

Эффективные численные методы решения задачи PageRank для дважды разреженных матриц

А. С. Аникин¹, А. В. Гасников^{2,3}, А. Ю. Горнов¹, Д. И. Камзолов^{2,3},
Ю. В. Максимов^{3,4}, and Ю. Е. Нестеров^{4,5}

¹Институт динамики систем и теории управления СО РАН

²НИУ Московский физико-технический институт

³Институт проблем передачи информации РАН

⁴Факультет Компьютерных Наук, НИУ “Высшая школа экономики”

⁵Center for Operational research and econometrics of the Catholique Universite de Louvain, Belgium

Аннотация

В работе приводятся три метода поиска вектора PageRank (вектора Фробениуса–Перрона стохастической матрицы) для дважды разреженных матриц. Все три метода сводят поиск вектора PageRank к решению задачи выпуклой оптимизации на симплексе (или седловой задаче). Первый метод базируется на обычном градиентном спуске. Однако особенностью этого метода является выбор нормы l_1 вместо привычной евклидовой нормы. Второй метод базируется на алгоритме Франк–Вульфа. Третий метод базируется на рандомизированном варианте метода зеркального спуска. Все три способа хорошо учитывают разреженность постановки задачи.

Ключевые слова: PageRank, разреженность, рандомизация, метод Франк–Вульфа, l_1 -оптимизация.

1. Введение

В данной работе мы сконцентрируемся на решении задачи PageRank и ее окрестностях. Хорошо известно (Брин–Пейдж, 1998 [1, 2]), что задача ранжирования web-страниц приводит к поиску вектора Фробениуса–Перрона стохастической матрицы $x^T P = x$. Размеры матрицы могут быть колоссальными (в современных реалиях это сто миллиардов на сто миллиардов). Выгрузить такую матрицу в оперативную память обычного компьютера не представляется возможным. Задачу PageRank можно переписать, как задачу выпуклой оптимизации (Назин–Поляк, 2011 [3]; Ю.Е. Нестеров, 2012 [4,5]; Гасников–Дмитриев, 2015 [6]) в разных вариантах: минимизация квадратичной формы $\|Ax - b\|_2^2$ и минимизация бесконечной нормы $\|Ax - b\|_\infty$ на единичном симплексе. Здесь $A = P^T - I$, I — единичная матрица. К аналогичным задачам приводят задачи анализа данных (Ridge regression, LASSO [7]), задачи восстановления матрицы корреспонденций по замерам трафика на линках в компьютерных сетях [8] и многие другие задачи. Особенностью постановок этих задач, также как и в случае задачи PageRank, являются колоссальные размеры.

В данной статье планируется сосредоточиться на изучении роли разреженности матрицы A , на использовании рандомизированных подходов и на специфике множества, на котором происходит оптимизация. Симплекс является (в некотором смысле) наилучшим возможным множеством, которое может порождать (независимо от разреженности матрицы A) разреженность решения (см., например, п. 3.3 С. Бубек, 2014 [9], это тесно связано с l_1 -оптимизацией [10]).

Все о чем здесь написано хорошо известно на таком уровне грубости. Поясним, дополнительно погружаясь в детали, в чем заключаются отличия развиваемых в статье подходов от известных подходов. Прежде всего, мы вводим специальный класс *дважды разреженных матриц* (одновременно разреженных и по строкам и по столбцам).¹ Такие матрицы, например, возникают в методе конечных элементов, и в более общем контексте при изучении разностных схем.² Если считать, что матрица A имеет размеры $n \times n$, а число элементов в каждой строке и столбце не больше чем $s \ll n$, то число ненулевых элементов в матрице может быть sn . Кажется, что это произведение точно должно возникать в оценках общей сложности (числа арифметических операций, типа умножения или сложения двух чисел типа double) решения задачи (с определенной фиксированной точностью). Оказывается, что для первой постановки (минимизация квадратичной формы на симплексе) сложность может быть сделана пропорциональна $s^2 \ln(2 + n/s^2)$ (разделы 2 и 3), а для второй (минимизация бесконечной нормы) и того меньше — $s \ln n$ (раздел 4).

