbf{c}_m \in \mathcal{C}$ путем построения последовательности (2). Иными словами, мы постепенно сужаем множество допустимых решений задачи, добавляя одно за другим ограничения $h_i$, $i = 1, \ldots, m$. При этом, сперва находится оптимальное решение задачи без ограничений, затем — с одним ограничением, двумя ограничениями и т. д. Процесс продолжается до тех пор, пока не будут учтены все ограничения и найдено удовлетворяющее им решение. Ключевая идея данного метода в том, что на каждом шаге этого процесса для получения оптимального решения используется оптимум, полученный на предыдущем шаге.

Реализуя этот подход, мы рассмотрим два частных случая, каждый из которых характеризуется некоторым предположением относительно структуры кода $\mathcal{C}$, а точнее — структуры графа Таннера его проверочной матрицы. Напомним, что *графом Таннера* матрицы $\mathbf{H}$ называется двудольный граф, который содержит вершины двух видов — проверочные и символьные. *Проверочные вершины*, соответствующие строкам матрицы $\mathbf{H}$, мы обозначим $f_i$, $i = 1, \ldots, m$. *Символьные вершины*, соответствующие столбцам матрицы $\mathbf{H}$, мы обозначим $x_j$, $j = 1, \ldots, n$. Ребра соединяют только те пары вершин, в которых одна является проверочной, а другая — символьной. Две вершины $f_i$ и $x_j$ соединяются ребром тогда и только тогда, когда $h_{ij} = 1$. Этот граф мы будем обозначать символом $\mathfrak{T}$. На рисунке 1 изображен пример графа Таннера для $[7, 4, 3]$-кода Хэмминга.

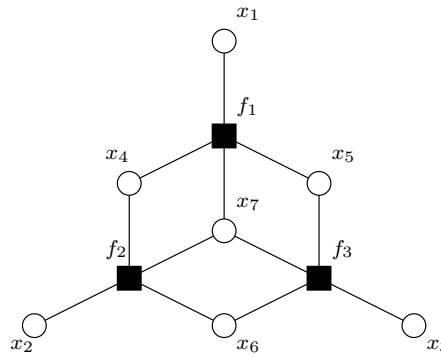


Рис. 1. Граф Таннера для $[7, 4, 3]$-кода Хэмминга.

Всюду далее мы будем предполагать, что строки проверочной матрицы $\mathbf{H}$ содержат не менее двух ненулевых элементов. Это значит, что граф $\mathfrak{T}$ не содержит концевых проверочных вершин. Также будем предполагать, что матрица $\mathbf{H}$ не содержит нулевых или одинаковых столбцов. Это значит, что минимальное расстояние кода $\mathcal{C}$ больше или равно трем. Наконец, будем считать, что $m/n \sim O(1)$ при $n \to \infty$. Впрочем, эти предположения являются естественными. Теперь мы сформулируем основные результаты.

**Теорема 1.** *Если граф Таннера $\mathfrak{T}$ ациклический, то существует алгоритм, вычисляющий последовательность (2), который выполняется за время $O(n^2)$ при $n \to \infty$.*

**Теорема 2.** *Если граф Таннера $\mathfrak{T}$ таков, что степени всех его символьных вершин не превосходят 2, то существует алгоритм, вычисляющий последовательность (2), который выполняется за время $O(n^4)$ при $n \to \infty$.*

## 3. Общий подход

Положим $I = \{i \in \mathbb{N} \mid 1 \leqslant i \leqslant m\}$ и $J = \{j \in \mathbb{N} \mid 1 \leqslant j \leqslant n\}$. Для любого вектора $\mathbf{a} \in \mathbb{F}_2^n$ определим его *носитель* как множество

$$\operatorname{supp} \mathbf{a} = \{j \in J \mid a_j = 1\}.$$

**Лемма 3.** *Для любых векторов* $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$ *и* $\boldsymbol{\alpha} \in \mathbb{R}^n$ *имеет место равенство* $\langle \boldsymbol{\alpha}, \mathbf{a}+\mathbf{b}\rangle = \langle \boldsymbol{\alpha}, \mathbf{a}\rangle + \langle \boldsymbol{\beta}, \mathbf{b}\rangle$, *где* $\boldsymbol{\beta} = ((-1)^{a_j}\alpha_j)$.

**Доказательство.** Имеем

$$\langle \boldsymbol{\beta}, \mathbf{b}\rangle = \sum_{j \in \operatorname{supp}\mathbf{b}} (-1)^{a_j}\alpha_j = \sum_{\substack{j \in \operatorname{supp}\mathbf{b} \\ \setminus \operatorname{supp}\mathbf{a}}} \alpha_j - \sum_{\substack{j \in \operatorname{supp}\mathbf{b} \\ \cap \operatorname{supp}\mathbf{a}}} \alpha_j =$$

$$= \sum_{\substack{j \in \operatorname{supp}\mathbf{b} \\ \triangle \operatorname{supp}\mathbf{a}}} \alpha_j - \sum_{j \in \operatorname{supp}\mathbf{a}} \alpha_j = \langle \boldsymbol{\alpha}, \mathbf{a}+\mathbf{b}\rangle - \langle \boldsymbol{\alpha}, \mathbf{a}\rangle.$$

Лемма доказана.

*Вектором жесткого решения* для $\boldsymbol{\lambda}$ называется вектор $\hat{\mathbf{r}} \in \mathbb{F}_2^n$ такой, что

$$\hat{r}_j = \begin{cases} 0, & \text{если } \lambda_j \geqslant 0, \\ 1, & \text{если } \lambda_j < 0 \end{cases} \qquad \text{для всех } j \in J.$$

А *вектором правдоподобия* для $\boldsymbol{\lambda}$ называется вектор $\boldsymbol{\mu} \in \mathbb{R}^n$ такой, что $\mu_j = |\lambda_j|$ для всех $j \in J$. Зададим функцию $d_{\boldsymbol{\mu}}\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{R}_+$ формулой

$$d_{\boldsymbol{\mu}}(\mathbf{a}, \mathbf{b}) = \sum_{j\colon a_j \neq b_j} \mu_j.$$

Она называется *взвешенным расстоянием Хэмминга* в $\mathbb{F}_2^n$. Заметим, что она не всегда является метрикой в строгом смысле. Несмотря на то, что расстояние $d_{\boldsymbol{\mu}}$ симметрично и удовлетворяет неравенству треугольника, оно становится вырожденным, если хотя бы одна из компонент вектора $\boldsymbol{\mu}$ равна нулю.

Как мы уже отмечали, вектор $\boldsymbol{\lambda}$ является достаточной статистикой. Из определения векторов $\hat{\mathbf{r}}$ и $\boldsymbol{\mu}$ следует, что $\lambda_j = (-1)^{\hat{r}_j}\mu_j$ для всех $j \in J$. Стало быть, пара $(\hat{\mathbf{r}}, \boldsymbol{\mu})$ также является достаточной статистикой. Более того, в силу леммы 3 имеем $d_{\boldsymbol{\mu}}(\hat{\mathbf{r}}, \mathbf{a}) = \langle \boldsymbol{\mu}, \hat{\mathbf{r}}+\mathbf{a}\rangle = \langle \boldsymbol{\mu}, \hat{\mathbf{r}}\rangle + \langle \boldsymbol{\lambda}, \mathbf{a}\rangle$ для всех $\mathbf{a} \in \mathbb{F}_2^n$. Поэтому задача (1) может быть переписана в виде

$$d_{\boldsymbol{\mu}}(\hat{\mathbf{r}}, \mathbf{c}) \to \min, \quad \mathbf{c} \in \mathcal{C},$$

и все элементы последовательности (2) представляются в виде

$$\mathbf{c}_t = \arg\min_{\mathbf{c}\in\mathcal{C}_t} d_{\boldsymbol{\mu}}(\hat{\mathbf{r}}, \mathbf{c}), \quad t = 0, \ldots, m.$$

Для всех $t$ вектор $\mathbf{c}_t$ является ближайшим к $\hat{\mathbf{r}}$ в множестве $\mathcal{C}_t$. В частности, можно положить $\mathbf{c}_0 = \hat{\mathbf{r}}$ (см. рисунок 2).

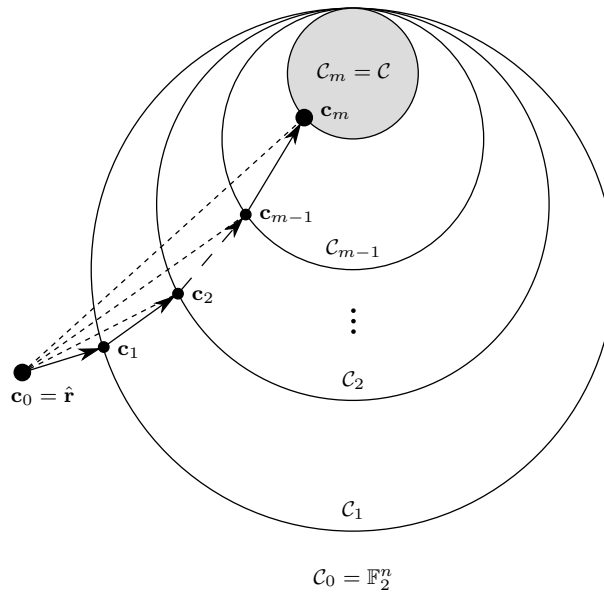


Рис. 2. Геометрический смысл элементов последовательности (2).

Для произвольных вектора $\boldsymbol{\alpha} \in \mathbb{R}^n$ и множества $\mathcal{A} \subset \mathbb{F}_2^n$ положим

$$\min_{\boldsymbol{\alpha}} \mathcal{A} = \{\mathbf{a} \in \mathcal{A} \mid \langle \boldsymbol{\alpha}, \mathbf{a}\rangle \leqslant \langle \boldsymbol{\alpha}, \mathbf{b}\rangle \text{ для всех } \mathbf{b} \in \mathcal{A}\}.$$

Из предыдущих рассуждений, в частности, следует

**Лемма 4.** *Верно включение* $\hat{\mathbf{r}} \in \min_{\boldsymbol{\lambda}} \mathbb{F}_2^n$.

Рассмотрим произвольные подмножество $K \subsetneq I$ и индекс $i \in I\backslash K$. Положим

$$\mathcal{C}_K = \{\mathbf{a} \in \mathbb{F}_2^n \mid h_i(\mathbf{a}) = 0 \text{ для всех } i \in K\},$$
$$\mathcal{C}_{K,i} = \{\mathbf{a} \in \mathcal{C}_K \mid h_i(\mathbf{a}) = 1\}.$$

Обозначим $L = K \cup \{i\}$.

**Предложение 5.** *Пусть* $\mathbf{c} \in \mathcal{C}_{K,i}$. *Имеем* $\mathbf{c}+\mathbf{u} \in \mathcal{C}_L$ *тогда и только тогда, когда* $\mathbf{u} \in \mathcal{C}_{K,i}$. *Более того,* $\mathbf{c}+\mathbf{u} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_L$ *тогда и только тогда, когда* $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$, *где* $\boldsymbol{\gamma} = ((-1)^{c_j}\lambda_j)$.

**Доказательство.** Имеем $h_i(\mathbf{c}+\mathbf{u}) = h_i(\mathbf{c})+h_i(\mathbf{u}) = 1+h_i(\mathbf{u})$, а также $h_{i'}(\mathbf{c}+\mathbf{u}) = h_{i'}(\mathbf{c}) + h_{i'}(\mathbf{u}) = h_{i'}(\mathbf{u})$ для всех $i' \in K$. Следовательно, $\mathbf{c}+\mathbf{u} \in \mathcal{C}_L$ тогда и только тогда, когда $h_i(\mathbf{u}) = 1$ и $h_{i'}(\mathbf{u}) = 0$ для всех $i' \in K$. А это и означает, что $\mathbf{u} \in \mathcal{C}_{K,i}$.

