ПРОБЛЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Том 60 **2024** **Вып. 4**

УДК 616-073.756.8 : 519.6

 Д.В. Полевой, Д.Д. Казимиров, М.В. Чукалина, Д.П. Николаев

ТРАНСПОНИРОВАНИЕ СУММИРУЮЩИХ АЛГОРИТМОВ С СОХРАНЕНИЕМ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ПРИ ПОМОЩИ ГРАФОВОГО ПРЕДСТАВЛЕНИЯ ВЫЧИСЛЕНИЙ

Представлен новый метод транспонирования суммирующих алгоритмов с использованием их графового представления, обеспечивающий большую гибкость по сравнению с предыдущими подходами, основанными на явном матричном представлении соответствующего суммирующего оператора. Применение нашего метода продемонстрировано на примере транспонирования нескольких алгоритмов быстрого преобразования Хафа. Важно отметить, что наш подход сохраняет асимптотическую вычислительную сложность исходного алгоритма. Последнее свойство очень важно для приложений в компьютерной томографии.

Ключевые слова: суммирующие алгоритмы, транспонированный оператор, быстрое преобразование Хафа, паттерны, компьютерная томография, оператор прямого проецирования, оператор обратного проецирования.

DOI: 10.31857/S0555292324040053, **EDN:** RGCBHA

§ 1. Введение

В широком спектре прикладных задач хорошо известны быстрые алгоритмы вычисления суммирующих операторов. Однако быстрое вычисление транспонированных (или сопряженных) операторов часто оказывается не менее, а может быть, и более важным. Примерами операторов, для которых требуется транспонирование, являются преобразование Хафа и оператор прямого проецирования, занимающий центральное место в компьютерной томографии (КТ). В следующем параграфе мы подробнее рассмотрим эти примеры.

Ранее в литературе был предложен метод транспонирования быстрых суммирующих алгоритмов [1]. Этот метод сохраняет асимптотическую вычислительную сложность исходного алгоритма, позволяя вычислять транспонированный оператор столь же эффективно, что и прямой. Предложенный универсальный метод транспонирования основан на разложении матрицы прямого оператора в произведение более простых матриц. Однако получение такого разложения матрицы часто является отдельной, достаточно сложной задачей, которую необходимо решить до применения метода транспонирования. Согласно ранее предложенному методу транспонирование суммирующего алгоритма предлагается выполнять только после разложения матрицы оператора. В данной статье мы представляем альтернативный метод транспонирования суммирующих алгоритмов, который не требует явной матричной записи суммирующего оператора. Вместо этого мы предлагаем интерпретацию на основе ориентированных ациклических графов (directed acyclic graph, или DAG), что позволяет облегчить процесс “ручного” транспонирования.

§ 2. Прямые и транспонированные суммирующие операторы и алгоритмы: примеры

2.1. Быстрое преобразование Хафа и быстрое транспонированное преобразование Хафа. Известным примером суммирующего оператора является преобразование Хафа (ПХ), которое также называют дискретным преобразованием Радона (ДПР). Преобразование Хафа – мощный метод в области обработки изображений и компьютерного зрения. Обычно его рассматривают как метод робастной оценки параметров одной или нескольких прямых на дискретном изображении путем подсчета количества точек, лежащих на каждой прямой из некоторого множества параметризованных прямых. Метод назван в честь Пола Хафа, который впервые представил его в 1959 году как средство идентификации прямолинейных треков в экспериментах с пузырьковой камерой [2]. По своей сути, ПХ накапливает "голоса" вдоль дискретных параметризованных прямых. Накопленное значение каждой прямой указывает на вероятность того, что она пдействительно присутствует на изображении, при этом бóльшие значения указывают на бóльшую вероятность присутствия прямой. Хотя ПХ чаще всего используется для обнаружения прямых линий или отрезков на изображениях [3–6], его применение выходит далеко за эти рамки [7]. Со временем ПХ успешно применялось в различных областях, включая бинаризацию изображений [8], сегментирование [9, 10], компьютерную томографию [1, 11] и др.

Быстрые алгоритмы для вычисления преобразования Хафа, обычно называемые алгоритмами быстрого преобразования Хафа (БПХ, или FHT), получили широкое развитие. Среди них можно отметить алгоритм Брейди – Ёна [12], ставший де-факто стандартным, и алгоритм $FHT2DT$ [13, 14]. Эти алгоритмы позволяют быстро вычислить преобразование Хафа для двумерного полутонового изображения размера $w \times h$, снижая вычислительную сложность с $\Theta(w^2 h)$, которая требуется при наивном подходе – при суммировании значений пикселей вдоль каждой дискретной прямой на изображении, до $\Theta(wh \log_2 w)$. Последнее обстоятельство позволило использовать БПХ в системах с жесткими аппаратными ограничениями и требованием работы в реальном времени [15–19].

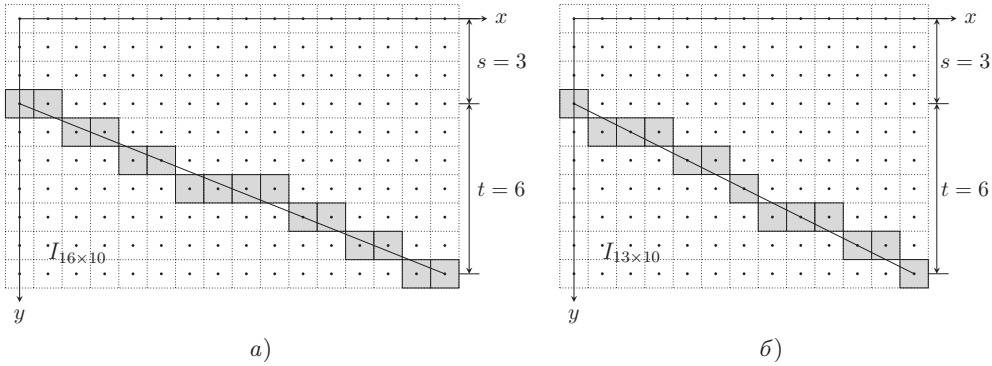
Упомянутые алгоритмы БПХ [12–14] аппроксимируют непрерывные прямые на изображении с помощью паттернов – в частности, дискретно-непрерывных прямых, совпадающих с исходными идеальными прямыми в граничных точках (см. рис. 1). В данном контексте под дискретной непрерывностью понимается отсутствие скачков в паттерне с величиной более 2 пикселей. Следует обратить внимание на то, что разные алгоритмы БПХ используют различные наборы паттернов для аппроксимации прямых линий в области изображения: в алгоритме Брейди – Ёна используются так называемые диадические паттерны длины, равной степени двойки, а в $FHT2DT$ – паттерны произвольной длины, совпадающие с диадическими только при ширине изображения, равной степени двойки.

Без ограничения общности мы будем рассматривать неубывающие прямые с наклоном в диапазоне $[0, 1]$ в системе координат изображения, называемые преимущественно горизонтальными прямыми. Преимущественно вертикальные прямые определяются аналогичным образом и получаются из преимущественно горизонтальных прямых путем отражения относительно вертикальной оси. Для описания БПХ-паттернов обычно используется st-параметризация. Пусть

$$p = p(t, s) = \{(x,\, p(t, s)(x)) \mid x \in \mathbb{Z}_{0,w-1}\}$$

представляет собой паттерн с параметрами

$$(t, s) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1},$$



а) *б*)

Рис. 1. Примеры паттернов, аппроксимирующих прямые линии в области изображения: *а*) Диадический паттерн $p_{BY}(6,3)$, используемый в алгоритме Брейди – Ёна. Здесь он служит для аппроксимации прямой $l(6,3)$ в пределах $\boldsymbol{I}_{16\times10}$. По своей конструкции диадические паттерны имеют длину, равную степени двойки; *б*) $FHT2DT$-паттерн $p_{DT}(6,3)$, аппроксимирующий прямую $l(6,3)$ в пределах $\boldsymbol{I}_{13\times10}$. Этот тип паттернов введен в алгоритме $FHT2DT$. Паттерны $FHT2DT$ могут быть произвольной длины

который, согласно конструкции конкретного алгоритма БПХ, аппроксимирует преимущественно горизонтальную прямую

$$l = l(t,s) = \left\{ \left(x, l(t,s)(x) = \left(s + \frac{t}{w-1} x \right) \bmod h \right) \;\middle|\; x \in \mathbb{Z}_{0,w-1} \right\}$$

в области изображения $\boldsymbol{I} = \boldsymbol{I}_{w\times h}$, где $w \times h$ обозначает размер изображения. По определению st-параметризации $s = p(0)$, а $p(w-1) = (t+s) \bmod h$ (рис. 1).