Первая задача может решаться обычным прямым градиентным методом с 1-нормой в прямом пространстве [11], [12] — нетривиальным тут является, в том числе, организация работы с памятью. В частности, требуется хранение градиента в массиве, обновление элементов которого и вычисления максимального/минимального элемента должно осуществляться за время, не зависящие от размера массива (то есть от n). Тут оказываются полезными фибоначчиевы и бродалевские кучи [13]. Основная идея — не рассчитывать на каждом шаге градиент функции заново $A^T A x_{k+1}$, а пересчитывать его, учитывая, что $x_{k+1} = x_k + e_k$, где вектор e_k состоит в основном из нулей:

$$A^T A x_{k+1} = A^T A x_k + A^T A e_k$$

Детали будут изложены в разделе 2. Аналогичную оценку общей сложности планируется получить методом условного градиента Франк–Вульфа [14], большой интерес к которому появился в последние несколько лет в основном в связи с задачами BigData (М. Ягги [15], Аршуи–Юдицкий–Немировский [16], Ю.Е. Нестеров [17]). Аккуратный анализ работы этого метода также при правильной организации работы с памятью позволяет (аналогично предыдущему подходу) находить такой x , что $\|Ax - b\|_2 \leq \varepsilon$ за число арифметических операций

$$O\left(n + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2}\right)$$

причем оценка оказывается вполне практической. Этому будет посвящен раздел 3.

Вторая задача (минимизации $\|Ax - b\|_\infty$ на единичном симплексе) с помощью техники Юдицкого–Немировского, 2012 (см. п. 6.5.2.3 [18]) может быть переписана как седловая задача на произведение двух симплексов. Если применять рандомизированный метод зеркального спуска (основная идея которого заключается в вычислении вместо градиента Ax стохастического градиента, рассчитываемого по следующему правилу: с вероятностью x_1 он равен первому столбцу матрицы A , с вероятностью x_2 второму и т.д.), то для того, чтобы обеспечить с вероятностью $\geq 1 - \sigma$ неравенство $\|Ax - b\|_\infty \leq \varepsilon$ достаточно выполнить

$$O(n \ln(n/\sigma)/\varepsilon^2)$$

арифметических операций. К сожалению, этот подход не позволяет в полном объеме учесть разреженность матрицы A . В статье предлагается другой путь, восходящий к работе Григориадиса–Хачияна, 1995 [19] (см. также [20]). В основе этого подхода лежит рандомизация не при вычислении градиента, а при последующем проектировании (согласно

¹Заметим, что за счет специального “раздутья” изначальной матрицы можно обобщить приведенные в статье оценки и подходы с класса дважды разреженных матриц, на матрицы, у которых есть небольшое число полных строк и/или столбцов. Такие задачи встречаются в приложениях заметно чаще.

²Применительно к ранжированию web-страниц, это свойство означает, что входящие и исходящие степени всех web-страниц равномерно ограничены.

KL -расстоянию, что отвечает экспоненциальному взвешиванию) градиента на симплекс. Предлагается вместо честной проекции на симплекс случайно выбирать одну из вершин симплекса (разреженный объект!) так, чтобы математическое ожидание проекции равнялось бы настоящей проекции [20]. В работе планируется показать, что это можно эффективно делать, используя специальные двоичные деревья пересчета компонент градиента [6]. В результате планируется получить следующую оценку для дважды разреженных матриц

$$O(n + s \ln n \ln(n/\sigma)/\varepsilon^2).$$

Напомним, что при этом число ненулевых элементов в матрице A может быть sn . Возникает много вопросов относительно того насколько все это практично. К сожалению, на данный момент наши теоретические оценки говорят о том, что константа в $O()$ может иметь порядок 10^2 . Подробнее об этом будет написано в разделе 4.