Далее, из доказанного следует, что $\mathcal{C}_L = \{\mathbf{c} + \mathbf{u}' \mid \mathbf{u}' \in \mathcal{C}_{K,i}\}$. Поэтому $\mathbf{c}+\mathbf{u} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_L$ тогда и только тогда, когда $\langle \boldsymbol{\lambda}, \mathbf{c}+\mathbf{u}'\rangle \geqslant \langle \boldsymbol{\lambda}, \mathbf{c}+\mathbf{u}\rangle$ для всех $\mathbf{u}' \in \mathcal{C}_{K,i}$. В силу леммы 3 последнее возможно тогда и только тогда, когда для всех же $\mathbf{u}' \in \mathcal{C}_{K,i}$ выполняется неравенство $\langle \boldsymbol{\gamma}, \mathbf{u}'\rangle \geqslant \langle \boldsymbol{\gamma}, \mathbf{u}\rangle$. Но это и означает, что $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$. Предложение доказано.

Как известно, *минимальными* называются кодовые векторы, носители которых не содержат носителей других ненулевых кодовых векторов. Следующее определение обобщает понятие минимального кодового вектора. Пусть $\mathcal{D}$ — произвольный линейный код в $\mathbb{F}_2^n$.

**Определение 1.** Ненулевой вектор $\mathbf{a} \in \mathbb{F}_2^n$ будем называть *неприводимым относительно* $\mathcal{D}$ (или $\mathcal{D}$-неприводимым), если не существует такого вектора $\mathbf{c} \in \mathcal{D}$, что $\mathbf{c} \neq \mathbf{0}$ и $\operatorname{supp}\mathbf{c} \subsetneq \operatorname{supp}\mathbf{a}$. Множество всех $\mathcal{D}$-неприводимых векторов будем обозначать $\operatorname{irr}\mathcal{D}$.

Таким образом, неприводимые векторы — в отличие от минимальных векторов — вовсе не обязаны быть кодовыми. Ясно, что для любого подкода $\mathcal{D}' \subset \mathcal{D}$ верно включение $\operatorname{irr}\mathcal{D} \subset \operatorname{irr}\mathcal{D}'$.

**Определение 2.** Сумму векторов $\mathbf{a}, \mathbf{b} \in \mathbb{F}_2^n$, носители которых не пересекаются, будем называть *прямой суммой* и обозначать записью $\mathbf{a} \oplus \mathbf{b}$. Так что $\operatorname{supp}(\mathbf{a} \oplus \mathbf{b}) = \operatorname{supp}\mathbf{a} \cup \operatorname{supp}\mathbf{b}$.

**Предложение 6.** *Всякий ненулевой вектор* $\mathbf{a} \in \mathbb{F}_2^n$ *можно представить в виде прямой суммы* $\mathbf{a} = \mathbf{a}' \oplus \mathbf{c}$, *где* $\mathbf{a}' \in \operatorname{irr}\mathcal{D}$ *и* $\mathbf{c} \in \mathcal{D}$.

**Доказательство.** Если $\mathbf{a} \in \operatorname{irr}\mathcal{D}$, то доказывать нечего. Если $\mathbf{a} \notin \operatorname{irr}\mathcal{D}$, то по определению существует такой вектор $\mathbf{c}_1 \in \mathcal{D}\backslash\{\mathbf{0}\}$, что

$\mathbf{a} = \mathbf{a}_1 \oplus \mathbf{c}_1$ и $\mathbf{a}_1 \neq \mathbf{0}$. Если $\mathbf{a}_1 \in \operatorname{irr} \mathcal{D}$, то все доказано. В противном случае применяем к вектору $\mathbf{a}_1$ то же рассуждение. Этот процесс можно продолжать до тех пор, пока не будет получено разложение вида $\mathbf{a} = \mathbf{a}_p \oplus \mathbf{c}_p \oplus \ldots \oplus \mathbf{c}_1$, в котором $\mathbf{a}_p \in \operatorname{irr} \mathcal{D}$ и $\mathbf{c}_1 \oplus \ldots \oplus \mathbf{c}_p \in \mathcal{D}$. Предложение доказано.

Такой вектор $\mathbf{a}'$ мы будем называть *$\mathcal{D}$-неприводимой компонентой* вектора $\mathbf{a}$. Следующее утверждение, несмотря на свою простоту, будет играть важную роль при доказательстве нашего основного результата, которое мы предъявим в одном из следующих разделов. Поэтому мы сформулируем его в виде отдельной леммы.

**Лемма 7.** *Если $\mathbf{c} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_K$, то для всех $\mathbf{c}' \in \mathcal{C}_K$ имеем $\langle \boldsymbol{\gamma}, \mathbf{c}' \rangle \geqslant 0$, где $\boldsymbol{\gamma} = ((-1)^{c_j} \lambda_j)$.*

**Доказательство.** $\langle \boldsymbol{\gamma}, \mathbf{c}' \rangle = \langle \boldsymbol{\lambda}, \mathbf{c} + \mathbf{c}' \rangle - \langle \boldsymbol{\lambda}, \mathbf{c} \rangle \geqslant 0$ для всех $\mathbf{c}' \in \mathcal{C}_K$. Лемма доказана.

**Предложение 8.** *Пусть $\mathbf{c} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_K$. Тогда:*

1) *Если $\mathbf{c} \notin \mathcal{C}_{K,i}$, то $\mathbf{c} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_L$.*
2) *Если $\mathbf{c} \in \mathcal{C}_{K,i}$, то найдется $\mathcal{C}_L$-неприводимый вектор $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$, где $\boldsymbol{\gamma} = ((-1)^{c_j} \lambda_j)$. При этом, $\mathbf{c} + \mathbf{u} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_L$.*

**Доказательство.** Первая часть предложения очевидна, если учесть, что $\mathcal{C}_K = \mathcal{C}_{K,i} \cup \mathcal{C}_L$ и $\mathcal{C}_{K,i} \cap \mathcal{C}_L = \varnothing$. Докажем второе утверждение.

Пусть $\mathbf{u}_0 \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$. Согласно предложению 6 из вектора $\mathbf{u}_0$ можно выделить $\mathcal{C}_L$-неприводимую компоненту $\mathbf{u}$, так что $\mathbf{u}_0 = \mathbf{u} \oplus \mathbf{c}'$, где $\mathbf{c}' \in \mathcal{C}_L$. Ясно, что $\mathbf{u} \in \mathcal{C}_{K,i}$. Поскольку $\mathbf{c}' \in \mathcal{C}_K$, согласно лемме 7 имеем $\langle \boldsymbol{\gamma}, \mathbf{c}' \rangle \geqslant 0$. Из этого неравенства и того, что $\mathbf{u}_0 = \mathbf{u} \oplus \mathbf{c}'$, получаем неравенство $\langle \boldsymbol{\gamma}, \mathbf{u}_0 \rangle = \langle \boldsymbol{\gamma}, \mathbf{u} \rangle + \langle \boldsymbol{\gamma}, \mathbf{c}' \rangle \geqslant \langle \boldsymbol{\gamma}, \mathbf{u} \rangle$. Следовательно, $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$, и в силу предложения 5 имеем $\mathbf{c} + \mathbf{u} \in \min_{\boldsymbol{\lambda}} \mathcal{C}_L$. Предложение доказано.

Обозначим $\mathcal{I}_{K,i} = \mathcal{C}_{K,i} \cap \operatorname{irr} \mathcal{C}_L$. Предложение 8 показывает, что для отыскания минимизирующего вектора $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{C}_{K,i}$ мы вполне можем ограничиться подмножеством $\mathcal{C}_L$-неприводимых векторов $\mathcal{I}_{K,i} \subset \mathcal{C}_{K,i}$ и искать вектор $\mathbf{u} \in \min_{\boldsymbol{\gamma}} \mathcal{I}_{K,i}$. Также в этой теореме легко читается основная идея объявленного нами алгоритма декодирования, который должен вычислять последовательность приближений (2): имея

вектор из $\min_{\boldsymbol{\lambda}} \mathcal{C}_K$, мы всегда сможем построить вектор из $\min_{\boldsymbol{\lambda}} \mathcal{C}_L$, коль скоро в нашем распоряжении будет алгоритм для решения подзадачи вида

$$\langle \boldsymbol{\gamma}, \mathbf{u} \rangle \to \min, \quad \mathbf{u} \in \mathcal{I}_{K,i_0} \quad (\text{если } \langle \boldsymbol{\gamma}, \mathbf{c} \rangle \geqslant 0 \text{ для всех } \mathbf{c} \in \mathcal{C}_K). \qquad (3)$$

Итак, мы можем дать нашему алгоритму следующее описание.

**Алгоритм 1 (Алгоритм декодирования).** *Пусть дана процедура* $\mathtt{A}$*, которая для любых параметров* $\boldsymbol{\gamma} \in \mathbb{R}^n$*,* $K \subsetneq I$ *и* $i_0 \in I \backslash K$ *выдает результат* $\mathbf{u} = \mathtt{A}(\boldsymbol{\gamma}, K, i_0)$*, являющийся оптимальным решением задачи (3). Алгоритм* $\mathtt{Decoder}_{\mathtt{A}}$ *имеет следующее определение.*

***Параметры****: вектор* $\boldsymbol{\lambda} \in \mathbb{R}^n$*.*

***Результат****: вектор* $\mathbf{c} \in \mathcal{C}$*.*

***Псевдокод****:*

$\mathbf{c} \leftarrow \hat{\mathbf{r}}(\boldsymbol{\lambda})$*;*
$K \leftarrow \varnothing$*;* $i \leftarrow 1$*;*
*Пока* $i \leqslant m$*, цикл*
  *Если* $h_i(\mathbf{c}) = 0$*, то*
    $\mathbf{u} \leftarrow \mathbf{0}$*;*
  *Иначе*
    $\boldsymbol{\gamma} \leftarrow ((-1)^{c_j} \lambda_j)$*;* $\mathbf{u} \leftarrow \mathtt{A}(\boldsymbol{\gamma}, K, i)$*;*
  *Конец если;*
  $\mathbf{c} \leftarrow \mathbf{c} + \mathbf{u}$*;*
  $K \leftarrow K \cup \{i\}$*;* $i \leftarrow i + 1$*;*
*Конец цикла;*
*Возврат* $\mathbf{c}$*.*

Результат выполнения алгоритма $\mathtt{Decoder}_{\mathtt{A}}$ с параметром $\boldsymbol{\lambda}$ обозначим $\mathtt{Decoder}_{\mathtt{A}}(\boldsymbol{\lambda})$. Из леммы 4, предложения 8 и описания алгоритма 1 вытекает

**Следствие 9.** *Результат* $\mathbf{c} = \mathtt{Decoder}_{\mathtt{A}}(\boldsymbol{\lambda})$*, получаемый алгоритмом* $\mathtt{Decoder}_{\mathtt{A}}$ *путем вычисления последовательности (2), является оптимальным решением задачи (1).*

Итак, нам удалось свести задачу (1) к задаче (3), решению которой мы посвятим два следующих раздела. Мы увидим, что при некоторых предположениях относительно структуры графа Таннера

$\mathfrak{T}$ ее решение оказывается достаточно простым. Сперва мы рассмотрим эту задачу применительно к случаю, когда граф $\mathfrak{T}$ не содержит циклов. После этого мы перейдем к изучению случая, когда все символьные вершины в графе $\mathfrak{T}$ имеют степень не более 2, но сам граф может содержать циклы.