При заданном наборе паттернов

$$\mathcal{P} = \{p(t,s) \mid (t,s) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\},$$

зависящем от конструкции конкретного БПХ-алгоритма, БПХ $\mathcal{H}$ линейно отображает изображение $\boldsymbol{I} = \boldsymbol{I}_{w\times h} \in \mathcal{I}_{w\times h}$ в изображение $\mathcal{H}\boldsymbol{I} \in \mathcal{I}_{w\times h}$, что через st-параметризацию выражается следующим образом:

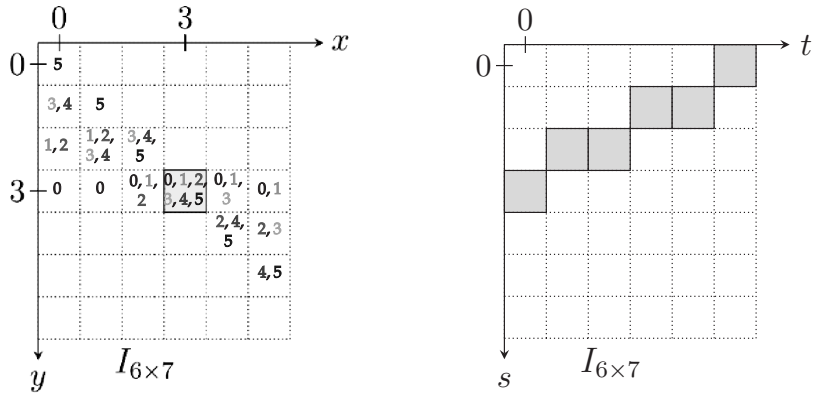
$$\mathcal{H}\colon \mathcal{I}_{w\times h} \to \mathcal{I}_{w\times h}, \quad \mathcal{H}\boldsymbol{I}(t,s) = \sum_{(x,y)\in p(t,s)} \boldsymbol{I}(x,y).$$

Здесь $\boldsymbol{I}(x,y)$ – значение пикселя с координатами $(x,y) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}$ изображения $\boldsymbol{I} \in \mathcal{I}_{w\times h}$. Через $\mathcal{I}_{w\times h}$ обозначено евклидово пространство всех двумерных одноканальных изображений размера $w \times h$ с естественным скалярным произведением $(\cdot\,,\cdot)_{\mathcal{I}_{w\times h}}$: для двух изображений $\boldsymbol{I}_1, \boldsymbol{I}_2 \in \mathcal{I}_{w\times h}$ имеем

$$(\boldsymbol{I}_1, \boldsymbol{I}_2)_{\mathcal{I}_{w\times h}} = \sum_{i\in\mathbb{Z}_{0,w-1}} \sum_{j\in\mathbb{Z}_{0,h-1}} \boldsymbol{I}_1(i,j) \cdot \boldsymbol{I}_2(i,j).$$

Полужирный шрифт $\boldsymbol{I}$ используется здесь для того, чтобы подчеркнуть, что изображение I рассматривается как элемент $\boldsymbol{I}$ евклидова векторного пространства $\mathcal{I}_{w\times h}$. Начиная с этого момента, мы будем обозначать одним символом как линейный оператор, так и его матрицу в стандартном базисе

$$\{\boldsymbol{e}_{ij} \mid (i,j) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\}, \quad \boldsymbol{e}_{ij}(k,l) = \delta_i^k \cdot \delta_j^l,$$



а) *б)*

Рис. 2. Построение транспонированного $FHT2DT$ паттерна $p^*_{DT}(3,3)$: *а)* Паттерн $FHT2DT$, содержащий пиксель с координатами $(x,y)=(3,3)$. Пиксель со значением t в нем указывает на то, что через этот пиксель проходит паттерн с наклоном t. Значение параметра s паттерна определяется значением ординаты его первого ($x=0$) пикселя; *б)* Транспонированный $FHT2DT$-паттерн $p^*_{DT}(3,3)$ как множество значений параметров (t,s) паттернов $FHT2DT$, проходящих через пиксель с координатами $(x,y)=(3,3)$

.

где δ_i^j – символ Кронекера. Например, $\mathcal{H}$ будет обозначать как БПХ, так и его матрицу в базисе $\{\boldsymbol{e}_{ij} \mid (i,j) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\}$. Так как матрица Грама базиса $\{\boldsymbol{e}_{ij} \mid (i,j) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\}$ является единичной, то из этого следует, что сопряженный, или транспонированный, оператор

$$\mathcal{H}^T \colon \mathcal{I}^{w\times h} \to \mathcal{I}^{w\times h}$$

имеет матрицу $\mathcal{H}^T$ в том же базисе $\{\boldsymbol{e}_{ij} \mid (i,j) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\}$.

В то время как прямое БПХ $\mathcal{H}$ выполняет суммирование значений пикселей изображения $\boldsymbol{I}$ вдоль некоторого набора паттернов

$$\mathcal{P} = \{p(t,s) \mid (t,s) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\},$$

транспонированное БПХ $\mathcal{H}^T$ выполняет суммирование значений пикселей изображения $\boldsymbol{I}$ вдоль некоторого набора транспонированных (сопряженных) паттернов

$$\mathcal{P}^T = \{p^T(x,y) \mid (x,y) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}\}.$$

Транспонированный паттерн $p^T(x,y)$ с параметрами $(x,y) \in \mathbb{Z}_{0,w-1} \times \mathbb{Z}_{0,h-1}$ определяется как

$$p^T(x,y) = \{(t,s) \mid (x,y) \in p(t,s),\ p(t,s) \in \mathcal{P}\}.$$

Иными словами, транспонированный паттерн $p^T(x,y)$ состоит из пар значений параметров (t,s) всех паттернов, проходящих через точку с координатами (x,y) (рис. 2). Действие транспонированного БПХ $\mathcal{H}^T$ можно записать следующим образом:

$$\mathcal{H}^T \colon \mathcal{I}_{w\times h} \to \mathcal{I}_{w\times h}, \quad \mathcal{H}^T\boldsymbol{I}(x,y) = \sum_{(t,s)\in p^T(x,y)} \boldsymbol{I}(t,s).$$

Быстрое и точное вычисление транспонированного ПХ крайне важно в ряде прикладных областей, в частности, оно является первостепенной необходимостью в задаче рентгеновской компьютерной томографии.

Хорошо известно, что транспонированный диадический паттерн также является диадическим [20,21]. Это означает, что оригинальный алгоритм Брейди – Ёна можно непосредственно использовать для вычисления сумм по транспонированным диадическим паттернам. Таким образом, алгоритм Брейди – Ёна дает самосопряженный метод для вычисления БПХ. Однако недостатком алгоритма Брейди – Ёна при вычислении БПХ является то, что он применим только к изображениям с шириной, равной степени двойки. На практике чаще всего встречаются изображения произвольной ширины, не обязательно равной степени двойки. Для таких изображений алгоритм Брейди – Ёна неприменим, что делает особенно актуальной потребность в транспонированных БПХ-алгоритмах, применимых к изображениям произвольной ширины, таких как алгоритм $FHT2DT$ [13, 14].

При этом матрица оператора, реализуемого алгоритмом $FHT2DT$, имеет нетривиальную структуру, что затрудняет применение метода транспонирования [1]. Это подчеркивает необходимость разработки сохраняющих сложность методов транспонирования для суммирующих алгоритмов, не зависящих от конкретного вида матрицы оператора. В настоящей статье предлагается такой метод.

2.2. Операторы прямого и обратного проецирования в КТ. В качестве еще одной особо важной задачи, ключевую роль в которой играет суммирующий оператор, в частности оператор БПХ, можно назвать задачу КТ. Наиболее вычислительно трудоемкой частью многих методов томографической реконструкции [22–24] является вычисление операторов прямого и обратного проецирования. Прямое проецирование описывается выражением

$$\boldsymbol{W}\boldsymbol{m} = \boldsymbol{g}, \tag{1}$$

где $\boldsymbol{g}$ – вектор линеаризованных лучевых сумм, $\boldsymbol{m}$ – вектор реконструированных значений, $\boldsymbol{W}$ – проекционная матрица размера $N \cdot C \times N^2$, N – число ячеек в линейке детектора, C – число проекций. Элементы проекционной матрицы $0 \leqslant w_{i,j} \leqslant 1$ определяют вклад пикселя в лучевую сумму, а сама матрица $\boldsymbol{W}$ описывает измерительную схему. Матрица $\boldsymbol{W}^T$ задает обратное проецирование [22].

Вычисление операторов проецирования используется для КТ-реконструкции на основе метода свертки и обратной проекции (filtered back projection, FBP) [25, 26], при реализации итеративных алгебраических подходов [21, 27, 28] и во многих подходах с использованием нейронных сетей [29–31]. Один из подходов быстрого приближенного вычисления операторов опирается на иерархическое разложение дискретного приближения линейных интегралов [12,32–37]. В дискретном пространстве семейства пересекающихся прямых могут иметь общие подпоследовательности пикселей (паттерны), поэтому требуемое для вычисления операторов число операций уменьшается за счет повторного использования частичных сумм, соответствующих пересечениям нескольких паттернов. Возникающие при использовании дискретного приближения ошибки аппроксимации ограничены и хорошо изучены [34, 37–39].