В данной статье при специальных предположениях относительно матрицы PageRank P (дважды разреженная) мы предлагаем методы, которые работают по наилучшим известным сейчас оценкам для задачи PageRank в условиях отсутствия контроля спектральной щели матрицы PageRank [6]. А именно, полученная в статье оценка (см. разделы 2, 3)

$$O\left(n + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2}\right)$$

сложности поиска такого x , что $\|Ax - b\|_2 \leq \varepsilon$, является на данный момент наилучшей при $s \ll \sqrt{n}$ для данного класса задач. Быстрее может работать только метод МСМС (для матрицы с одинаковыми по строкам внедиагональными элементами)

$$O(\ln n \ln(n/\sigma)/(\alpha\varepsilon^2))$$

требующий, чтобы спектральная щель α была достаточно большой [6].

Приведенные в статье подходы, применимы не только к квадратным матрицам A и не только к задаче PageRank. В частности, можно обобщать приведенные в статье подходы на композитные постановки [21]. Тем не менее, свойство двойной разреженности матрицы A является существенным. Если это свойство не выполняется, и имеет место, скажем, только разреженность в среднем по столбцам, то приведенные в статье подходы могут быть доминируемы рандомизированными покомпонентными спусками. Если матрица A сильно вытянута по числу строк (такого рода постановки характерны для задач анализа данных), то покомпонентные методы применяются к двойственной задаче [22], [23], если по числу столбцов (такого рода постановки характерны для задач поиска равновесных конфигураций в больших транспортных/компьютерных сетях), то покомпонентные методы применяются к прямой задаче [24], [25]. Подчеркнем, что при условии двойной разреженности A с $s \ll \sqrt{n}$ нам не известны более эффективные (приведенных в данной статье) способы решения описанных задач. В частности, это относится и к упомянутым выше покомпонентным спускам.

В заключительном пятом разделе работы приводятся результаты численных экспериментов.

2. Прямой градиентный метод в 1-норме

Задача поиска вектора PageRank может быть сведена к следующей задаче выпуклой оптимизации [12] (далее для определенности будем полагать $\gamma = 1$, в действительности, по этому параметру требуется прогонка)

$$f(x) = \frac{1}{2}\|Ax\|_2^2 + \frac{\gamma}{2} \sum_{i=1}^n (-x^i)_+^2 \rightarrow \min_{\langle x, e \rangle = 1}$$

где $A = P^T - I$, I — единичная матрица, $e = (1, \dots, 1)^T$,

$$(y)_+ = \begin{cases} y, & y \geq 0 \\ 0, & y < 0 \end{cases}$$

При этом мы считаем, что в каждом столбце и каждой строке матрицы P не более $s \ll \sqrt{n}$ элементов отлично от нуля (P — разрежена). Эту задачу предлагается решать обычным градиентным методом, но не в евклидовой норме, а в 1-норме (см., например, [11]):

$$x_{k+1} = x_k + \operatorname{argmin}_{h: \langle h, e \rangle = 0} \left\{ f(x_k) + \langle \nabla f(x_k), h \rangle + \frac{L}{2} \|h\|_1^2 \right\}$$

где $L = \max_{i=1, \dots, n} \|A^{(i)}\|_2^2 + 1 \leq 3$ ($A^{(i)}$ — i -й столбец матрицы A). Точку старта x_0 итерационного процесса выберем в одной из вершин симплекса.

Для достижения точности ε^2 по функции потребуется сделать

$$O(LR^2/\varepsilon^2) = O(1/\varepsilon^2)$$

итераций [11]. Не сложно проверить, что пересчет градиента на каждой итерации заключается в умножении $A^T A h$, что может быть сделано за $O(s^2)$. Связано это с тем, что вектор h всегда имеет только две компоненты

$$\frac{1}{8} \left(\max_{i=1, \dots, n} \partial f(x_k) / \partial x^i - \min_{i=1, \dots, n} \partial f(x_k) / \partial x^i \right) \text{ и } -\frac{1}{8} \left(\max_{i=1, \dots, n} \partial f(x_k) / \partial x^i - \min_{i=1, \dots, n} \partial f(x_k) / \partial x^i \right)$$