## 4. Граф $\mathfrak{T}$ без циклов

В этом разделе мы продемонстрируем, как можно применить описанный выше общий подход в наиболее простой ситуации, когда граф $\mathfrak{T}$ является ациклическим.

**Описание множества $\mathcal{I}_{K,i_0}$.** Введем обозначения, которые нам понадобятся в дальнейшем. Для всех индексов $i \in I$ и $j \in J$ таких, что $h_{ij} = 1$, будем писать $i \sim j$ или $f_i \sim x_j$, имея в виду, что вершины $f_i$ и $x_j$ графа $\mathfrak{T}$ соединены ребром. Положим

$$I_j = \{i \in I \mid i \sim j\}, \quad J_i = \{j \in J \mid i \sim j\}.$$

Всюду далее, говоря о подграфах графа $\mathfrak{T}$, мы будем иметь в виду только *порожденные* подграфы, которые вместе с любыми двумя своими вершинами содержат все ребра, соединяющие эти вершины в графе $\mathfrak{T}$. Для произвольного подграфа $\mathfrak{S}$ в $\mathfrak{T}$ положим

$$I(\mathfrak{S}) = \{i \in I \mid f_i \text{ — вершина } \mathfrak{S}\}, \;\; J(\mathfrak{S}) = \{j \in J \mid x_j \text{ — вершина } \mathfrak{S}\}.$$

Пусть, как раньше, имеется множество $K \subsetneq I$ и индекс $i_0 \in I \backslash K$. Положим $L = K \cup \{i_0\}$. Если $K = \{i_1, \ldots, i_p\}$, то рассмотрим матрицу

$$\mathbf{H}_L = \begin{pmatrix} h_{i_0 1} & h_{i_0 2} & \ldots & h_{i_0 n} \\ h_{i_1 1} & h_{i_1 2} & \ldots & h_{i_1 n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{i_p 1} & h_{i_p 2} & \ldots & h_{i_p n} \end{pmatrix},$$

которая является проверочной матрицей кода $\mathcal{C}_L$ и получается из матрицы $\mathbf{H}$ удалением всех строк, индексы которых не принадлежат множеству $L$ (а также, возможно, изменением порядка строк).

Граф Таннера матрицы $\mathbf{H}_L$ обозначим $\mathfrak{T}_L$. Мы будем естественным образом отождествлять его с подграфом в $\mathfrak{T}$, получающимся из последнего удалением всех проверочных вершин, индексы которых не принадлежат множеству $L$, и всех ребер, инцидентных этим вершинам. Подграфом, *порожденым в* $\mathfrak{T}_L$ *вектором* $\mathbf{a} \in \mathbb{F}_2^n$, будем называть подграф $\mathfrak{T}_L^{\mathbf{a}}$ в $\mathfrak{T}_L$ такой, что

$$I(\mathfrak{T}_L^{\mathbf{a}}) = \bigcup_{j \in \operatorname{supp} \mathbf{a}} L \cap I_j \quad \text{и} \quad J(\mathfrak{T}_L^{\mathbf{a}}) = \operatorname{supp} \mathbf{a}.$$

Подграф, порожденный в $\mathfrak{T}_L$ вектором из кода $\mathcal{C}_L$, мы будем называть *кодовым в* $\mathfrak{T}_L$ (см. рисунок 3).

Отсюда и до конца этого раздела предполагается, что граф $\mathfrak{T}$ не содержит циклов. В этом случае все связные компоненты графа $\mathfrak{T}_L$ являются деревьями. Ту из них, что содержит вершину $f_{i_0}$, обозначим $\mathfrak{T}_{K,i_0}$. Мы начнем с того, что дадим описание множества $\mathcal{I}_{K,i_0}$ в терминах подграфов графа $\mathfrak{T}_L$. Затем, используя это описание, мы построим алгоритм, вычисляющий оптимальное решение задачи (3).

**Предложение 10.** *Множество* $\mathcal{I}_{K,i_0}$ *состоит в точности из всех векторов* $\mathbf{u} \in \mathbb{F}_2^n$ *таких, что подграф* $\mathfrak{T}_L^{\mathbf{u}}$ *является деревом, в котором все проверочные вершины с индексами из множества* $K$ *имеют степень 2, а вершина* $f_{i_0}$ *имеет степень 1.*

**Доказательство.** Пусть вектор $\mathbf{u} \in \mathcal{I}_{K,i_0}$ порождает в $\mathfrak{T}_L$ подграф $\mathfrak{T}_L^{\mathbf{u}}$. Допустим, что он не является связным и, стало быть, имеет связную компоненту, которая не содержит вершину $f_{i_0}$. Обозначим эту связную компоненту $\mathfrak{S}$. Докажем, что она является кодовым в $\mathfrak{T}_L$ подграфом. Действительно, имеем

$$I(\mathfrak{S}) = \bigcup_{j \in J(\mathfrak{S})} L \cap I_j.$$

Следовательно, подграф $\mathfrak{S}$ порожден в $\mathfrak{T}_L$ вектором $\mathbf{a} \in \mathbb{F}_2^n$ с носителем $\operatorname{supp} \mathbf{a} = J(\mathfrak{S})$. Для всех $i \in I(\mathfrak{S})$ имеем $J_i \cap \operatorname{supp} \mathbf{a} = J_i \cap \operatorname{supp} \mathbf{u}$. Но так как $\mathbf{u} \in \mathcal{C}_{K,i_0}$, для всех $i \in K$ множество $J_i \cap \operatorname{supp} \mathbf{u}$ содержит четное число элементов. По предположению $i_0 \notin I(\mathfrak{S})$, то есть $I(\mathfrak{S}) \subset K$. Поэтому для всех $i \in I(\mathfrak{S})$ множество $J_i \cap \operatorname{supp} \mathbf{a}$ содержит

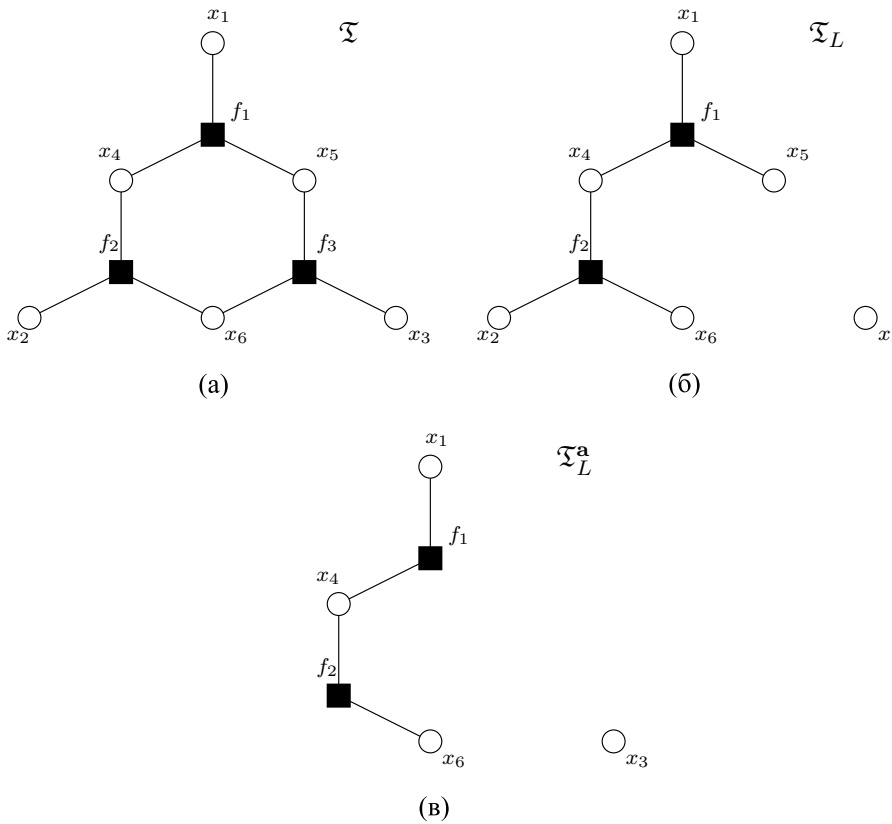


Рис. 3. Пример графа Таннера $\mathfrak{T}$ (а), его подграфа $\mathfrak{T}_L$ при $L = \{1, 2\}$ (б) и подграфа $\mathfrak{T}_L^{\mathbf{a}}$, порожденного в $\mathfrak{T}_L$ вектором $\mathbf{a} = (1\,0\,1\,1\,0\,1)$ (в).

четное число элементов. Это и означает, что подграф $\mathfrak{S}$ является кодовым в $\mathfrak{T}_L$ и $\mathbf{a} \in \mathcal{C}_L$. Поскольку $\operatorname{supp} \mathbf{a} \subsetneq \operatorname{supp} \mathbf{u}$, имеем противоречие с тем, что $\mathbf{u} \in \operatorname{irr} \mathcal{C}_L$. Этим доказано, что граф $\mathfrak{T}_L^{\mathbf{u}}$ связный (является деревом).

Обозначим $\mathfrak{S} = \mathfrak{T}_L^{\mathbf{u}}$. Для всех $i \in I(\mathfrak{S}) \cap K$ множество $J_i \cap \operatorname{supp} \mathbf{u}$ непусто и содержит четное число элементов. Для каждого $j \in J_i \cap \operatorname{supp} \mathbf{u}$ обозначим через $\mathfrak{S}_j$ наибольшее поддерево в $\mathfrak{S}$, содержащее вершину $x_j$, но не содержащее вершину $f_i$. Если $|J_i \cap \operatorname{supp} \mathbf{u}| > 2$, то найдутся такие $j_1, j_2 \in J_i \cap \operatorname{supp} \mathbf{u}$, что поддеревья $\mathfrak{S}_{j_1}$ и $\mathfrak{S}_{j_2}$ не

содержат вершину $f_{i_0}$. Тогда поддерево в $\mathfrak{S}$, образуемое вершиной $f_i$ и поддеревьями $\mathfrak{S}_{j_1}$ и $\mathfrak{S}_{j_2}$ является кодовым в $\mathfrak{T}_L$. Это противоречит $\mathcal{C}_L$-неприводимости $\mathbf{u}$, так что $|J_i \cap \operatorname{supp}\mathbf{u}| = 2$. Что касается множества $J_{i_0} \cap \operatorname{supp}\mathbf{u}$, то оно также непусто и содержит нечетное число элементов. Используя аналогичное рассуждение, можно показать, что $|J_{i_0} \cap \operatorname{supp}\mathbf{u}| = 1$.

Обратно, пусть подграф $\mathfrak{S}$ порожден в $\mathfrak{T}_L$ вектором $\mathbf{u} \in \mathbb{F}_2^n$ и удовлетворяет перечисленным условиям. Поскольку $|J_i \cap \operatorname{supp}\mathbf{u}| = 2$ для всех $i \in I(\mathfrak{S}) \cap K$ и $|J_{i_0} \cap \operatorname{supp}\mathbf{u}| = 1$, то $\mathbf{u} \in \mathcal{C}_{K,i_0}$. Пусть $\mathbf{c} \in \mathcal{C}_L \backslash \{\mathbf{0}\}$ и $\operatorname{supp}\mathbf{c} \subset \operatorname{supp}\mathbf{u}$. Положим $J_{i_0} \cap \operatorname{supp}\mathbf{u} = \{j_0\}$ и $j_1 \in \operatorname{supp}\mathbf{c}$. По предположению граф $\mathfrak{S}$ связный, поэтому в нем существует единственная простая цепь, соединяющая вершины $x_{j_1}$ и $x_{j_0}$, образуемая некоторой последовательностью вершин $x_{j_1}, f_{i_1}, x_{j_2}, f_{i_2}, \ldots, x_{j_p}, f_{i_p}, x_{j_0}$. При этом, среди индексов $i_1, \ldots, i_p$ не может быть индекса $i_0$. Далее, поскольку $j_1 \in \operatorname{supp}\mathbf{c}$ и $|J_{i_1} \cap \operatorname{supp}\mathbf{u}| = 2$, имеем $j_2 \in \operatorname{supp}\mathbf{c}$. Применяя это рассуждение последовательно ко всем индексам $j_2, \ldots, j_p$, получаем $j_0 \in \operatorname{supp}\mathbf{c}$. Следовательно, $J_{i_0} \cap \operatorname{supp}\mathbf{c} = \{j_0\}$, что противоречит условию $\mathbf{c} \in \mathcal{C}_L$. Поэтому вектор $\mathbf{u}$ является $\mathcal{C}_L$-неприводимым и, стало быть, $\mathbf{u} \in \mathcal{I}_{K,i_0}$. Предложение доказано.