В рамках вышеупомянутого подхода иерархического разложения линейных интегралов паттерны как общие подмножества близко расположенных прямых в дискретном пространстве непосредственно относятся к определенным ранее паттернам БПХ. Таким образом, БПХ представляет собой существенное ядро целого ряда быстрых схем вычислений для операторов прямого проецирования в КТ. В то же время оператор обратного проецирования можно вычислить, используя транспонированный оператор прямого проецирования [1], что означает, что транспонированное БПХ служит эффективным средством для вычисления оператора обратного проецирования. Таким образом, схемы быстрого вычисления операторов прямого и обратного проецирования могут быть основаны на алгоритмах, эффективно вычисляющих пару операторов – прямое и транспонированное БПХ.

Схему быстрых приближенных вычислений для (1) можно представить в виде линейного оператора

$$\boldsymbol{B}\colon \mathbb{R}^n \to \mathbb{R}^m. \tag{2}$$

Матрица $\boldsymbol{B}$ этого оператора является булевой матрицей вида

$$\boldsymbol{B} \stackrel{\text{def}}{=} (\boldsymbol{B}(y,x) = b_{y,x} \in \{0,1\} :\ y \in \mathbb{Z}_{0,H-1}, x \in \mathbb{Z}_{0,W-1}),$$

где y обозначает номер строки в матрице $\boldsymbol{B}$, x – номер столбца в матрице $\boldsymbol{B}$, через $H \equiv H(\boldsymbol{B})$ обозначается количество строк матрицы, а через $W \equiv W(\boldsymbol{B})$ – количество столбцов матрицы, $H(\boldsymbol{B}), W(\boldsymbol{B}) \in \mathbb{Z}_{1,\infty}$.

Для оператора $\boldsymbol{B}$ вида (2) оператором обратного проецирования $\boldsymbol{B}^T$, задаваемым матрицей $\boldsymbol{B}^T$, будем называть транспонированный оператор

$$\boldsymbol{B}^T\colon \mathbb{R}^m \to \mathbb{R}^n. \tag{3}$$

Пусть $\mathrm{dec}^p(\cdot)$ – предварительно установленное разложение булевой матрицы в произведение p булевых матриц:

$$\mathrm{dec}^p(\boldsymbol{B}) \stackrel{\text{def}}{=} \prod_{i=p}^{1} \mathrm{dec}_i^p(\boldsymbol{B}) = \prod_{i=p}^{1} \boldsymbol{B}_i = \boldsymbol{B}_p \boldsymbol{B}_{p-1} \ldots \boldsymbol{B}_1 = \boldsymbol{B}, \tag{4}$$

где $\mathrm{dec}_i^p(\boldsymbol{B}) \stackrel{\text{def}}{=} \boldsymbol{B}_i$ – i-я компонента разложения. Мы предполагаем, что $\mathrm{dec}^1(\boldsymbol{B}) \equiv \boldsymbol{B}$, и без ограничения общности далее будем считать, что матрица $\boldsymbol{B}$ оператора $\boldsymbol{B}$ и все матрицы $\mathrm{dec}_i^p(\boldsymbol{B})$ в разложении не содержат нулевых строк и столбцов. Для каждого разложения $\mathrm{dec}^p(\boldsymbol{B})$ матрицы $\boldsymbol{B}$, представляющей прямой оператор $\boldsymbol{B}$, существует соответствующее разложение $\mathrm{dec}^p(\boldsymbol{B}^T)$ матрицы $\boldsymbol{B}^T$ для транспонированного оператора $\boldsymbol{B}^T$. Оно вычисляется через транспонирование произведения матриц:

$$\mathrm{dec}^p(\boldsymbol{B}^T) = \boldsymbol{B}^T = (\boldsymbol{B}_p \boldsymbol{B}_{p-1} \ldots \boldsymbol{B}_1)^T = \boldsymbol{B}_1^T \ldots \boldsymbol{B}_{p-1}^T \boldsymbol{B}_p^T = \prod_{i=p}^{1} \boldsymbol{B}_{p-i+1}^T. \tag{5}$$

Алгоритм вычисления оператора проецирования соответствует умножению на булевы матрицы. Как было отмечено выше, общий метод построения быстрого алгоритма вычисления линейного оператора обратного проецирования (3) при известном быстром алгоритме вычисления линейного оператора прямого проецирования (2) был предложен в работе [1] и опирается на соотношения (4) и (5). Согласно этому методу алгоритмы вычисления прямого и транспонированного операторов, а именно операторов прямого и обратного проецирования, имеют асимптотическую сложность (по числу сумм) одного порядка. Однако представление вычисления операторов в виде цепочек умножения булевых матриц, как требуется в методе из [1], усложняет его применение к известному быстрому алгоритму, выраженному в виде псевдокода или программы на каком-либо языке программирования. Матрица оператора суммирования, неявно реализуемая суммирующими алгоритмами, часто определяется рекурсивно (подобно рекурсивному построению паттернов в алгоритмах Брейди – Ёна и $FHT2DT$) и имеет размер, зависящий от размера входных данных, который в приложениях может быть достаточно большим для “ручного” анализа.

Далее в данной статье этот недостаток устраняется за счет изложения метода транспонирования суммирующих алгоритмов в терминах эквивалентной алгоритму вычислительной сети.

§ 3. Общие определения и методология

Назовем (суммирующей) сетью ориентированный ациклический граф (directed acyclic graph, DAG)

$$D \stackrel{\text{def}}{=} (V, A),$$

где V – множество вершин (значений), A – множество дуг (ориентированных ребер), описывающих порядок суммирования.

Далее используются следующие обозначения:

$\text{prev}(v) \stackrel{\text{def}}{=} \{u \in V : \exists(u, v) \in A\}$ – множество предков (входов) вершины $v \in V$,

$\text{next}(v) \stackrel{\text{def}}{=} \{u \in V : \exists(v, u) \in A\}$ – множество потомков (выходов) вершины $v \in V$,

$\text{inp}(D) \stackrel{\text{def}}{=} \{v \in V : \text{prev}(v) = \varnothing\}$ – вход сети, множество входных вершин сети D,

$\text{out}(D) \stackrel{\text{def}}{=} \{v \in V : \text{next}(v) = \varnothing\}$ – выход сети, множество выходных вершин сети D.

При заданных значениях входа $\text{inp}(D)$ значения в остальных вершинах такой сети вычисляются по формуле

$$v = \sum_{u \in \text{prev}(v)} u, \quad \forall v \in V \setminus \text{inp}(D).$$

3.1. Суммирующая сеть, эквивалентная матричному разложению в произведение булевых матриц. Покажем, как построить вычислительную суммирующую сеть, соответствующую алгоритму вычисления оператора проецирования, задаваемого разложением произведения булевых матриц вида (4). Для этого введем следующие обозначения:

$$\boldsymbol{V}^{(0)} = \boldsymbol{m},$$
$$\boldsymbol{V}^{(p)} = \boldsymbol{B}(\boldsymbol{m}),$$
$$\boldsymbol{V}^{(i)} = \boldsymbol{B}_i \boldsymbol{V}^{(i-1)}.$$

Вычисление оператора записывается в виде

$$\boldsymbol{V}^{(p)} = \boldsymbol{B}_p \boldsymbol{B}_{p-1} \dots \boldsymbol{B}_1 \boldsymbol{V}^{(0)} = \prod_{i=p}^{1} \text{dec}_i^p(\boldsymbol{B}) \boldsymbol{V}^{(0)}. \tag{6}$$

Далее будем использовать обозначение $V^{(i)}$ для множества вершин сети, составленного из элементов вектора $\boldsymbol{V}^{(i)}$. Алгоритму вычисления выражения (6) соответствует суммирующая сеть вида

$$D_{\boldsymbol{B}} = (V_{\boldsymbol{B}}, A_{\boldsymbol{B}}),$$
$$V_{\boldsymbol{B}} = \bigcup_{i=0}^{p} V^{(i)} : \quad \forall i \neq j, \quad V^{(i)}, V^{(j)} \in V \ \rightarrow \ V^{(i)} \cap V^{(j)} = \varnothing,$$
$$A_{\boldsymbol{B}} = \bigcup_{i,x,y} \left(v_x^{(i-1)}, v_y^{(i)}\right) : \quad \boldsymbol{B}_i(y, x) = 1,$$
$$\text{inp}(D_{\boldsymbol{B}}) = V^{(0)},$$
$$\text{out}(D_{\boldsymbol{B}}) = V^{(p)}.$$

Существование в сети дуги $a = (v_x^{(i-1)}, v_y^{(i)})$ означает, что значение вершины $v_x^{(i-1)} \in V^{(i-1)}$ входит в сумму при расчете значения вершины $v_y^{(i)} \in V^{(i)}$, поэтому

$$\boldsymbol{B}_i(y,x) = 1 \;\Leftrightarrow\; \exists (v_x^{(i-1)}, v_y^{(i)}) \in A_{\boldsymbol{B}}.$$

3.2. Эквипотенциальная суммирующая сеть. Покажем, что суммирущая сеть вычислительно эквивалентна цепочке умножений на булевы матрицы. Для этого определим расстояние $\mathrm{dist}\colon V_{\boldsymbol{B}} \to \mathbb{Z}_{0,\infty}$ в сети $D_{\boldsymbol{B}}$ от произвольной вершины $v \in V_{\boldsymbol{B}}$ до входа сети $\mathrm{inp}(D_{\boldsymbol{B}})$ по формуле

$$\mathrm{dist}(v) \stackrel{\mathrm{def}}{=} \begin{cases} 0, & \forall v \in \mathrm{inp}(D_{\boldsymbol{B}}), \\ \max\limits_{u \in \mathrm{prev}(v)} (\mathrm{dist}(u)) + 1, & \forall v \in V_{\boldsymbol{B}} \setminus \mathrm{inp}(D_{\boldsymbol{B}}). \end{cases}$$