отличные от нуля (такая разреженность получилась благодаря выбору 1-нормы), причем эти компоненты определяются, соответственно, как

$$\operatorname{argmin}_{i=1, \dots, n} \partial f(x_k) / \partial x^i \text{ и } \operatorname{argmax}_{i=1, \dots, n} \partial f(x_k) / \partial x^i$$

Это можно пересчитывать (при использовании кучи для поддержания максимальной и минимальной компоненты градиента) за $O(s^2 \ln(2 + n/s^2))$. Указанная оценка достигается следующим образом: группа из n координат разбивается на $s^2 + 1$ непересекающихся групп по не более, чем n/s^2 в каждой. Операция обновления компоненты градиента осуществляется в соответствующей ей группе (за время $O(\ln(2 + n/s^2))$ с помощью бинарной кучи [13]). Выбор минимальной компоненты градиента осуществляется просмотром не более чем $s^2 + 1$ вершин кучи (в вершине кучи расположен минимальный элемент). Аналогичным образом поддерживается набор куч с максимальным элементом в корне для поиска максимальной компоненты градиента. Важно отметить, что логарифмический фактор указанного вида не улучшаем. Последнее следует из неулучшаемости оценки скорости сортировки $O(n \ln n)$ основанной на попарном сравнении элементов. Таким образом, с учетом препроцессинга и стоимости первой итерации $O(n)$ общая трудоемкость предложенного метода будет

$$O \left(n + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2} \right)$$

что заметно лучше многих известных методов [6].

Некоторая проблема состоит в том, что в решении могут быть маленькие отрицательные компоненты, и требуется прогонка по γ . Также для того, чтобы сполна использовать разреженность требуется либо “структура разреженности” (скажем, заранее известно, что матрица A имеет отличные от нуля элементы только в $s/2$ окрестности диагонали), либо препроцессинг. В нашем случае (впрочем, это относится и к последующим разделам) препроцессинг заключается в представлении матрицы по строкам в виде списка смежности: в каждой строке отличный от нуля элемент хранит ссылку на следующий отличный от нуля элемент, аналогичное представление матрицы делается и по столбцам. Заметим, что

препроцессинг помогает ускорять решения задач не только в связи с более полным учетом разреженности постановки, но и, например, в связи с более эффективной организацией рандомизации [6].

Обратим внимание на то, что число элементов в матрице P , отличных от нуля, даже при наложенном условии разреженности (по строкам и столбцам), все равно может быть достаточно большим sn . Удивляет то, что в оценке общей трудоемкости этот размер фактически никак не присутствует. Это в перспективе (при правильной организации работы с памятью) позволяет решать задачи колоссальных размеров. Более того, даже в случае небольшого числа не разреженных ограничений вида

$$\langle a_i, x \rangle = b_i, \quad i = 1, \dots, m = O(1),$$

можно “раздуть” пространство (не более чем в два раза), в котором происходит оптимизация (во многих методах, которые учитывают разреженность, это число входит подобно рассмотренному примеру, так что такое раздутие не приведет к серьезным затратам), и переписать эту систему в виде $Ax = b$, где матрица будет иметь размеры $O(n) \times O(n)$, но число отличных от нуля элементов в каждой строке и столбце будет $O(1)$. Таким образом, допускается небольшое число “плотных” ограничений.