Итак, любой вектор из $\mathcal{I}_{K,i_0}$ порождает в графе $\mathfrak{T}_L$ поддерево, содержащееся в $\mathfrak{T}_{K,i_0}$, в котором все проверочные вершины кроме $f_{i_0}$ имеют степень 2, а сама вершина $f_{i_0}$ является концевой. И обратно, любой вектор, порождающий в $\mathfrak{T}_L$ такое поддерево, принадлежит $\mathcal{I}_{K,i_0}$. Нашей следующей целью становится построение процедуры, которая производила бы вычисление такого поддерева, обеспечивая при этом оптимальность получаемого решения в смысле задачи (3). Всюду далее мы рассматриваем дерево $\mathfrak{T}_{K,i_0}$ как корневое с корнем в вершине $f_{i_0}$.

**Оптимизация на множестве $\mathcal{I}_{K,i_0}$.** Снабдим граф $\mathfrak{T}$ дополнительной структурой следующим образом.

1) Каждой проверочной вершине $f_i$ и символьной вершине $x_j$ в $\mathfrak{T}$ припишем индексы $p(f_i) \in J_i$ и $p(x_j) \in I_j$ соответственно.
2) Каждой проверочной вершине $f_i$ и символьной вершине $x_j$ в $\mathfrak{T}$ припишем числа $c(f_i) \in \mathbb{Z}$ и $c(x_j) \in \mathbb{Z}$ соответственно.

3) Каждой проверочной вершине $f_i$ и символьной вершине $x_j$ в $\mathfrak{T}$ припишем веса $w(f_i) \in \mathbb{R} \cup \{+\infty\}$ и $w(x_j) \in \mathbb{R} \cup \{+\infty\}$ соответственно.
4) Каждой проверочной вершине $f_i$ в $\mathfrak{T}$ припишем индекс $s(f_i) \in J_i$.
5) Каждой символьной вершине $x_j$ в $\mathfrak{T}$ припишем значение $v(x_j) \in \mathbb{F}_2$.

Для всех представленных в этом разделе алгоритмов предполагается, что они используют эту дополнительную структуру графа $\mathfrak{T}$ в своих вычислениях. Также будем считать, что все процедуры используют общий целочисленный стек $S$, предназначенный для операций с индексами вершин графа $\mathfrak{T}$. Отметим, что некоторые из перечисленных выше переменных могут принимать «пустое значение». Мы будем обозначать его литералом *пусто*.

**Алгоритм 2.** *Алгоритм* `BuildTree` *имеет следующее определение.*

***Параметры:*** *множество* $K \subsetneq I$ *и индекс* $i_0 \in I \backslash K$.
***Результат:*** *вычисление значений* $p(\cdot)$ *и* $c(\cdot)$.
***Псевдокод:***

```
Для всех i ∈ I цикл
    p(f_i) ← пусто; c(f_i) ← пусто;
Конец цикла;
Для всех j ∈ J цикл
    p(x_j) ← пусто; c(x_j) ← пусто;
Конец цикла;
Очистить стек S; S ↶ i_0;
Пока стек S не пустой, цикл
    u ↶ S;
    Если u ⩽ m, то
        i ← u; c ← 0;
        Для всех j ∈ J_i таких, что j ≠ p(f_i), цикл
            p(x_j) ← i; S ↶ m + j; c ← c + 1;
        Конец цикла;
        c(f_i) ← c;
    Иначе
        j ← u − m; c ← 0;
        Для всех i ∈ I_j ∩ K таких, что i ≠ p(x_j), цикл
            p(f_i) ← j; S ↶ i; c ← c + 1;
```

*Конец цикла;*
$c(x_j) \leftarrow c$;
*Конец если;*
*Конец цикла.*

Задача алгоритма `BuildTree` в том, чтобы вычислить для каждой вершины корневого дерева $\mathfrak{T}_{K,i_0}$ указатель на ее родительскую вершину, а также сосчитать количество ее дочерних вершин. Алгоритм, как видно из описания, представляет собой обычную процедуру обхода дерева «в глубину» с использованием стека. На этапе инициализации всем переменным $p(\cdot)$ и $c(\cdot)$ присваивается пустое значение, после чего в стек $S$ добавляется указатель на корень $f_{i_0}$.

Особенность данной реализации алгоритма состоит в том, что граф $\mathfrak{T}$ является двудольным, и для нумерации его вершин используется два различных, но пересекающихся индексных множества. Для того, чтобы обойти дерево $\mathfrak{T}_{K,i_0}$, используя общий стек, нам потребовалось ввести уникальный индекс, которым нумеруются все вершины графа $\mathfrak{T}$, и который вычисляется из индексов $i \in I$ и $j \in J$ по следующему правилу: если $i$ — индекс проверочной вершины, то $u = i$, а если $j$ — индекс символьной вершины, то $u = m + j$. Это правило используется при помещении индекса вершины в стек, а при извлечении из стека уникальный номер разыменовывается в индекс соответствующей вершины по правилу: если $u \leqslant m$, то $u$ — уникальный номер проверочной вершины с индексом $i = u$, а если $u > m$, то $u$ — уникальный номер символьной вершины с индексом $j = u - m$.

Итак, алгоритм обходит все вершины дерева $\mathfrak{T}_{K,i_0}$, начиная с корня. При этом, каждая вершина этого дерева попадает в стек и извлекается из него ровно один раз. Важно отметить, что если некоторая вершина помещена в стек, то соответствующая ей переменная $p(\cdot)$ уже содержит ссылку на ее родительскую вершину (за исключением вершины $f_{i_0}$, у которой значение $p(f_{i_0})$ пусто). При извлечении из стека каждая вершина обрабатывается следующим образом.

Если это проверочная вершина $f_i$, то алгоритм обходит все ее дочерние символьные вершины, используя индексы $j \in J_i$ такие, что $j \neq p(f_i)$. Для каждой дочерней вершины $x_j$ алгоритм прописывает в переменную $p(x_j)$ индекс ее родительской вершины, который равен $i$, после чего добавляет ее в стек, используя уникальный номер

$u = m + j$. Если, напротив, это символьная вершина $x_j$, то алгоритм обходит все ее дочерние проверочные вершины, используя индексы $i \in I_j \cap K$ такие, что $i \neq p(x_j)$. Для каждой дочерней вершины $f_i$ алгоритм прописывает в переменную $p(f_i)$ индекс ее родительской вершины, который равен $j$, после чего добавляет ее в стек, используя уникальный номер $u = i$. В обоих случаях алгоритм подсчитывает количество дочерних вершин, а затем сохраняет его в соответствующей переменной $c(\cdot)$. Этот процесс изображен на рисунке 4. Из описания алгоритма `BuildTree` следует

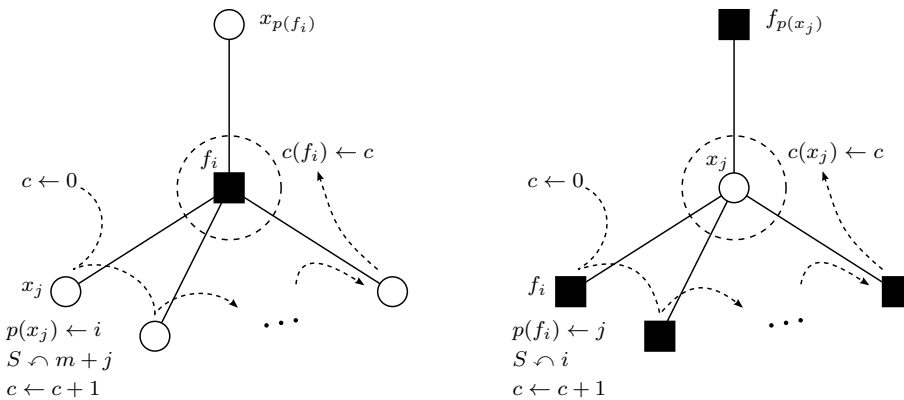


Рис. 4. Обработка вершин дерева $\mathfrak{T}_{K,i_0}$ алгоритмом `BuildTree`.

**Предложение 11.** *После выполнения алгоритма* `BuildTree` *с параметрами $K$ и $i_0$ будут справедливы утверждения:*

1) *Для всех $i \in I(\mathfrak{T}_{K,i_0})$ значение $c(f_i)$ равно количеству дочерних по отношению к $f_i$ вершин в дереве $\mathfrak{T}_{K,i_0}$. При этом, если $i \neq i_0$, то значение $p(f_i)$ равно индексу родительской символьной вершины для $f_i$, а значение $p(f_{i_0})$ пусто.*
2) *Для всех $j \in J(\mathfrak{T}_{K,i_0})$ значение $c(x_j)$ равно количеству дочерних по отношению к $x_j$ вершин в дереве $\mathfrak{T}_{K,i_0}$, тогда как значение $p(x_j)$ равно индексу родительской проверочной вершины для $x_j$.*
3) *Для всех других вершин графа $\mathfrak{T}$ значения $p(\cdot)$ и $c(\cdot)$ являются пустыми.*

Вычисленные алгоритмом `BuildTree` значения $p(\cdot)$ и $c(\cdot)$ по сути носят вспомогательный характер и предназначены для использования в следующей процедуре.

**Алгоритм 3.** *Алгоритм* `Optimize` *имеет следующее определение.*

***Параметры:*** *вектор* $\boldsymbol{\gamma} \in \mathbb{R}^n$.

***Результат:*** *вычисление значений* $w(\cdot)$ *и* $s(\cdot)$.

***Псевдокод:***

```
Очистить стек S;
Для всех i ∈ I цикл
    w(f_i) ← +∞; s(f_i) ← пусто;
Конец цикла;
Для всех j ∈ J цикл
    w(x_j) ← γ_j;
    Если p(x_j) ≠ пусто и c(x_j) = 0, то
        S ↶ m + j;
    Конец если;
Конец цикла;
Пока стек S не пустой, цикл
    u ↶ S;
    Если u > m, то
        j ← u − m; i ← p(x_j);
        Если w(x_j) < w(f_i), то
            w(f_i) ← w(x_j); s(f_i) ← j;
        Конец если;
        c(f_i) ← c(f_i) − 1;
        Если c(f_i) = 0, то
            S ↶ i;
        Конец если;
    Иначе
        i ← u; j ← p(f_i);
        Если j = пусто, то
            Прервать цикл;
        Конец если;
        w(x_j) ← w(x_j) + w(f_i);
        c(x_j) ← c(x_j) − 1;
        Если c(x_j) = 0, то
            S ↶ m + j;
        Конец если;
```

*Конец если;*
*Конец цикла.*

Рассмотрим в деталях логику данного алгоритма. Будем предполагать, что значения переменных $p(\cdot)$ и $c(\cdot)$ уже вычислены при помощи процедуры `BuildTree` с параметрами $K$ и $i_0$. Фиксируем некоторый вектор $\boldsymbol{\gamma} \in \mathbb{R}^n$. Итак, алгоритм `Optimize` начинает выполнение с инициализации значений $w(\cdot)$ и $s(\cdot)$. Здесь всем переменным $w(f_i)$ присваивается значение $+\infty$, всем переменным $w(x_j)$ присваивается значение соответствующей компоненты $\gamma_j$ вектора $\boldsymbol{\gamma}$, а всем переменным $s(f_i)$ присваивается пустое значение. После этого все концевые вершины дерева $\mathfrak{T}_{K,i_0}$ помещаются в стек $S$. Поскольку в дереве $\mathfrak{T}$ по нашему предположению нет концевых проверочных вершин, то таковых нет и в $\mathfrak{T}_{K,i_0}$. Поэтому концевыми вершинами дерева $\mathfrak{T}_{K,i_0}$ могут быть только символьные вершины $x_j$, у которых значение $p(x_j)$ не пусто, и количество дочерних вершин равно нулю, то есть $c(x_j) = 0$. Таким образом, все символьные вершины, удовлетворяющие этим условиям, помещаются в стек $S$. Как мы видим, для операций со стеком $S$ алгоритм `Optimize` использует тот же способ уникальной нумерации, что и алгоритм `BuildTree`.