Назовем i-слоем сети $D_{\boldsymbol{B}}$ и будем обозначать через $V^{(i)}$ эквидистантное подмножество вершин с заданным расстоянием $0 \leqslant i \leqslant \mathrm{depth}(D_{\boldsymbol{B}})$ до входа

$$V^{(i)} \stackrel{\mathrm{def}}{=} \{v \in V_{\boldsymbol{B}} :\ \mathrm{dist}(v) = i\},$$

где $\mathrm{depth}(D) = \max\limits_{v \in V_{\boldsymbol{B}}} \mathrm{dist}(v)$ – глубина сети (максимальная длина пути в сети). По определению

$$V_{\boldsymbol{B}} = \bigcup_{j=0}^{\mathrm{depth}(D_{\boldsymbol{B}})} V^{(j)}, \quad \forall i \neq k \;\to\; V^{(i)} \cap V^{(k)} = \varnothing.$$

Определим длину дуги $\mathrm{len}\colon A_{\boldsymbol{B}} \to \mathbb{N}$, $\forall a = (u,v) \in A_{\boldsymbol{B}}$, как

$$\mathrm{len}(a) = \mathrm{len}(u,v) \stackrel{\mathrm{def}}{=} \mathrm{dist}(v) - \mathrm{dist}(u).$$

Назовем сеть эквипотенциальной, если выполняется условие

$$\forall a \in A_{\boldsymbol{B}} \;\to\; \mathrm{len}(a) = 1. \tag{7}$$

В эквипотенциальной сети каждый последующий слой зависит только от предыдущего. При некотором заданном на $V_{\boldsymbol{B}}$ линейном порядке каждому слою $V^{(i)}$ соответствует единственный вектор $\boldsymbol{V}^{(i)}$, а зависимость между слоями можно описать в векторно-матричном виде как

$$\forall\, 1 \leqslant j \leqslant \mathrm{depth}(D_{\boldsymbol{B}}) \to \boldsymbol{V}^{(j)} = \boldsymbol{B}_j \boldsymbol{V}^{(j-1)} = \boldsymbol{B}_j \boldsymbol{B}_{j-1} \ldots \boldsymbol{B}_1 \boldsymbol{V}^{(0)},$$

где $\boldsymbol{B}_j$ – булева матрица. Таким образом, при выполнении условия (7) вычисление в сети описывается произведением булевых матриц вида (4).

3.3. Нормализация суммирующей сети. Покажем, что если условие эквипотенциальности (7) нарушено, то сеть D может быть нормализована, т.е. приведена к вычислительно эквивалентному эквипотенциальному виду. Пример основных шагов процесса нормализации показан на рис. 3, *а*)–*в*).

Вырожденной назовем вершину $v \in V$ суммирущей сети D, которая не участвует в сложениях (рис. 3*а*) и 3*б*)), не влияет на результат вычисления сети и не изменяет количество суммирований в сети. Для вырожденной вершины выполняется соотношение

$$\dim(\mathrm{prev}(v)) = \dim(\mathrm{next}(v)) = 1.$$

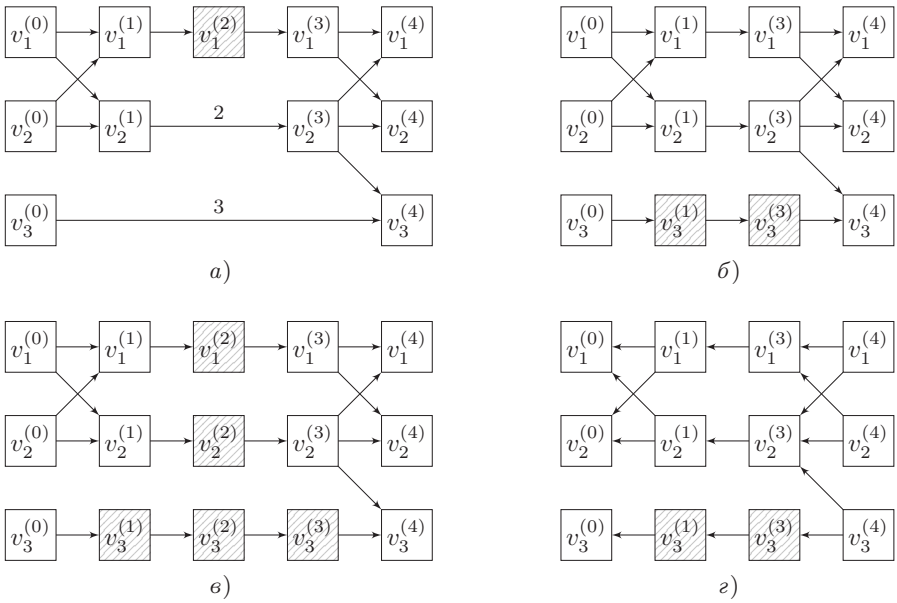


Рис. 3. Пример суммирующей сети; длины дуг подписаны только для значений, отличных от 1: *а*) исходная сеть с нарушением эквипотенциальности и вырожденной вершиной $v_1^{(2)}$; *б*) эквипотенциальная сеть с вырожденным слоем $V^{(2)}$; *в*) эквипотенциальная сеть после удаления вырожденных вершин $v_3^{(1)}$ и $v_3^{(3)}$; *г*) обратная эквипотенциальная суммирующая сеть с вырожденными вершинами $v_3^{(1)}$ и $v_3^{(3)}$

Слой сети, полностью состоящий из вырожденных вершин, как показано на рис. 3*б*), назовем вырожденным слоем. Удаление вырожденного слоя из эквипотенциальной сети сохраняет ее эквипотенциальность и не изменяет результатов вычисления.

Сформулируем способ приведения суммирущей сети к нормализованной эквипотенциальной форме. Пусть $\exists(u,v) \in A$: $1 < \ell = \operatorname{len}(u,v)$, тогда нормализация осуществляется добавлением в сеть $\ell - 1$ вырожденных вершин $\{e_i\}_{i=1}^{\ell-1}$ и заменой дуги (u,v) на последовательность дуг $(u,e_1),(e_1,e_2),\ldots,(e_{\ell-1},v)$. Таким способом можно заменить каждую дугу, нарушающую условие (7), на последовательность дуг, удалить все вырожденные слои и получить эквипотенциальную суммирующую сеть, вычислительно эквивалентную исходной (см. рис. 3*в*)). В дальнейшем все суммирующие сети будем считать эквипотенциальными.

3.4. Метод транспонирования суммирующего алгоритма. Покажем, как в терминах обращения суммирующих сетей представляется транспонирование алгоритма вычисления оператора проецирования. Пусть алгоритму вычисления прямого оператора соответствует сеть $D_{\boldsymbol{B}}$. Рассмотрим обратную сеть $D_{\boldsymbol{B}^{\mathrm{T}}}$, которая получается из прямой сети сменой направления всех дуг на противоположное:

$$\operatorname{inv}(a) = \operatorname{inv}\bigl(v_x^{(i-1)}, v_y^{(i)}\bigr) = \bigl(v_y^{(i)}, v_x^{(i-1)}\bigr).$$

Рассмотрим связь двух соседних слоев исходной сети. Для каждого ненулевого элемента матрицы $\boldsymbol{B}_i(k,j) = 1$ вершина $\boldsymbol{V}_j^{(i-1)}$ входит в сумму для вычисления $\boldsymbol{V}_k^{(i)}$, поэтому смена направления дуг в сети соответствует транспонированию матрицы перехода между слоями. При смене направления всех дуг слои суммируются в об-

ратном порядке. Таким образом, для обратной сети справедливо равенство

$$\boldsymbol{V}^{(i-1)} = \boldsymbol{B}_i^T \boldsymbol{V}^{(i)}$$

и $\forall i \in \mathbb{Z}_{0,p-1}$ можно вычислить

$$\boldsymbol{V}^{(i)} = \boldsymbol{B}_{i+1}^T \boldsymbol{V}^{(i+1)} = \boldsymbol{B}_{i+1}^T \boldsymbol{B}_{i+2}^T \dots \boldsymbol{B}_p^T \boldsymbol{V}^{(p)}.$$

Учитывая разложение (5), полная обратная сеть соответствует алгоритму вычисления транспонированного оператора

$$\boldsymbol{V}^{(0)} = \boldsymbol{B}_1^T \boldsymbol{B}_2^T \dots \boldsymbol{B}_p^T \boldsymbol{V}^{(p)} = \boldsymbol{B}^T \boldsymbol{V}^{(p)}.$$

Таким образом, если прямая сеть описывает вычисление оператора $\boldsymbol{B}$, то обратная сеть описывает вычисление оператора $\boldsymbol{B}^T$.