В заключение заметим, что после переформулировки исходной задачи мы ушли от оптимизации на симплексе, и стали оптимизировать на гиперплоскости, содержащей симплекс. Проблема тут возникает из-за того, что нужно оценить l_1 -размер множества Лебега функции $f(x)$ для уровня $f(x_0)$, где $x_0 \in S_n(1)$ — точка старта (см. [11]), $S_n(1)$ — единичный симплекс в \mathbb{R}^n . Можно показать, что это приводит к уже использованной нами выше оценке $R = O(1)$. Но, к сожалению, точного значения R мы не знаем. Как следствие, при практической реализации метода мы не можем просто сделать предписанное число итераций и остановиться. Однако выход есть. В данной задаче мы знаем оптимальное значение: $f_* = 0$. Сделав $N = 1/\varepsilon^2$ итераций можно за $O(sn)$ арифметических операций проверить, выполняется ли условие $f(x_N) \leq \varepsilon^2/2$ или нет. Если нет, то $N = 3N$ и продолжаем итерационный процесс (брать ли число 3 или какое-то другое > 1 можно определить исходя из того, как соотносятся sn и s^2/ε^2 , чем больше второе первого, тем меньше надо брать это число). Минус тут в том, что тогда оценка получится

$$O\left(sn + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2}\right),$$

а не

$$O\left(n + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2}\right),$$

как хотелось бы. С другой стороны, избавляться совсем от sn в данном случае и не обязательно, поскольку в следующем разделе предлагается метод, где этих проблем нет.

3. Метод условного градиента Франк–Вульфа

Перепишем задачу поиска вектора PageRank следующим образом

$$f(x) = \frac{1}{2} \|Ax\|_2^2 \rightarrow \min_{x \in S_n(1)}$$

Для решения этой задачи будем использовать метод условного градиента Франк–Вульфа [14] – [17]. Напомним, в чем состоит метод.

Выберем одну из вершин симплекса и возьмем точку старта x_1 в этой вершине. Далее по индукции, шаг которой имеет следующий вид. Решаем задачу

$$\langle \nabla f(x_k), y \rangle \rightarrow \min_{y \in S_n(1)}$$

Обозначим решение этой задачи через

$$y_k = (0, \dots, 0, 1, 0, \dots, 0)$$

где 1 стоит на позиции

$$i_k = \operatorname{argmin}_{i=1, \dots, n} \partial f(x_k) / \partial x^i$$

Положим

$$x_{k+1} = (1 - \gamma_k)x_k + \gamma_k y_k, \quad \gamma_k = \frac{2}{k+1}, \quad k = 1, 2, \dots$$

Имеет место следующая оценка [14] – [17]

$$f(x_N) - f_* \leq \frac{2L_p R_p^2}{N+1},$$

где

$$R_p^2 \leq \max_{x, y \in S_n(1)} \|y - x\|_p^2, \quad L_p \leq \max_{\|h\|_p \leq 1} \langle h, A^T A h \rangle = \max_{\|h\|_p \leq 1} \|A h\|_2^2, \quad 1 \leq p \leq \infty.$$

С учетом того, что оптимизация происходит на симплексе, мы выберем $p = 1$. Не сложно показать, что этот выбор оптимален. В результате получим, что $R_1^2 = 4$,

$$L_1 = \max_{i=1, \dots, n} \|A^{(i)}\|_2^2 \leq 2.$$

Таким образом, чтобы $f(x_N) \leq \varepsilon^2/2$ достаточно сделать $N = 32\varepsilon^{-2}$ итераций. В действительности, число 32 можно почти на порядок уменьшить немного более тонкими рассуждениями (детали мы вынуждены здесь опустить).

Сделав дополнительные вычисления стоимостью $O(n)$, можно так организовать процедуру пересчета $\nabla f(x_k)$ и вычисления $\operatorname{argmin}_{i=1, \dots, n} \partial f(x_k) / \partial x^i$, что каждая итерация будет иметь сложность $O(s^2 \ln(2 + n/s^2))$. Для этого вводим

$$\beta_k = \prod_{r=1}^{k-1} (1 - \gamma_r), \quad z_k = x_k / \beta_k, \quad \tilde{\gamma}_k = \gamma_k / \beta_{k+1}.$$

Тогда итерационный процесс можно переписать следующим образом

$$z_{k+1} = z_k + \tilde{\gamma}_k y_k.$$

Пересчитывать $A^T A z_{k+1}$ при известном значении $A^T A z_k$ можно за $O(s^2 \ln(2 + n/s^2))$. Далее, задачу

$$i_{k+1} = \operatorname{argmin}_{i=1, \dots, n} \partial f(x_{k+1}) / \partial x^i$$