В основном цикле алгоритм `Optimize` выполняет обход дерева $\mathfrak{T}_{K,i_0}$, начиная от его концевых вершин и заканчивая корнем $f_{i_0}$. Каждая вершина этого дерева помещается в стек $S$ и извлекается из него ровно один раз. Если из стека извлекается символьная вершина $x_j$, и значение $w(x_j)$ меньше текущего значения $w(f_i)$, где $i = p(x_j)$, то в переменной $w(f_i)$ запоминается значение $w(x_j)$, а в переменной $s(f_i)$ — индекс $j$. При этом, в любом случае переменная $c(f_i)$ уменьшается на единицу. Это гарантирует нам, что вершина $f_i$ будет помещена в стек сразу после того, как все ее дочерние символьные вершины будут обработаны. Если теперь из стека извлекается проверочная вершина $f_i$, то значение $w(f_i)$ просто прибавляется к текущему значению $w(x_j)$, где $j = p(f_i)$. При этом, переменная $c(x_j)$ уменьшается на единицу. Это, как и выше, гарантирует нам, что вершина $x_j$ будет помещена в стек, как только все ее дочерние проверочные вершины будут обработаны. Наконец, если из стека извлечена вершина $f_{i_0}$, то цикл прерывается, и алгоритм завершает работу.

Для всех $i \in I(\mathfrak{T}_{K,i_0})$ и $j \in J(\mathfrak{T}_{K,i_0})$ обозначим через $J_i^*$ и $I_j^*$ множества индексов дочерних вершин в дереве $\mathfrak{T}_{K,i_0}$ соответственно для $f_i$ и $x_j$:

$$J_i^* = \{j' \in J_i \mid j' \neq p(f_i)\}, \quad I_j^* = \{i' \in I_j \cap K \mid i' \neq p(x_j)\}.$$

Мы видим, что в процессе выполнения алгоритма `Optimize` каждая вершина дерева $\mathfrak{T}_{K,i_0}$ аккумулирует информацию, получаемую от своих дочерних вершин: в переменных $w(f_i)$ и $s(f_i)$ вычисляются вес и индекс дочерней для $f_i$ символьной вершины с минимальным весом, а в переменной $w(x_j)$ вычисляется сумма $\gamma_j$ с весами всех дочерних для $x_j$ проверочных вершин (см. рисунок 5). Получаем

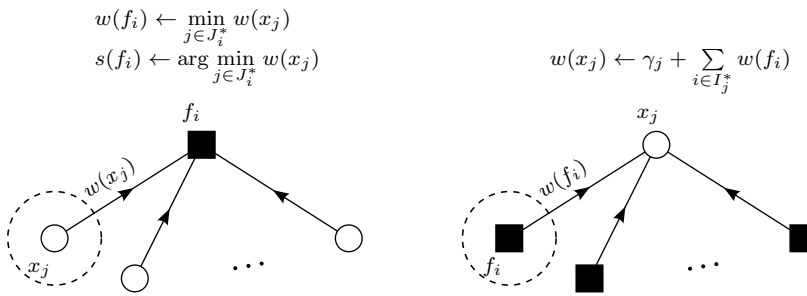


Рис. 5. Обработка вершин дерева $\mathfrak{T}_{K,i_0}$ алгоритмом `Optimize`.

**Предложение 12.** *Пусть значения переменных $p(\cdot)$ и $c(\cdot)$ вычислены алгоритмом* `BuildTree` *с параметрами $K$ и $i_0$. Тогда после выполнения алгоритма* `Optimize` *с параметром $\boldsymbol{\gamma} \in \mathbb{R}^n$ будут справедливы утверждения:*

*1) Для всех индексов $i \in I(\mathfrak{T}_{K,i_0})$ имеем*

$$w(f_i) = \min_{j \in J_i^*} w(x_j) \quad \text{и} \quad s(f_i) = \arg\min_{j \in J_i^*} w(x_j).$$

*2) Для всех индексов $j \in J(\mathfrak{T}_{K,i_0})$ имеем*

$$w(x_j) = \gamma_j + \sum_{i \in I_j^*} w(f_i).$$

Для всех индексов $i \in I(\mathfrak{T}_{K,i_0})$ и $j \in J(\mathfrak{T}_{K,i_0})$ обозначим через $\mathfrak{S}_{f_i}$ и $\mathfrak{S}_{x_j}$ корневые поддеревья в $\mathfrak{T}_{K,i_0}$ с корнями в вершинах $f_i$ и $x_j$ соответственно. Также положим

$$\mathcal{I}_{f_i} = \{\mathbf{u} \in \mathcal{I}_{K,i_0} \mid i \in I(\mathfrak{T}_L^{\mathbf{u}})\}, \quad \mathcal{I}_{x_j} = \{\mathbf{u} \in \mathcal{I}_{K,i_0} \mid j \in J(\mathfrak{T}_L^{\mathbf{u}})\}.$$

**Предложение 13.** *Пусть значения переменных $p(\cdot)$ и $c(\cdot)$ вычислены алгоритмом* `BuildTree` *с параметрами $K$ и $i_0$. Тогда после выполнения алгоритма* `Optimize` *с параметром $\boldsymbol{\gamma} \in \mathbb{R}^n$ для всех индексов $i \in I(\mathfrak{T}_{K,i_0})$ и $j \in J(\mathfrak{T}_{K,i_0})$ будем иметь*

$$w(f_i) = \min_{\mathbf{u} \in \mathcal{I}_{f_i}} \left\{ \sum_{j' \in J(\mathfrak{S}_{f_i}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right\}, \quad w(x_j) = \min_{\mathbf{u} \in \mathcal{I}_{x_j}} \left\{ \sum_{j' \in J(\mathfrak{S}_{x_j}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right\}.$$

**Доказательство.** Обозначим правые части доказываемых равенств как $\hat{w}(f_i)$ и $\hat{w}(x_j)$. Пусть $x_j$ — произвольная концевая вершина дерева $\mathfrak{T}_{K,i_0}$ и $I_j \cap L = \{i\}$. Тогда $J(\mathfrak{S}_{x_j}) \cap \operatorname{supp} \mathbf{u} = \{j\}$ для любого вектора $\mathbf{u} \in \mathcal{I}_{x_j}$ и, следовательно, $\hat{w}(x_j) = \gamma_j$.

Пусть теперь $x_j$ — произвольная неконцевая символьная вершина дерева $\mathfrak{T}_{K,i_0}$. Для любого вектора $\mathbf{u} \in \mathcal{I}_{x_j}$ имеем

$$\sum_{j' \in J(\mathfrak{S}_{x_j}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} = \gamma_j + \sum_{i' \in I_j^*} \left( \sum_{j' \in J(\mathfrak{S}_{f_{i'}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right),$$

откуда

$$\hat{w}(x_j) = \gamma_j + \min_{\mathbf{u} \in \mathcal{I}_{x_j}} \left\{ \sum_{i' \in I_j^*} \left( \sum_{j' \in J(\mathfrak{S}_{f_{i'}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right) \right\}.$$

Так как поддеревья $\mathfrak{S}_{f_{i'}}$ при $i' \in I_j^*$ не пересекаются, то найдется такой вектор $\mathbf{u}_0 \in \mathcal{I}_{x_j}$, что для всех $i' \in I_j^*$ справедливо равенство

$$\sum_{j' \in J(\mathfrak{S}_{f_{i'}}) \cap \operatorname{supp} \mathbf{u}_0} \gamma_{j'} = \min_{\mathbf{u} \in \mathcal{I}_{x_j}} \left\{ \sum_{j' \in J(\mathfrak{S}_{f_{i'}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right\}.$$

Ясно, что для всех $i' \in I_j^*$ имеем $\mathcal{I}_{f_{i'}} = \mathcal{I}_{x_j}$. Следовательно,

$$\hat{w}(x_j) = \gamma_j + \sum_{i' \in I_j^*} \left( \min_{\mathbf{u} \in \mathcal{I}_{x_j}} \left\{ \sum_{j' \in J(\mathfrak{S}_{f_{i'}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j'} \right\} \right) = \gamma_j + \sum_{i' \in I_j^*} \hat{w}(f_{i'}).$$

Рассмотрим произвольный вектор $\mathbf{u} \in \mathcal{I}_{f_i}$. В соответствии с предложением 10 множество $J_i^* \cap \operatorname{supp} \mathbf{u}$ содержит ровно один элемент. Положим $J_i^* \cap \operatorname{supp} \mathbf{u} = \{j_0\}$, так что $J(\mathfrak{S}_{f_i}) \cap \operatorname{supp} \mathbf{u} = J(\mathfrak{S}_{x_{j_0}}) \cap \operatorname{supp} \mathbf{u}$ и, стало быть,

$$\sum_{j'' \in J(\mathfrak{S}_{f_i}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j''} = \sum_{j'' \in J(\mathfrak{S}_{x_{j_0}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j''},$$

откуда

$$\hat{w}(f_i) = \min_{j' \in J_i^*} \left\{ \min_{\mathbf{u} \in \mathcal{I}_{x_{j'}}} \left\{ \sum_{j'' \in J(\mathfrak{S}_{x_{j'}}) \cap \operatorname{supp} \mathbf{u}} \gamma_{j''} \right\} \right\} = \min_{j' \in J_i^*} \hat{w}(x_{j'}).$$

Таким образом, величины $\hat{w}(f_i)$ и $\hat{w}(x_j)$ соотносятся между собой так же, как и величины $w(f_i)$ и $w(x_j)$ в предложении 12. Но так как они определяются этими соотношениями однозначно, то $\hat{w}(f_i) = w(f_i)$ и $\hat{w}(x_j) = w(x_j)$ для всех $i \in I(\mathfrak{T}_{K,i_0})$ и $j \in J(\mathfrak{T}_{K,i_0})$. Предложение доказано.

Теперь рассмотрим следующий алгоритм.