Общий метод транспонирования суммирующего алгоритма вычисления оператора $\boldsymbol{B}$ можно представить в виде следующих шагов:

1. представить суммирующий алгоритм вычисления оператора $\boldsymbol{B}$ в виде нормализованной суммирующей сети $D_{\boldsymbol{B}}$;
2. построить обратную суммирующую сеть $D_{\boldsymbol{B}^T}$ путем изменения направления всех дуг сети $D_{\boldsymbol{B}}$ на противоположные;
3. представить суммирующую сеть $D_{\boldsymbol{B}^T}$ в виде алгоритма вычисления оператора $\boldsymbol{B}^{\boldsymbol{T}}$.

Далее мы рассмотрим примеры практического применения обобщенного метода транспонирования суммирующих алгоритмов применительно к задачам вычисления операторов обратного проецирования в методах томографической реконструкции. На основе метода транспонирования с использованием ориентированных ациклических графов, предложенного в этой статье, также будет получено транспонирование алгоритма $FHT2DT$.

§ 4. Результаты

4.1. Транспонирование оператора прямого проецирования для задачи реконструкции в двумерной малоракурсной КТ с параллельно-лучевой схемой. Недостатком матричного представления в работе [1] является нетривиальность сопоставления алгоритма с их матричными факторизациями. В ней представлены примеры транспонирования алгоритмов, но при этом не показывается, какие части алгоритмов с какими матричными умножениями соотносятся, и кроме того, термин “распространение” (spreading) используется без формального определения.

Далее рассмотрим, как графовую интерпретацию можно применять для транспонирования алгоритма 2 вычисления оператора прямого проецирования для задачи двумерной малоракурсной КТ с параллельно-лучевой схемой из [1]. Здесь термин “малоракурсная” означает, что лучевые интегралы вычисляются по разреженному набору направлений, т.е. на вход алгоритма реконструкции подаются проекции, соответствующие разреженному набору направлений распространения рентгеновских лучей. Этот алгоритм для входного изображения I_{2^n} размера $2^n \times 2^n$ вычисляет выходной вектор сумм $\boldsymbol{s} = (s_j)_{j=0}^{q-1}$ для q дискретных прямых с параметрами (s,t), определяемых как множество $L = \{(x_j, a_j)\}_{j=0}^{q-1}$.

Алгоритм 2 Алгоритм для прямого БПХ с терминацией и досчетом

1: **procedure** $dirPFHT(L, I_{2^n}, n, k)$
2: $R_0(x, y, 0) \leftarrow I_{2^n}(x, y) \quad \forall x \in \mathbb{Z}_{0,2^n-1}, y \in \mathbb{Z}_{0,2^n-1}$
3: $R_0(x, y, 0) \leftarrow 0 \quad \forall x \in \mathbb{Z}_{2^n,2^{n+1}-1}, y \in \mathbb{Z}_{0,2^n-1}$
4: $R_k \leftarrow dirPFHTk(R_k, n, k)$
5: $s_j \leftarrow dirSUMk(x_j, a_j, R_k, n, k) \quad \forall j \in \mathbb{Z}_{0,q-1}, (x_j, a_j) \in L$
6: **return** s

Для понимания структуры суммирования элементов следует подробнее рассмотреть строки 4 и 5. Алгоритм 3 *dirPFHTk* вычисляет и сохраняет суммы для входного тензора R_0 в виде тензора R_k, производя суммирование по подпаттернам длины 2^k.

Алгоритм 3 Прямое БПХ с терминацией

1: **procedure** $dirPFHTk(R_0, n, k)$
2: **for** $i = 1$ to k **do**
3: **for** $a = 0$ to $2^i - 1$ **do**
4: **for** $y = 0$ to $2^n - 2^i$ step 2^i **do**
5: **for** $x = 0$ to $2^{n+1} - 1$ **do**
6: $x_2 \leftarrow (x - \lceil a/2 \rceil) \bmod 2^{n+1}$
7: $y_2 \leftarrow y + 2^{i-1}$
8: $R_i(x, y, a) \leftarrow R_{i-1}(x, y, \lfloor a/2 \rfloor) + R_{i-1}(x_2, y_2, \lfloor a/2 \rfloor)$
9: **return** R_k

Алгоритм 4 *dirSUMk* досчитывает сумму по паттерну с заданными параметрами (x, a), где $x \in \mathbb{Z}_{0,2^n-1}, a \in \mathbb{Z}_{0,2^n-1}$, для тензора R_k с суммами по подпаттернам длины 2^k через рекурсивный вызов функции *recSUMk*.

Алгоритм 4 Досчет для прямого оператора проецирования

1: **procedure** $dirSUMk(x, a, R_k, n, k)$
2: **return** $recSUMk(x, 0, a, n, R_k, n, k)$

Алгоритм 5 *recSUMk* рекурсивно суммирует предварительно вычисленные частичные суммы по паттернам длины 2^i, $i \in \mathbb{Z}_{k,n}$, в R_k вдоль дискретной прямой, заданной значениями (x, a).

Алгоритм 5 Досчет

1: **procedure** $recSUMk(x, y, a, i, R_k, n, k)$
2: **if** $i = k$ **then**
3: $s \leftarrow R_k(x, y, a)$
4: **else**
5: $x_2 \leftarrow (x - \lceil a/2 \rceil) \bmod 2^{n+1}$
6: $y_2 \leftarrow y + 2^{i-1}$
7: $s \leftarrow recSUMk(x, y, \lfloor a/2 \rfloor, i - 1, R_k, n, k) + recSUMk(x_2, y_2, \lfloor a/2 \rfloor, i - 1, R_k, n, k)$
8: **return** s

В алгоритме 2 прямое суммирование идет последовательно двумя этапами, что соответствует естественному разделению вычислительной сети на два последовательных сегмента, для первого из которых один некоторый слой является выходным, а для второго сегмента – входным. Первый сегмент сети соответствует вызову *dirPFHTk* в строке 4, второй сегмент сети соответствует всем вызовам *dirSUMk* в строке 5. При транспонировании алгоритма направления дуг сети меняются в каждом из сегментов, а сами сегменты вычисляются в обратном порядке. Транспонированный алгоритм 6 называется *revPFHT* (reversed Partial Fast Hough Transform).

Алгоритм 6 Транспонированный алгоритм *dirPFHT*

1: **procedure** $revPFHT(L, s, n, k)$
2: $R_k(x, y) \leftarrow 0 \quad \forall\, x \in \mathbb{Z}_{0,2^{n+1}-1}, y \in \mathbb{Z}_{0,2^n-1}$
3: $R_k \leftarrow revSUMk(s_j, x_j, a_j, R_k, n, k)\ \forall j \in \mathbb{Z}_{1,q}$
4: $R_0 \leftarrow revPFHTk(R_k, n, k)$
5: $I_{2^n}(x, y) \leftarrow R_0(x, y)\ \forall\, x \in \mathbb{Z}_{0,2^n-1}, y \in \mathbb{Z}_{0,2^n-1}$
6: **return** I_{2^n}

Он вычисляет действие транспонированного оператора на вектор лучевых сумм $\boldsymbol{s} = (s_i)_{i=0}^{q-1}$ и набор дискретных прямых $L = \{(x_i, a_i)\}_{i=0}^{q-1}$ в заданном диапазоне направлений. Для транспонирования первого сегмента сети в строке 2 алгоритма 6 используем алгоритм 7, запускающий вызов рекурсивной функции *recSPRk* алгоритма 8.

Алгоритм 7 Транспонированный алгоритм досчета *dirSUMk*

1: **procedure** $revSUMk(s, x, a, R_k, n, k)$
2: $R_k \leftarrow recSPRk(s, x, 0, a, n, R_k, n, k)$
3: **return** R_k

Положения значений частичных сумм в R_i определяются рекурсивным спуском в алгоритме 8, а "накопление суммы вдоль паттерна" после транспонирования соответствует накоплению в элементах R_k значений из $\boldsymbol{s}$ в соответствии с тем, из каких частичных сумм вдоль прямых складывались значения s_i.

Алгоритм 8 Алгоритм рекурсивного распространения для вычисления транспонированного оператора

1: **procedure** $recSPRk(s, x, y, a, i, R_k, n, k)$
2: **if** $i = k$ **then**
3: $R_k(x, y + a) \leftarrow R_k(x, y + a) + s$
4: **else**
5: $R_k \leftarrow recSPRk(s, x, y, \lfloor a/2 \rfloor, i - 1, R_k, n, k)$
6: $R_k \leftarrow recSPRk(s, x - \lceil a/2 \rceil, y + 2^{i-1}, \lfloor a/2 \rfloor, i - 1, R_k, n, k)$
7: **return** R_k

Второй сегмент сети соответствует прямому суммированию в алгоритме 3, которое выполняется в строке 8. Чтобы получить сеть с обратным порядком сложения, организуем счетчик циклов i в строке 2 в обратном порядке, а также "обратное" суммирование, что соответствует строкам 9 и 10 алгоритма 9.

Алгоритм 9 Окончание вычисления полностью транспонированного оператора для матрицы R_k

1: **procedure** $revPFHTk(R_k, n, k)$
2: **for** $i = k$ to 1 step -1 **do**
3: $R_{i-1}(x, y, 0) \leftarrow 0 \quad \forall\, x \in \mathbb{Z}_{0,2^{n+1}-1}, y \in \mathbb{Z}_{0,2^n-1}$
4: **for** $a = 0$ to $2^i - 1$ **do**
5: **for** $y = 0$ to $2^n - 2^i$ step 2^i **do**
6: **for** $x = 0$ to $2^{n+1} - 1$ **do**
7: $x_2 \leftarrow (x - \lceil a/2 \rceil) \bmod 2^{n+1}$
8: $y_2 \leftarrow y + 2^{i-1}$
9: $R_{i-1}(x, y, \lfloor a/2 \rfloor) \leftarrow R_{i-1}(x, y, \lfloor a/2 \rfloor) + R_i(x, y, a)$
10: $R_{i-1}(x_2, y_2, \lfloor a/2 \rfloor) \leftarrow R_{i-1}(x_2, y_2, \lfloor a/2 \rfloor) + R_i(x, y, a)$
11: **return** R_k

Измерения временной сложности алгоритмов, представленных в данном пункте, для различных значений входных параметров можно найти в [1].

4.2. Транспонирование алгоритма *FHT2DT*. Недавно в [13, 14] был предложен новый быстрый алгоритм $FHT2DT$ для вычисления преобразования Хафа. Алгоритм $FHT2DT$ отличается возможностью обработки изображений произвольного размера, тогда как, например, де-факто стандартный алгоритм Брейди – Ёна позволяет быстро вычислить преобразование Хафа для изображений с шириной, строго равной степени двойки. При этом, как доказано в том же исследовании [13, 14], алгоритм $FHT2DT$ является более точным по сравнению с другими известными ранее и описанными в литературе алгоритмами быстрого преобразования Хафа, работающими для случая произвольного размера изображения.

Алгоритм $FHT2DT$ устроен следующим образом (см. алгоритм 10, [13,14]). Изображение $I = I_{w\times h}$ произвольной ширины w и высоты h разбивается на два подызображения – левое I_L и правое I_R (алгоритм 10, строки 4–8). Левое подызображение I_L имеет ширину

$$w_L = 2^{\lceil \log_2 w \rceil - 1},$$

т.е. равно максимальной степени двойки, меньшей ширины исходного изображения; второе подызображение I_R имеет ширину, равную $w_R = w - w_L$. Высота обоих подызображений равна h. Для левого и правого подызображений I_L и I_R вычисляется (рекурсивно) преобразование Хафа $FHT2DT$ и получаются Хаф-образы J_L и J_R (алгоритм 10, строки 9–10). Затем Хаф-образы J_L и J_R подызображений I_L и I_R объединяются (алгоритм 10, строки 11–18) и образуют изображение J, которое и является результатом работы алгоритма $FHT2DT$.

Алгоритм 10 Алгоритм $FHT2DT$ ($dirFHT2DT$) вычисления БПХ для изображения произвольного размера

```
 1: procedure FHT2DT(w, h, I = I_{w×h})
 2:     if w > 1 then
 3:         p ← ⌈log_2 w⌉ − 1
 4:         w_L ← 2^p
 5:         w_R ← w − w_L
 6:         I_L ← I(0 : w_L, :)                          ▷ I_L – изображение
 7:         I_R ← I(w_L : w, :)                          ▷ I_R – изображение
 8:         J_L ← FHT2DT(w_L, h, I_L)                    ▷ J_L – изображение
 9:         J_R ← FHT2DT(w_R, h, I_R)                    ▷ J_R – изображение
10:         J ← Create_Zeroed_Image(w, h)                ▷ J – изображение
11:         k_L ← (w_L − 1)/(w − 1)
12:         k_R ← (w_R − 1)/(w − 1)
13:         for t ← 0 to w − 1 do
14:             t_L ← [t k_L]
15:             t_R ← [t k_R]
16:             s ← (t − t_R) mod h
17:             J(t, :) ← J_L(t_L, :) + Concat(J_R(t_R, s : h), J_R(t_R, 0 : s))
18:     else
19:         J ← I
20:     return J = J_{w×h}
```

В алгоритме 10 используется несколько вспомогательных функций. Функция $Create\_Zeroed\_Image(w, h)$ возвращает изображение размера $w \times h$. Кроме того, функция $Concat(I_1, I_2)$ возвращает изображение, являющееся конкатенацией изображений I_1 и I_2 по горизонтальной оси. Мы также будем придерживаться slice-нотации, т.е. $n_1 : n_2$ обозначает диапазон от n_1 до n_2 (включая n_1, не включая n_2).

Отсутствие индекса n_1 или n_2 указывает на полный диапазон с соответствующей стороны. Отсутствие обоих индексов указывает на полный диапазон.

Метод транспонирования суммирующих алгоритмов, предложенный в данной статье, мы применили к алгоритму $FHT2DT$ ($dirFHT2DT$). Алгоритм 11 описывает транспонированный алгоритм $revFHT2DT$. Он позволяет быстро вычислить транспонированное (сопряженное) преобразование Хафа, поскольку наш метод транспонирования суммирующих алгоритмов сохраняет асимптотическую вычислительную сложность прямого алгоритма $FHT2DT$ ($dirFHT2DT$).

Алгоритм 11 Алгоритм $revFHT2DT$ вычисления транспонированного БПХ для изображения произвольного размера

```
1:  procedure revFHT2DT(w, h, J = J_{w×h})
2:      if w > 1 then
3:          p ← ⌈log_2 w⌉ − 1
4:          w_L ← 2^p
5:          w_R ← w − w_L
6:          k_L ← (w_L − 1)/(w − 1)
7:          k_R ← (w_R − 1)/(w − 1)
8:          J_L ← Create_Zeroed_Image(w_L, h)
                                  ▷ Для распространения по левым подпаттернам
9:          J_R ← Create_Zeroed_Image(w_R, h)
                                  ▷ Для распространения по правым подпаттернам
10:         for t ← 0 to w − 1 do
11:             t_L ← [t k_L] mod w_L
12:             t_R ← [t k_R] mod w_R
13:             s ← (t_R − t) mod h
14:             J_L(t_L, :) ← J_L(t_L, :) + J(t, :)   ▷ Распространение по подпаттернам
15:             J_R(t_R, :) ← J_R(t_R, :) + Concat(J(t, s : h), J(t, 0 : s))
16:         I_L ← revFHT2DT(w_L, h, J_L)
                          ▷ Вычисление изображения Хафа после распространения
17:         I_R ← revFHT2DT(w_R, h, J_R)
                          ▷ Вычисление изображения Хафа после распространения
18:         I ← Concat(I_L, I_R)                  ▷ Конкатенация изображений Хафа
19:     else
20:         I ← J
21:     return I = I_{w×h}
```

Для транспонирования алгоритма $FHT2DT$, следуя обращенным ребрам вычислительного графа для алгоритма $FHT2DT$, мы распространяем (в смысле, описанном в предыдущем пункте) значение $J(t, s)$ каждого пикселя (t, s) входного изображения $J = J_{w\times h}$ на левую (длины w_L) и правую (длины w_R) части паттерна $FHT2DT$ (которые называются подпаттернами) с параметрами (t, s). Это реализовано в строках 4–16 (алгоритм 11). В результате получаются два изображения J_L и J_R шириной w_L и w_R со значениями пикселей входного изображения J, распространенными по левому (J_L) и правому (J_R) подпаттернам. Далее для изображений J_L и J_R вычисляются (рекурсивно) образы I_L и I_R транспонированного преобразования $FHT2DT$ (алгоритм 11, строки 17–18). Выходом транспонированного алгоритма $revFHT2DT$ является конкатенация изображений I_L и I_R по их высотному измерению (алгоритм 11, строка 19).

§ 5. Обсуждение

Была проведена проверка корректности разработанного нами и представленного в предыдущем параграфе алгоритма $revPFHT$ для вычисления оператора об-

ратного проецирования: результаты, полученные при применении к изображениям произвольного размера, согласуются с результатами соответствующего алгоритма из оригинальной работы [1]. Алгоритм $revFHT2DT$ также был протестирован на широкой выборке изображений. Для этого его результаты сравнивалась с результатами работы эталонного алгоритма, который, следуя определению транспонированного БПХ, наивно строит транспонированные паттерны $FHT2DT$ и циклически суммирует значения внутри них. Тестирование не выявило никаких расхождений.

Обсуждая суть предлагаемого подхода, основанного на использовании DAG, мы добавим, что создание формализмов для представления вычислительного процесса в виде ориентированного графа (data flow graph, DFG) продолжается уже около 50 лет, причем DFG является одним из важных внутренних промежуточных представлений [40]. В общем случае DFG не содержит всей информации о программе, но вычислительный граф для операторов проецирования можно сгенерировать без управляющих инструкций, поэтому порядок вычислений определяется только топологией сети DFG.