**Алгоритм 4.** *Алгоритм* `BuildSolution` *имеет следующее определение.*

***Параметры:*** *множество* $K \subsetneq I$ *и индекс* $i_0 \in I \backslash K$.
***Результат:*** *вычисление значений* $v(\cdot)$.
***Псевдокод:***

*Для всех* $j \in J$ *цикл*
    $v(x_j) \leftarrow 0$;
*Конец цикла;*
*Очистить стек* $S$; $S \curvearrowleft i_0$;
*Пока стек* $S$ *не пустой, цикл*

$i \curvearrowleft S$; $j \leftarrow s(f_i)$;
$v(x_j) \leftarrow 1$;
*Для всех* $i' \in I_j \cap K$ *таких, что* $i' \neq i$, *цикл*
$S \curvearrowleft i'$;
*Конец цикла;*
*Конец цикла.*

Будем считать, что значения $p(\cdot)$ и $c(\cdot)$ уже вычислены при помощи процедуры `BuildTree` с параметрами $K$ и $i_0$, а значения $w(\cdot)$ и $s(\cdot)$ вычислены при помощи процедуры `Optimize` с параметром $\boldsymbol{\gamma} \in \mathbb{R}^n$. Итак, алгоритм `BuildSolution` инициализирует все переменные $v(x_j)$ значением 0 и помещает индекс проверочной вершины $f_{i_0}$ в стек. В основном цикле алгоритма каждая проверочная вершина $f_i$, извлеченная из стека, обрабатывается так: вычисляется индекс $j = s(f_i)$ одной из ее дочерних символьных вершин, и переменной $v(x_j)$ присваивается значение 1, после чего все дочерние для $x_j$ проверочные вершины помещаются в стек (см. рисунок 6). Наконец, рассмотрим

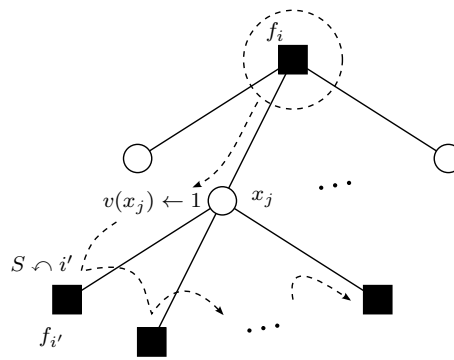


Рис. 6. Обработка вершин дерева $\mathfrak{T}_{K,i_0}$ алгоритмом `BuildSolution`.

**Алгоритм 5.** *Алгоритм* `Irreducible`$_1$ *имеет следующее определение.*

***Параметры:*** *вектор* $\boldsymbol{\gamma} \in \mathbb{R}^n$, *множество* $K \subsetneq I$ *и индекс* $i_0 \in I \backslash K$.
***Результат:*** *вектор* $\mathbf{u} \in \mathbb{F}_2^n$.
***Псевдокод:***

`BuildTree`$(K, i_0)$`;`
`Optimize`$(\boldsymbol{\gamma})$`;`
`BuildSolution`$(K, i_0)$`;`
*Возврат* $(v(x_j))$.

Результат выполнения алгоритма $\texttt{Irreducible}_1$ с параметрами $\boldsymbol{\gamma}$, $K$ и $i_0$ обозначим $\texttt{Irreducible}_1(\boldsymbol{\gamma}, K, i_0)$. Из предложений 10, 12 и 13, а также описания алгоритмов `BuildSolution` и $\texttt{Irreducible}_1$ вытекает

**Следствие 14.** *Результат* $\mathbf{u} = \texttt{Irreducible}_1(\boldsymbol{\gamma}, K, i_0)$ *является оптимальным решением задачи (3).*

Алгоритм $\texttt{Decoder}_{\texttt{A}}$ с подпроцедурой $\texttt{A} = \texttt{Irreducible}_1$ обозначим просто $\texttt{Decoder}_1$.

**Предложение 15.** *Временная сложность алгоритма* $\texttt{Decoder}_1$ *лежит в классе* $O(n^2)$ *при* $n \to \infty$, *тогда как его пространственная сложность лежит в классе* $O(n)$ *при* $n \to \infty$.

**Доказательство.** Для начала заметим, что число ребер в дереве $\mathfrak{T}$ равно $m + n - 1$. Поэтому объем памяти, необходимый для хранения матрицы $\mathbf{H}$ в виде списков смежности $J_i$ и $I_j$, составляет $O(n)$, $n \to \infty$. Кроме того, алгоритмы `BuildTree`, `Optimize` и `BuildSolution` используют общую структуру данных, ассоциированную с графом $\mathfrak{T}$ (см. стр. 251) и стек $S$, предназначенный для операций с индексами вершин графа $\mathfrak{T}$. Ясно, что объем памяти, необходимый для хранения этих структур, зависит линейно от величины $m + n$.

Сверх этого алгоритм $\texttt{Decoder}_1$ использует вектор $\boldsymbol{\lambda}$, являющийся его параметром, и векторы $\mathbf{c}$ и $\mathbf{u}$, содержащие промежуточные решения, а также вектор $\boldsymbol{\gamma}$ и описание множества $K$, передаваемые субалгоритмам в качестве параметров. Объем памяти, необходимый для хранения этих данных, также составляет $O(n)$, $n \to \infty$. Итак, мы видим, что пространственная сложность алгоритма $\texttt{Decoder}_1$ действительно лежит в классе $O(n)$ при $n \to \infty$.

Инициализация алгоритма $\texttt{Decoder}_1$ занимает время порядка $O(n)$, $n \to \infty$. Что касается алгоритмов `BuildTree`, `Optimize` и `BuildSolution`, то во всех этих процедурах каждая вершина графа

$\mathfrak{T}$ помещается в стек $S$ и извлекается из него не более одного раза. Зная это, нетрудно понять, что время, необходимое для выполнения этих субалгоритмов (а значит и алгоритма $\texttt{Irreducible}_1$), составляет $O(n)$, $n \to \infty$. Стоимость других операций, выполняемых внутри основного цикла алгоритма $\texttt{Decoder}_1$, зависит линейно от $n$. Поэтому общее время, затрачиваемое на выполнение одной итерации основного цикла, также составляет $O(n)$, $n \to \infty$. А поскольку число таких итераций равно $m$, временная сложность алгоритма $\texttt{Decoder}_1$ лежит в классе $O(n^2)$ при $n \to \infty$. Предложение доказано.

Объединяя следствия 9, 14 и предложение 15, получаем теорему 1. Можно заметить, что при построении оптимального решения задачи (3) в случае, когда граф $\mathfrak{T}$ не содержит циклов, мы не пользовались дополнительным знанием о том, что $\langle \boldsymbol{\gamma}, \mathbf{c} \rangle \geqslant 0$ для всех $\mathbf{c} \in \mathcal{C}_K$, которое дает нам лемма 7. В следующем разделе мы представим алгоритм решения данной задачи для случая, когда граф $\mathfrak{T}$ может содержать циклы, но каждая его символьная вершина имеет степень не более 2. На этот раз положительная определенность формы $\langle \boldsymbol{\gamma}, \cdot \rangle$ будет иметь значение.

## 5. Граф $\mathfrak{T}$ с наибольшей левой степенью $\leqslant 2$

*Левыми* принято называть степени символьных вершин графа Таннера (в противоположность проверочным вершинам, степени которых называют *правыми*). Таким образом, до конца этого раздела мы будем предполагать, что степени всех символьных вершин в графе Таннера $\mathfrak{T}$ не превосходят 2.

**Описание множества $\mathcal{I}_{K,i_0}$.** Рассмотрим некоторую символьную вершину $x_j$ в графе $\mathfrak{T}$. Если ее степень равна двум, удалим эту вершину вместе с парой ребер, которые ей инцидентны, а пару ее смежных вершин соединим новым ребром, которое обозначим так же, как удаленную символьную вершину, то есть — $x_j$. Если степень вершины $x_j$ равна одному, то добавим в граф новую вершину $f$, соединив ее ребром с вершиной $x_j$, и удалим вершину $x_j$ описанным выше способом. Если вершина $x_j$ изолирована, то добавим в граф пару новых вер-

шин $f$ и $f'$, соединив каждую из них ребром с вершиной $x_j$, и также удалим вершину $x_j$ описанным способом. Подвергнем этой процедуре все символьные вершины в $\mathfrak{T}$ в порядке возрастания индекса $j$. Новые вершины пронумеруем натуральными индексами $m+1, m+2, \ldots$ в порядке их добавления. Полученный граф назовем *редукцией* графа $\mathfrak{T}$ и обозначим $\mathfrak{R}$. Вершины с индексами $i > m$ будем называть *мнимыми*, а остальные вершины — *действительными*. Пример такого преобразования изображен на рисунке 7 а.

Все действительные вершины графа $\mathfrak{R}$ соответствуют проверочным вершинам в $\mathfrak{T}$, а все ребра графа $\mathfrak{R}$ соответствуют символьным вершинам в $\mathfrak{T}$. При этом, наличие мнимой вершины на одном из концов ребра просто указывает на то, что соответствующая символьная вершина графа $\mathfrak{T}$ является концевой (или изолированной, если оба конца ребра являются мнимыми). Поэтому для всех $i \in I$ индексы ребер графа $\mathfrak{R}$, инцидентных вершине $f_i$, образуют множество $J_i$, а для всех $j \in J$ индексы действительных вершин графа $\mathfrak{R}$, инцидентных ребру $x_j$, образуют множество $I_j$, так что граф $\mathfrak{T}$ однозначно восстанавливается из своей редукции $\mathfrak{R}$. Для произвольного подграфа $\mathfrak{P}$ в $\mathfrak{R}$ положим

$$I(\mathfrak{P}) = \{i \in I \mid f_i \text{ — вершина } \mathfrak{P}\}, \quad J(\mathfrak{P}) = \{j \in J \mid x_j \text{ — ребро } \mathfrak{P}\}.$$

Напомним, что по нашему предположению в графе $\mathfrak{T}$ нет концевых проверочных вершин, а минимальное расстояние кода $\mathcal{C}$ больше или равно трем (см. стр. 242). Из этого следует, что, во-первых, в графе $\mathfrak{R}$ концевыми являются все мнимые вершины и только они, а во-вторых, граф $\mathfrak{R}$ не содержит параллельных ребер.

Как раньше, фиксируем некоторое подмножество $K \subsetneq I$ с индексом $i_0 \in I \backslash K$ и положим $L = K \cup \{i_0\}$. Редукцию графа $\mathfrak{T}_L$ обозначим $\mathfrak{R}_L$ (см. рисунок 7 б). Ее можно получить из графа $\mathfrak{R}$ путем замены каждой проверочной вершины, индекс которой не принадлежит множеству $L$, на набор мнимых вершин, по одной на каждое ребро, инцидентное данной проверочной вершине. Для произвольного $\mathbf{a} \in \mathbb{F}_2^n$ обозначим через $\mathfrak{R}_L^{\mathbf{a}}$ подграф в $\mathfrak{R}_L$, содержащий те и только те ребра $x_j$, для которых верно включение $j \in \operatorname{supp} \mathbf{a}$. Ясно, что он изоморфен редукции графа $\mathfrak{T}_L^{\mathbf{a}}$ (см. рисунок 7 в). Мы будем называть его

подграфом, *порожденным в* $\mathfrak{R}_L$ *вектором* $\mathbf{a}$. В случае, если $\mathbf{a} \in \mathcal{C}_L$, мы также будем называть его *кодовым в* $\mathfrak{R}_L$ подграфом.

Теперь мы дадим описание множества $\mathcal{I}_{K,i_0}$ в терминах подграфов графа $\mathfrak{R}_L$. А затем, используя это описание, предъявим алгоритм, вычисляющий оптимальное решение задачи (3).

**Предложение 16.** *Множество* $\mathcal{I}_{K,i_0}$ *состоит в точности из всех векторов* $\mathbf{u} \in \mathbb{F}_2^n$ *таких, что подграф* $\mathfrak{R}_L^{\mathbf{u}}$ *является простой цепью в* $\mathfrak{R}_L$*, которая соединяет вершину* $f_{i_0}$ *с некоторой мнимой вершиной.*

**Доказательство.** Пусть вектор $\mathbf{u} \in \mathbb{F}_2^n$ порождает в $\mathfrak{R}_L$ подграф $\mathfrak{P}$, который является простой цепью, соединяющей вершину $f_{i_0}$ с некоторой мнимой вершиной. Тогда $|J_i \cap \operatorname{supp} \mathbf{u}| = 2$ для всех $i \in I(\mathfrak{P}) \cap K$ и $|J_{i_0} \cap \operatorname{supp} \mathbf{u}| = 1$, так что $\mathbf{u} \in \mathcal{C}_{K,i_0}$.