Графовые представления алгоритмов остаются в центре внимания исследователей и активно используются для распараллеливания и оптимизации [41–43], чтобы максимально использовать масштабируемые гетерогенные архитектуры. Оптимизацию можно выполнять эффективно и автоматически на основе DFG для алгоритмов вычисления операторов как прямого, так и обратного проецирования. Мы также надеемся, что предложенное представление вычислительного графа для алгоритмов вычисления оператора проецирования и автоматизация генерации вычислительной сети оператора обратного проецирования будут способствовать развитию нейросетевых методов с использованием алгоритмов БПХ [44, 45], в том числе в задаче КТ [46–50]. Кроме того, мы считаем, что предложенный подход к транспонированию послужит важным инструментом для изучения самосопряженных БПХ-алгоритмов, обобщающих алгоритм Брейди – Ёна для изображений произвольного размера.

§ 6. Заключение

В данной статье предложен общий метод транспонирования суммирующего алгоритма на основе его DAG-представления. Другими словами, метод позволяет, имея алгоритм вычисления действия прямого суммирующего оператора в виде DAG, эффективно получить результат действия транспонированного оператора. Последнее, как доказано в статье, достигается технически несложной инверсией дуг нормализованного эквипотенциального DAG. Таким образом, предложенный в данной статье метод транспонирования на основе DAG имеет существенное преимущество, поскольку не требует описания входного алгоритма в виде конкретного перемножения булевых матриц. Мы подчеркиваем, что итоговый транспонированный алгоритм, полученный в результате применения предложенного метода транспонирования, обладает той же асимптотической вычислительной сложностью, что и исходный прямой алгоритм (см. [1]).

В статье приведены инструкции по применению общего метода, включающие этап задания порядка вычислений для DAG-представления алгоритма, нормализации вычислительной сети и последующей смены ориентации дуг графа. При этом изложенный метод, в соответствии с инструкциями, был применен в качестве подробного примера к задаче вычисления оператора обратного проецирования в задаче реконструкции в двумерной малоракурсной КТ с параллельно-лучевой схемой. Кроме того, предложенный метод транспонирования был применен к новому алгоритму $FHT2DT$ для быстрого и точного вычисления ПХ. Алгоритм транспонирования $revFHT2DT$ обеспечивает новый вычислительно эффективный подход к вычислению транспонированного ПХ для изображений произвольного размера,

устраняя ограничение транспонированного алгоритма Брейди – Ёна, который применим только к изображениям с шириной, равной степени двойки.

СПИСОК ЛИТЕРАТУРЫ

1. *Polevoy D., Gilmanov M., Kazimirov D., Chukalina M., Ingacheva A., Kulagin P., Nikolaev D.* Tomographic Reconstruction: General Approach to Fast Back-Projection Algorithms // Mathematics. 2023. V. 11. № 23. Paper No. 4759 (37 pp.). https://doi.org/10.3390/math11234759
2. *Hough P.V.C.* Machine Analysis of Bubble Chamber Pictures // Proc. 2nd Int. Conf. on High-Energy Accelerators and Instrumentation (HEACC 1959). CERN, Geneva, Switzerland. Sept. 14–19, 1959. P. 554–558.
3. *Illingworth J., Kittler J.* A Survey of the Hough Transform // Comput. Vision Graph. Image Process. 1988. V. 44. № 1. P. 87–116. https://doi.org/10.1016/S0734-189X(88)80033-1
4. *Chaloeivoot T., Phiphobmongkol S.* Building Detection from Terrestrial Images // J. Image Graph. 2016. V. 4. № 1. P. 46–50.
5. *Rahmdel P.S., Comley R., Shi D., McElduff S.* A Review of Hough Transform and Line Segment Detection Approaches // Proc. 10th Int. Conf. on Computer Vision Theory and Applications (VISAPP 2015). Berlin, Germany. Mar. 11–14, 2015. V. 2. P. 411–418. https://doi.org/10.5220/0005268904110418
6. *Aggarwal N., Karl W.* Line Detection in Images through Regularized Hough Transform // IEEE Trans. Image Process. 2006. V. 15. № 3. P. 582–591. https://doi.org/10.1109/TIP.2005.863021
7. *Mukhopadhyay P., Chaudhuri B.B.* A Survey of Hough Transform // Pattern Recognit. 2015. V. 48. № 3. P. 993–1010. https://doi.org/10.1016/j.patcog.2014.08.027
8. *Алиев М.А., Николаев Д.П., Сараев А.А.* Построение быстрых вычислительных схем настройки алгоритма бинаризации Ниблэка // Тр. ИСА РАН. 2014. Т. 64. № 3. С. 25–34.
9. *Ozturk H., Saricam I.T.* Core Segmentation and Fracture Path Detection Using Shadows // J. Image Graph. 2018. V. 6. № 1. P. 69–73.
10. *Saha S., Basu S., Nasipuri M., Basu D.* A Hough Transform Based Technique for Text Segmentation // J. Comput. 2010. V. 2. № 2. P. 134–141.
11. *Yazdi M., Mohammadi M.* Metal Artifact Reduction in Dental Computed Tomography Images Based on Sinogram Segmentation Using Curvelet Transform Followed by Hough Transform // J. Med. Signals Sens. 2017. V. 7. № 3. P. 145–152.
12. *Brady M.L., Yong W.* Fast Parallel Discrete Approximation Algorithms for the Radon Transform // Proc. 4th Ann. ACM Symp. on Parallel Algorithms and Architectures (SPAA'92). San Diego, California, USA. June 29 – July 1, 1992. P. 91–99. https://doi.org/10.1145/140901.140911
13. *Kazimirov D., Nikolaev D., Rybakova E., Terekhin A.* Generalization of Brady–Yong Algorithm for Fast Hough Transform to Arbitrary Image Size, https://arxiv.org/abs/2411.07351 [cs.CV], 2024.
14. *Kazimirov D., Nikolaev D., Rybakova E., Terekhin A.* Generalization of Brady–Yong Algorithm for Fast Hough Transform to Arbitrary Image Size // Proc. 5th Symp. on Pattern Recognition and Applications (SPRA 2024). Istanbul, Turkey. Nov. 11–13, 2024 (to appear).
15. *Jahan R, Suman P., Singh D.K.* Lane Detection Using Canny Edge Detection and Hough Transform on Raspberry Pi // Int. J. Adv. Res. Comput. Sci. 2018. V. 9. № 2. P. 85–89.
16. *Thongpan N., Rattanasiriwongwut M., Ketcham M.* Lane Detection Using Embedded System // Int. J. Comput. Internet Manag. 2020. V. 28. № 2. P. 46–51.
17. *Panfilova E., Shipitko O.S., Kunina I.* Fast Hough Transform-Based Road Markings Detection For Autonomous Vehicle // 13th Int. Conf. on Machine Vision (ICMV 2020). Rome, Italy. Nov. 2–6, 2020. Proc. SPIE. V. 11605. P. 671–680. https://doi.org/10.1117/12.2587615
18. *Котов А.А., Коноваленко И.А., Николаев Д.П.* Прослеживание объектов в видеопотоке, оптимизированное с помощью быстрого преобразования Хафа // ИтиВС. 2015. № 1. С. 56–68.

19. *van den Braak G.-J., Nugteren C., Mesman B., Corporaal H.* Fast Hough Transform on GPUs: Exploration of Algorithm Trade-Offs // Advanced Concepts For Intelligent Vision Systems: Proc. 13th Int. Conf. ACIVS 2011. Ghent, Belgium. Aug. 22–25, 2011. Lect. Notes Comput. Sci. V. 6915. Berlin: Springer, 2011. P. 611–622. https://doi.org/10.1007/978-3-642-23687-7_55

20. *Brady M.L.* A Fast Discrete Approximation Algorithm for the Radon Transform // SIAM J. Comput. 1998. V. 27. № 1. P. 107–119. https://doi.org/10.1137/S0097539793256673

21. *Prun V.E., Nikolaev D.P., Buzmakov A.V., Chukalina M.V., Asadchikov V.E.* Effective Regularized Algebraic Reconstruction Technique for Computed Tomography // Crystallogr. Rep. 2013. V. 58. № 7. P. 1063–1066. https://doi.org/10.1134/S1063774513070158

22. *Buzug T.M.* Computed Tomography: From Photon Statistics to Modern Cone-Beam CT. Berlin: Springer, 2008. https://doi.org/10.1007/978-3-540-39408-2

23. *Withers P.J., Bouman C., Carmignato S., Cnudde V., Grimaldi D., Hagen C.K., Maire E., Manley M., Du Plessis A., Stock S.R.* X-Ray Computed Tomography // Nat. Rev. Methods Primers. 2021. V. 1. № 1. Article No. 18. https://doi.org/10.1038/s43586-021-00015-4

24. *Arlazarov V.L., Nikolaev D.P., Arlazarov V.V., Chukalina M.V.* X-Ray Tomography: The Way from Layer-by-Layer Radiography to Computed Tomography // Компьютерная оптика. 2021. Т. 45. № 6. С. 897–906. https://doi.org/10.18287/2412-6179-CO-898

25. *Lewitt R.M.* Reconstruction Algorithms: Transform Methods // Proc. IEEE. 1983. V. 71. № 3. P. 390–408. https://doi.org/10.1109/PROC.1983.12597