Предположим, что существует ненулевой вектор $\mathbf{c} \in \mathcal{C}_L$ такой, что $\operatorname{supp} \mathbf{c} \subsetneq \operatorname{supp} \mathbf{u}$. Тогда подграф, порожденный в $\mathfrak{R}_L$ вектором $\mathbf{c}$, будет состоять, вообще говоря, из нескольких связных компонент, каждая из которых представляет собой некоторый фрагмент цепи $\mathfrak{P}$. Ясно, что по меньшей мере один из концов любого такого фрагмента должен быть действительной вершиной, отличной от $f_{i_0}$. Если $f_i$ — одна из таких вершин, то $i \in K$ и $|J_i \cap \operatorname{supp} \mathbf{c}| = 1$. Это противоречит тому, что $\mathbf{c} \in \mathcal{C}_L$. Поэтому $\mathbf{u} \in \mathcal{I}_{K,i_0}$.

Обратно, пусть вектор $\mathbf{u} \in \mathcal{I}_{K,i_0}$ порождает в $\mathfrak{R}_L$ подграф $\mathfrak{R}_L^{\mathbf{u}}$. Подграф $\mathfrak{T}_L^{\mathbf{u}}$, порождаемый в $\mathfrak{T}_L$ вектором $\mathbf{u}$, является связным, и доказательство этого факта дословно переносится из доказательства предложения 10. Поэтому граф $\mathfrak{R}_L^{\mathbf{u}}$, будучи редукцией графа $\mathfrak{T}_L^{\mathbf{u}}$, также является связным. Равенство $h_{i_0}(\mathbf{u}) = 1$ означает, что степень вершины $f_{i_0}$ в $\mathfrak{R}_L^{\mathbf{u}}$ нечетна. Согласно лемме о рукопожатиях должна существовать по меньшей мере еще одна вершина, степень которой в $\mathfrak{R}_L^{\mathbf{u}}$ нечетна. Поскольку $\mathbf{u} \in \mathcal{C}_K$, эта вершина должна быть мнимой. А так как граф $\mathfrak{R}_L^{\mathbf{u}}$ связный, в нем существует простая цепь $\mathfrak{P}$, которая соединяет эту мнимую вершину с $f_{i_0}$.

Рассмотрим вектор $\mathbf{u}'$ такой, что $\operatorname{supp} \mathbf{u}' = J(\mathfrak{P})$. Так как цепь $\mathfrak{P}$ содержится в $\mathfrak{R}_L^{\mathbf{u}}$, то $\operatorname{supp} \mathbf{u}' \subset \operatorname{supp} \mathbf{u}$. По доказанному вектор $\mathbf{u}'$ принадлежит $\mathcal{I}_{K,i_0}$. Если $\mathbf{u} \neq \mathbf{u}'$, то $\mathbf{u} = \mathbf{u}' \oplus \mathbf{c}$, где $\mathbf{c} \in \mathcal{C}_L$, $\mathbf{c} \neq \mathbf{0}$. Но это противоречит $\mathcal{C}_L$-неприводимости вектора $\mathbf{u}$, поэтому $\mathbf{u} = \mathbf{u}'$ и $\mathfrak{R}_L^{\mathbf{u}} = \mathfrak{P}$. Предложение доказано.

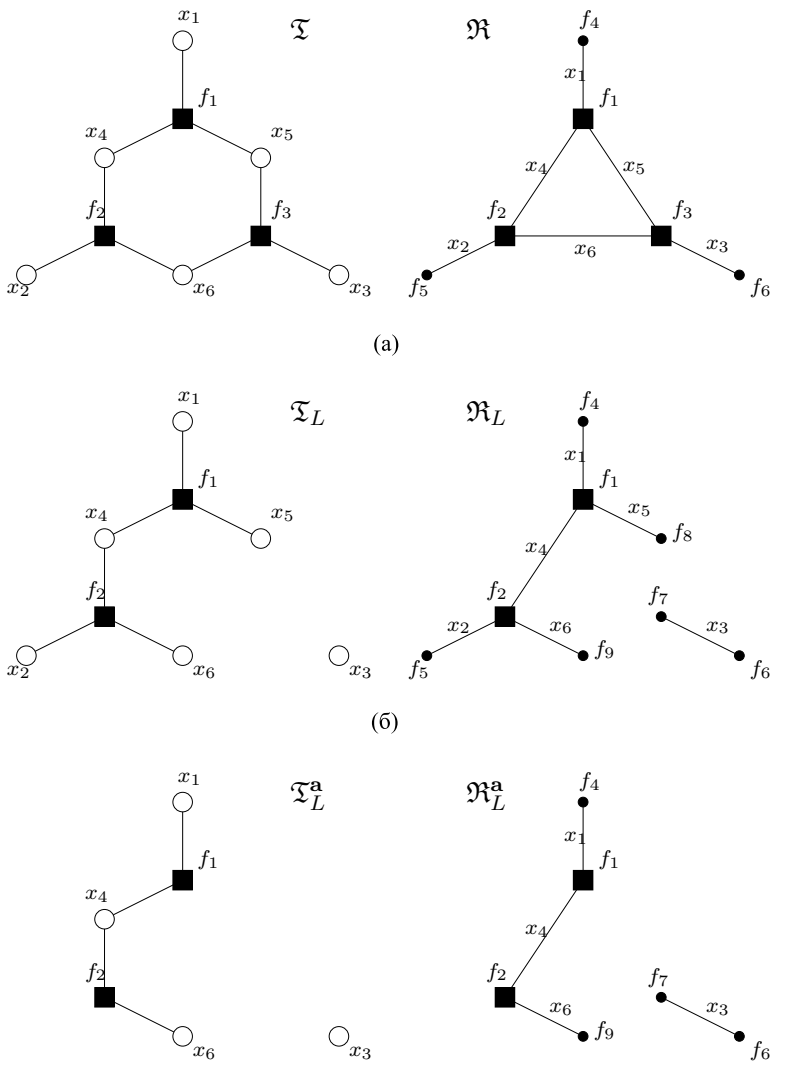


Рис. 7. Пример редукции $\mathfrak{R}$ исходного графа Таннера $\mathfrak{T}$ (а), редукции $\mathfrak{R}_L$ графа $\mathfrak{T}_L$ при $L = \{1, 2\}$ (б) и подграфа $\mathfrak{R}_L^{\mathbf{a}}$, порожденного в $\mathfrak{R}_L$ вектором $\mathbf{a} = (1\,0\,1\,1\,0\,1)$ (в).

Мы видим, что в терминах подграфов графа $\mathfrak{R}_L$ множество $\mathcal{I}_{K,i_0}$ получает очень простую характеризацию. Теперь мы можем построить алгоритм, вычисляющий оптимальное решение задачи (3).

**Оптимизация на множестве $\mathcal{I}_{K,i_0}$.** Фиксируем вектор $\boldsymbol{\gamma} \in \mathbb{R}^n$ такой, что $\langle \boldsymbol{\gamma}, \mathbf{c} \rangle \geqslant 0$ для всех $\mathbf{c} \in \mathcal{C}_K$. Обозначим через $(\mathfrak{R}_L, \boldsymbol{\gamma})$ взвешенный граф $\mathfrak{R}_L$, в котором каждому ребру $x_j$ приписан вес $\gamma_j$. Очевидно, что для любого вектора $\mathbf{a} \in \mathbb{F}_2^n$ величина $\langle \boldsymbol{\gamma}, \mathbf{a} \rangle$ равна весу подграфа $\mathfrak{R}_L^{\mathbf{a}}$ в графе $(\mathfrak{R}_L, \boldsymbol{\gamma})$. Из предложения 16 немедленно получаем

**Следствие 17.** *Вектор $\mathbf{u} \in \mathbb{F}_2^n$ является оптимальным решением задачи (3) тогда и только тогда, когда подграф $\mathfrak{R}_L^{\mathbf{u}}$ является кратчайшей простой цепью во взвешенном графе $(\mathfrak{R}_L, \boldsymbol{\gamma})$, соединяющей вершину $f_{i_0}$ с ближайшей мнимой вершиной.*

Таким образом, задача (3) сводится к *задаче о кратчайшей цепи в неориентированном взвешенном графе с произвольными весами*. Как известно, эта задача решается за полиномиальное время, если граф не содержит циклов отрицательного веса. Поэтому следующее утверждение, являющееся по сути прямым следствием леммы 7, устанавливает принципиальную возможность решения задачи (3) за полиномиальное время.

**Лемма 18.** *Граф $(\mathfrak{R}_L, \boldsymbol{\gamma})$ не содержит циклов с отрицательным весом.*

**Доказательство.** Поскольку $\mathcal{C}_L \subset \mathcal{C}_K$, то для всех $\mathbf{c} \in \mathcal{C}_L$ верно неравенство $\langle \boldsymbol{\gamma}, \mathbf{c} \rangle \geqslant 0$. Остается заметить, что всякий простой цикл в графе $\mathfrak{R}_L$ порождается вектором из $\mathcal{C}_L$, и что любой цикл распадается на совокупность простых циклов с непересекающимися множествами ребер. Лемма доказана.

Итак, нам лишь требуется приспособить для своих целей один из имеющихся алгоритмов решения *консервативной* (без циклов отрицательного веса) неориентированной задачи о кратчайшей цепи. Единственный известный на данное время полиномиальный метод

решения этой задачи заключается в сведении ее к *задаче о совершенном паросочетании минимальноого веса* [14, теорема 12.12]. Если число вершин и число ребер в графе равны соответственно $p$ и $q$, то последняя решается, например, с помощью алгоритма Эдмондса [14, гл. 11] за время порядка $O(p^3)$. Хотя, вообще говоря, самая быстрая известная на сегодня реализация этого алгоритма выполняется за время порядка $O(pq + p^2 \log p)$ [15]. Согласно следствию 17 нам нужно найти кратчайшую простую цепь в $(\mathfrak{R}_L, \boldsymbol{\gamma})$ среди всех простых цепей, соединяющих вершину $f_{i_0}$ с мнимыми вершинами. Для этого мы воспользуемся одной из версий названного алгоритма, которая решает консервативную неориентированную задачу о кратчайших цепях, *исходящих из одной вершины*.

Пусть имеется неориентированный граф $\mathfrak{G} = (V, E, \varepsilon)$, где $V$ и $E$ — соответственно множества его вершин и ребер, а $\varepsilon\colon E \to \binom{V}{2}$ — отображение, сопоставляющее каждому ребру из $E$ множество его концевых вершин. Далее, пусть имеется весовая функция $w\colon E \to \mathbb{R}$ такая, что граф $\mathfrak{G}$ не содержит циклов отрицательного веса. Наконец, пусть даны вершина $v_0 \in V$ и множество $U \subset V$ такие, что $v_0 \notin U$. Из доказательства теоремы 12.13 в [14] следует

**Теорема 19.** *Существует алгоритм (обозначим его* `ShortestPaths`*), находящий все кратчайшие простые цепи во взвешенном графе $\mathfrak{G}$ с весовой функцией $w$ без циклов отрицательного веса, соединяющие вершину $v_0$ с каждой из вершин множества $U$, либо обнаруживающий их отсутствие, который выполняется за время $O(|V|^3)$ при $|V| \to \infty$.*

Результат выполнения алгоритма `ShortestPaths` с параметрами $\mathfrak{G}$, $w$, $v_0$ и $U$, который мы обозначим $\texttt{ShortestPaths}(\mathfrak{G}, w, v_0, U)$, будем представлять как отображение $\sigma\colon U \to 2^E$ такое, что $\sigma(v)$ являет собой подмножество в $E$, образующее кратчайшую простую цепь, соединяющую вершины $v_0$ и $v$, для всех $v \in U$. Если такой цепи нет, то $\sigma(v) = \varnothing$.

**Алгоритм 6.** *Алгоритм* `BuildReduction` *имеет следующее определение.*

***Параметры:*** *множество $L \subset I$.*

***Результат:*** *пара* $(\mathfrak{G}, M)$, *где* $\mathfrak{G}$ *— граф, а* $M$ *— подмножество вершин* $\mathfrak{G}$.
***Псевдокод:***

$M \leftarrow \varnothing$; $i \leftarrow m$;
*Для всех* $j \in J$ *цикл*
  *Если* $|I_j \cap L| = 2$, *то*
    $\varepsilon(j) \leftarrow I_j \cap L$;
  *Иначе если* $|I_j \cap L| = 1$, *то*
    $i \leftarrow i + 1$; $M \leftarrow M \cup \{i\}$;
    $\varepsilon(j) \leftarrow (I_j \cap L) \cup \{i\}$;
  *Иначе если* $|I_j \cap L| = 0$, *то*
    $i' \leftarrow i + 1$; $i \leftarrow i + 2$; $M \leftarrow M \cup \{i, i'\}$;
    $\varepsilon(j) \leftarrow \{i, i'\}$;
  *Конец если*;
*Конец цикла*;
$V \leftarrow L \cup M$; $\mathfrak{G} \leftarrow (V, J, \varepsilon)$;
*Возврат* $(\mathfrak{G}, M)$.

Из описания алгоритма `BuildReduction` следует

**Предложение 20.** *Если* $(\mathfrak{G}, M) =$ `BuildReduction`$(L)$, *где* $\mathfrak{G} = (V, E, \varepsilon)$, *и* $\mathfrak{R}_L = (V', E', \varepsilon')$, *то отображения* $\phi\colon V \to V'$ *и* $\psi\colon E \to E'$ *такие, что* $\phi(i) = f_i$ *и* $\psi(j) = x_j$, *осуществляют изоморфизм графов* $\mathfrak{G}$ *и* $\mathfrak{R}_L$. *При этом, образ* $\phi(M)$ *равен множеству всех мнимых вершин графа* $\mathfrak{R}_L$.

Для произвольного вектора $\boldsymbol{\alpha} \in \mathbb{R}^n$ определим функцию $w_{\boldsymbol{\alpha}} : J \to \mathbb{R}$ такую, что $w_{\boldsymbol{\alpha}}(j) = \alpha_j$ для всех $j \in J$. Рассмотрим

**Алгоритм 7.** *Алгоритм* `Irreducible`$_2$ *имеет следующее определение.*

***Параметры:*** *вектор* $\boldsymbol{\gamma} \in \mathbb{R}^n$, *множество* $K \subsetneq I$ *и индекс* $i_0 \in I \backslash K$.
***Результат:*** *вектор* $\mathbf{u} \in \mathbb{F}_2^n$.
***Псевдокод:***

$(\mathfrak{G}, M) \leftarrow$ `BuildReduction`$(K \cup \{i_0\})$;
$\sigma \leftarrow$ `ShortestPaths`$(\mathfrak{G}, w_{\boldsymbol{\gamma}}, i_0, M)$;
$i_1 \leftarrow \arg \min\limits_{\substack{i \in M: \\ \sigma(i) \neq \varnothing}} \left\{ \sum_{j \in \sigma(i)} \gamma_j \right\}$;
*Составить* $\mathbf{u} \in \mathbb{F}_2^n$ *такой, что* $\operatorname{supp} \mathbf{u} = \sigma(i_1)$;
*Возврат* $\mathbf{u}$.

**Следствие 21.** *Алгоритм* $\texttt{Irreducible}_2$ *выполняется за время* $O(n^3)$ *при* $n \to \infty$, *а результат его выполнения* $\mathbf{u} = \texttt{Irreducible}_2(\boldsymbol{\gamma}, K, i_0)$ *является оптимальным решением задачи (3).*

**Доказательство.** Ясно, что вычисление пары $(\mathfrak{G}, M)$ и вектора $\mathbf{u}$ выполняется за время $O(n)$, $n \to \infty$. Согласно теореме 19 вычисление $\sigma$ занимает время $O(n^3)$, $n \to \infty$, а вычисление $i_1$, очевидно, выполняется за время $O(n^2)$, $n \to \infty$. Вторая часть утверждения вытекает из следствия 17, леммы 18, теоремы 19 и предложения 20.

Алгоритм $\texttt{Decoder}_{\texttt{A}}$ с подпроцедурой $\texttt{A} = \texttt{Irreducible}_2$ обозначим просто $\texttt{Decoder}_2$.

**Следствие 22.** *Алгоритм* $\texttt{Decoder}_2$ *выполняется за время* $O(n^4)$ *при* $n \to \infty$.

**Доказательство.** Инициализация алгоритма $\texttt{Decoder}_2$ занимает время порядка $O(n)$, $n \to \infty$. Стоимость операций, выполняемых внутри основного цикла алгоритма $\texttt{Decoder}_2$, кроме вызова процедуры $\texttt{Irreducible}_2$, зависит линейно от $n$. Поэтому общее время, затрачиваемое на выполнение одной итерации основного цикла, составляет $O(n^3)$, $n \to \infty$. А поскольку число таких итераций равно $m$, временная сложность алгоритма $\texttt{Decoder}_2$ лежит в классе $O(n^4)$ при $n \to \infty$. Следствие доказано.

Объединяя следствие 9 с двумя последними утверждениями, получаем теорему 2.

**Задача о $T$-соединении минимального веса.** В заключение мы рассмотрим случай, когда все символьные вершины в графе $\mathfrak{T}$ имеют степень 2. В этой ситуации задача (1) может быть решена за время $O(n^3)$ при $n \to \infty$.

Пусть имеется неориентированный взвешенный граф $\mathfrak{G} = (V, E, \varepsilon)$ с весовой функцией $w\colon E \to \mathbb{R}$, и пусть дано некоторое подмножество $T \subset V$ такое, что $|T|$ четно. Определим функцию $\nu\colon V \to 2^E$ такую, что $\nu(v)$ представляет собой множество всех ребер, инцидентных вершине $v$, для всех $v \in V$. Напомним, что *$T$-соединением в* $\mathfrak{G}$ называется всякое подмножество $X \subset E$ такое, что $|X \cap \nu(v)|$ нечетно тогда

и только тогда, когда $v \in T$. *Задача о $T$-соединении минимального веса* состоит в том, чтобы найти таковое, либо заключить, что оно отсутствует.

Как известно, эта задача является обобщением таких оптимизационных задач, как неориентированная задача о китайском почтальоне, неориентированная задача о кратчайшей цепи и задача о совершенном паросочетании минимального веса. Согласно следствию 12.11 в [14] существует алгоритм решения задачи о $T$-соединении минимального веса, который выполняется за время $O(|V|^3)$ при $|V| \to \infty$.

**Лемма 23.** *Если все символьные вершины графа $\mathfrak{T}$ имеют степень 2, то код $\mathcal{C}$ состоит в точности из всех векторов $\mathbf{c} \in \mathbb{F}_2^n$ таких, что множество ребер подграфа $\mathfrak{R}^{\mathbf{c}}$, порожденного в $\mathfrak{R}$ вектором $\mathbf{c}$, является $\varnothing$-соединением.*

**Доказательство.** Если ребра подграфа $\mathfrak{R}^{\mathbf{c}}$ образуют $\varnothing$-соединение, то $|J_i \cap \operatorname{supp} \mathbf{c}|$ четно для всех $i \in I$. Обратное также верно, поскольку редукция $\mathfrak{R}$ не содержит мнимых вершин. Лемма доказана.

Таким образом, задача (1) сводится к поиску $\varnothing$-соединения минимального веса, и имеет место

**Следствие 24.** *Если граф Таннера $\mathfrak{T}$ таков, что все его символьные вершины имеют степень 2, то существует алгоритм, находящий оптимальное решение задачи (1), который выполняется за время $O(n^3)$ при $n \to \infty$.*

Если граф $\mathfrak{T}$ содержит символьные вершины степени меньше 2, то задачу (1) можно свести к задаче о $T$-соединении минимального веса, используя аналогичное рассуждение. Для этого снова рассмотрим редукцию $\mathfrak{R}$ графа $\mathfrak{T}$ и решим задачу о $T$-соединении для каждого подмножества $T$ в множестве мнимых вершин графа $\mathfrak{R}$. После чего из всех полученных вариантов выберем решение наименьшего веса. Если количество концевых символьных вершин в $\mathfrak{T}$ зависит от $n$ как $f(n)$, то сложность такого решения составит $O(n^3 2^{f(n)})$ при $n \to \infty$, что неприемлемо, если $f(n)$ зависит от $n$ линейно. Алгоритм $\texttt{Decoder}_2$ можно рассматривать как один из способов преодолеть это затруднение ценой увеличения сложности алгоритма на один порядок.

## Список литературы

[1] Berlekamp E. R., McEliece R. J., Van Tilborg H. C. A. On the inherent intractability of certain coding problems // IEEE Trans. Inform. Theory. Vol. 24. 1978. P. 384–386.

[2] Bruck J., Naor M. The hardness of decoding linear codes with preprocessing // IEEE Trans. Inform. Theory. Vol. 36. 1990. P. 381–385.

[3] Coffey J. T., Goodman R. M. F., Farrell P. G. New approaches to reduced-complexity decoding // Discr. Appl. Math. Vol. 33. 1991. P. 43–60.

[4] Hwang T. Decoding linear block codes for minimizing word error rate // IEEE Trans. Inform. Theory. Vol. 25. 1979. P. 733–737.

[5] Agrell E. Voronoi regions for binary linear block codes // IEEE Trans. Inform. Theory. Vol. 42. 1996. P. 310–316.

[6] Massey J. L. Minimal codewords and secret sharing // Proc. of the 6th Joint Swedish-Russian International Workshop on Information Theory. Molle, Sweden, 1993. P. 246–249.

[7] Ashikhmin A., Barg A. Minimal vectors in linear codes and sharing of secrets // Proc. of the 4th International Workshop on Algebraic and Combinatorial Coding Theory. Novgorod, 1994. P. 1–15.

[8] Barg A. Minimum distance decoding algorithms for linear codes // Proc. of the 12th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Lecture Notes in Computer Science. Vol. 1255. 1997. P. 1–14.

[9] Ashikhmin A., Barg A. Minimal vectors in linear codes // IEEE Trans. Inform. Theory. Vol. 44. 1998. P. 2010–2017.

[10] Wiberg N. Codes and decoding on general graphs / Ph. D. dissertation. Sweden: Linkoping Univ., 1996.

[11] Lin S., Costello D. J. Error Control Coding: Fundamentals and Applications, 2nd Edition. Upper Saddle River, NJ: Pearson Education, 2004.

[12] Moon T. K. Error Correction Coding: Mathematical Methods and Algorithms. Hoboken, NJ: Wiley, 2005.

[13] Richardson T., Urbanke R. Modern Coding Theory. NY: Cambridge University, 2008.

[14] Korte B., Vygen J. Combinatorial Optimization: Theory and Algorithms, 4th Edition. Berlin: Springer-Verlag, 2008.

[15] Gabow H. N. Data structures for weighted matching and nearest common ancestors with linking // Proc. of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms. 1990. P. 434—443.