26. *Dolmatova A., Chukalina M., Nikolaev D.* Accelerated FBP for Computed Tomography Image Reconstruction // Proc. 2020 IEEE Int. Conf. on Image Processing (ICIP 2020). Abu Dhabi, United Arab Emirates. Virtual Conf. Oct. 25–28, 2020. P. 3030–3034. https://doi.org/10.1109/ICIP40778.2020.9191044

27. *Mileto A., Guimaraes L.S., McCollough C.H., Fletcher J.G., Yu. L.* State of the Art in Abdominal CT: The Limits of Iterative Reconstruction Algorithms // Radiology. 2019. V. 293. № 3. P. 491–503. https://doi.org/10.1148/radiol.2019191422

28. *Kasai R., Yamaguchi Y., Kojima T., Abou Al-Ola O.M., Yoshinaga T.* Noise-Robust Image Reconstruction Based on Minimizing Extended Class of Power-Divergence Measures // Entropy. V. 23. № 8. 2021. Paper No. 1005 (16 pp.). https://doi.org/10.3390/e23081005

29. *Kerr J.P., Bartlett E.B.* Neural Network Reconstruction of Single-Photon Emission Computed Tomography Images // J. Digit. Imaging. 1995. V. 8. № 3. P. 116–126. https://doi.org/10.1007/BF03168085

30. *Adler J., Öktem O.* Learned Primal-Dual Reconstruction // IEEE Trans. Med. Imaging. 2018. V. 37. № 6. P. 1322–1332. https://doi.org/10.1109/TMI.2018.2799231

31. *Yamaev A.V., Chukalina M.V., Nikolaev D.P., Kochiev L.G., Chulichkov A.I.* Neural Network Regularization in the Problem of Few View Computed Tomography // Компьютерная оптика. 2022. Т. 46. № 3. С. 422–428. https://doi.org/10.18287/2412-6179-CO-1035

32. *Götz W.A., Druckmüller H.J.* A Fast Digital Radon Transform—An Efficient Means for Evaluating the Hough Transform // Pattern Recognit. 1995. V. 28. № 12. P. 1985–1992. https://doi.org/10.1016/0031-3203(95)00057-7

33. *Wu T.-K., Brady M.L.* A Fast Approximation Algorithm for 3D Image Reconstruction // Proc. 1998 Int. Computer Symp. Workshop on Image Processing and Character Recognition. Tainan, Taiwan. Dec. 17–19, 1998. P. 213–220.

34. *Ершов Е.И., Терехин А.П., Николаев Д.П.* Обобщение быстрого преобразования Хафа для трехмерных изображений // Информационные процессы. 2017. Т. 17. № 4. С. 294–308.

35. *Aliev M., Ershov E.I., Nikolaev D.P.* On the Use of FHT, Its Modification for Practical Applications and the Structure of Hough Image // 11th Int. Conf. on Machine Vision (ICMV 2018). Munich, Germany. Nov. 1–3, 2018. Proc. SPIE. V. 11041. P. 1–9. https://doi.org/10.1117/12.2522803

36. *Bulatov K.B., Chukalina M.V., Nikolaev D.P.* Fast X-Ray Sum Calculation Algorithm for Computed tomography problem // Вестник ЮурГУ ММП. 2020. Т. 13. № 1. С. 95–106. https://doi.org/10.14529/mmp200107

37. *Nikolaev D., Ershov E., Kroshnin A., Limonova E., Mukovozov A., Faradzhev I.* On a Fast Hough/Radon Transform as a Compact Summation Scheme over Digital Straight Line Segments // Mathematics. 2023. V. 15. № 15. Papre No. 3336 (22 pp.). https://doi.org/10.3390/math11153336

38. *Карпенко С.М., Ершов Е.И.* Исследование свойств диадического паттерна быстрого преобразования Хафа // Пробл. передачи информ. 2021. Т. 57. № 3. С. 102–111. https://doi.org/10.31857/S0555292321030074

39. *Ershov E., Terekhin A., Nikolaev D., Postnikov V., Karpenko S.* Fast Hough Transform Analysis: Pattern Deviation from Line Segment // 8th Int. Conf. on Machine Vision (ICMV 2015). Barcelona, Spain. Nov. 19–21, 2015. Proc. SPIE. V. 9875. P. 42–46. https://doi.org/10.1117/12.2228852

40. *Stanier J., Watson D.* Intermediate Representations in Imperative Compilers: A Survey // ACM Comput. Surv. (CSUR). 2013. V. 45. № 3. Article No. 26. P. 1–27. https://doi.org/10.1145/2480741.2480743

41. *Gandarillas V., Joshy A.J., Sperry M.Z., Ivanov A.K., Hwang J.T.*, A Graph-based Methodology for Constructing Computational Models that Automates Adjoint-based Sensitivity Analysis // Struct. Multidiscip. Optim. 2024. V. 67. № 5. P. 76. https://doi.org/10.1007/s00158-024-03792-0

42. *Shingde N., Blattner T., Bardakoff A., Keyrouz W., Berzins M.* An Illustration of Extending Hedgehog to Multi-Node GPU Architectures Using GEMM // SN Comput. Sci. 2024. V. 5. № 5. Article No. 654. https://doi.org/10.1007/s42979-024-02917-y

43. *Bardakoff A.* Analysis and Execution of a Data-Flow Graph Explicit Model Using Static Metaprogramming. Ph.D. Thesis. Université Clermont Auvergne, Clermont-Ferrand, France, 2021. Available at https://theses.hal.science/tel-03813645v1.

44. *Sheshkus A., Ingacheva A., Arlazarov V., Nikolaev D.* HoughNet: Neural Network Architecture for Vanishing Points Detection // Proc. 15th IAPP Int. Conf. on Document Analysis and Recognition (ICDAR 2019). Sept. 20–25, 2019. Sydney, NSW, Australia. P. 844–849. https://doi.org/10.1109/ICDAR.2019.00140

45. *Sheshkus A., Nikolaev D.P., Arlazarov V.L.* Houghencoder: Neural Network Architecture for Document Image Semantic Segmentation // Proc. 2020 IEEE Int. Conf. on Image Processing (ICIP 2020). Abu Dhabi, United Arab Emirates. Virtual Conf. Oct. 25–28, 2020. P. 1946–1950. https://doi.org/10.1109/ICIP40778.2020.9191182

46. *Yamaev A., Chukalina M., Nikolaev D., Sheshkus A. Chulichkov A.* Lightweight Denoising Filtering Neural Network for FBP Algorithm // 13th Int. Conf. on Machine Vision (ICMV 2020). Rome, Italy. Nov. 2–6, 2020. Proc. SPIE. V. 11605. P. 158–167. https://doi.org/10.1117/12.2587185

47. *Ge R., He Y., Xia C., Sun H., Zhang Y., Hu D., Chen S., Chen Y., Li S., Zhang D.* DDPNet: A Novel Dual-Domain Parallel Network for Low-Dose CT Reconstruction // Medical Image Computing and Computer Assisted Intervention – MICCAI 2022: Proc. 25th Int. Conf. Singapore. Sept. 18–22, 2022. Part VI. Lect. Notes Comput. Sci. V. 6915. Cham: Springer, 2022. P. 748–757. https://doi.org/10.1007/978-3-031-16446-0_71

48. *Niu C., Li M., Guo X., Wang G.* Self-supervised Dual-Domain Network for Low-Dose CT Denoising // Developments in X-Ray Tomography XIV. San Diego, California, United States. Aug. 22–24, 2022. Proc. SPIE. V. 12242. P. 85–91. https://doi.org/10.1117/12.2633197

49. *Smolin A., Yamaev A., Ingacheva A., Shevtsova T., Polevoy D., Chukalina M., Nikolaev D., Arlazarov V.* Reprojection-based Numerical Measure of Robustness for CT Reconstruction Neural Networks Algorithms // Mathematics. 2022. V. 10. № 22. Paper No. 4210 (17 pp.). https://doi.org/10.3390/math10224210

50. *Kojima T., Yoshinaga T.* Iterative Image Reconstruction Algorithm with Parameter Estimation by Neural Network for Computed Tomography // Algorithms. 2023. V. 16. № 1. Paper No. 60 (18 pp.). https://doi.org/10.3390/a16010060

Полевой Дмитрий Валерьевич
Институт проблем передачи информации
им. А.А. Харкевича РАН, Москва
Федеральный исследовательский центр
"Информатика и управление" РАН, Москва
ООО "Смарт Энджинс Сервис", Москва
dvpsun@gmail.com
Казимиров Данил Дмитриевич
Институт проблем передачи информации
им. А.А. Харкевича РАН, Москва
ООО "Смарт Энджинс Сервис", Москва
d.kazimirov@smartengines.com
Чукалина Марина Валерьевна
Николаев Дмитрий Петрович
Федеральный исследовательский центр
"Информатика и управление" РАН, Москва
ООО "Смарт Энджинс Сервис", Москва
m.chukalina@smartengines.com
d.p.nikolaev@smartengines.com

Поступила в редакцию
18.10.2024
После доработки
06.12.2024
Принята к публикации
25.12.2024