можно переписать как

$$i_{k+1} = \operatorname{argmin}_{i=1, \dots, n} (A^T A z_{k+1})^i.$$

Поиск i_{k+1} можно также осуществить за $O(s^2 \ln(2 + n/s^2))$. Определив в конечном итоге z_N , мы можем определить x_N , затратив дополнительно не более

$$O(n + \ln N) = O(n)$$

арифметических операций. Таким образом, итоговая оценка сложности описанного метода будет

$$O\left(n + \frac{s^2 \ln(2 + n/s^2)}{\varepsilon^2}\right).$$

Стоит отметить, что функционал, выбранный в этом примере, обеспечивает намного лучшую оценку $\|Ax\|_2 \leq \varepsilon$ по сравнению с функционалом из раздела 4, который обеспечивает лишь $\|Ax\|_\infty \leq \varepsilon$. Наилучшая (в разреженном случае без, условий на спектральную цель матрицы P [6]) из известных нам на данный момент оценок

$$O(n + s \ln n \ln(n/\sigma)/\varepsilon^2)$$

(см. раздел 4) для $\|Ax\|_\infty$ может быть улучшена приведенной в этом разделе оценкой, поскольку $\|Ax\|_2$ может быть (и так часто бывает) в $\sim \sqrt{n}$ раз больше $\|Ax\|_\infty$, а $s \ll \sqrt{n}$.

4. Седловое представление задачи PageRank и рандомизированный зеркальный спуск в форме Григориадиаса–Хачияна

Перепишем задачу поиска вектора PageRank следующим образом

$$f(x) = \|Ax\|_\infty \rightarrow \min_{x \in S_n(1)}$$

Следуя [18] эту задачу можно переписать седловым образом

$$\min_{x \in S_n(1)} \max_{\|y\|_1 \leq 1} \langle Ax, y \rangle.$$

В свою очередь последнюю задачу можно переписать как

$$\min_{x \in S_n(1)} \max_{\omega \in S_{2n}(1)} \langle Ax, J\omega \rangle,$$

где

$$J = [I_n, -I_n].$$

В итоге задачу можно переписать, с сохранением свойства разреженности, как

$$\min_{x \in S_n(1)} \max_{\omega \in S_{2n}(1)} \langle \omega, \tilde{A}x \rangle.$$

Следуя [20], опишем эффективный и интерпретируемый способ решения этой задачи. Пусть есть два игрока А и Б. Задана матрица (антагонистической) игры $\tilde{A} = \|\tilde{a}_{ij}\|$, где $|\tilde{a}_{ij}| \leq 1$, \tilde{a}_{ij} — выигрыш игрока А (проигрыш игрока Б) в случае, когда игрок А выбрал стратегию i , а игрок Б стратегию j . отождествим себя с игроком Б. И предположим, что игра повторяется $N \gg 1$ раз (это число может быть заранее неизвестно [20], однако для простоты изложения будем считать это число известным). Введем функцию потерь (игрока Б) на шаге k

$$f_k(x) = \langle \omega^k, \tilde{A}x \rangle, \quad x \in S_n(1),$$

где $\omega^k \in S_{2n}(1)$ — вектор (вообще говоря, зависящий от всей истории игры до текущего момента включительно, в частности, как-то зависящий и от текущей стратегии (не хода) игрока Б, заданной распределением вероятностей (результат текущего разыгрывания (ход Б) игроку А не известен)) со всеми компонентами равными 0, кроме одной компоненты, соответствующей ходу А на шаге k , равной 1. Хотя функция $f_k(x)$ определена на единичном симплексе, по “правилам игры” вектор x^k имеет ровно одну единичную компоненту, соответствующую ходу Б на шаге k , остальные компоненты равны нулю. Обозначим цену игры (в нашем случае $C = 0$)

$$C = \max_{\omega \in S_{2n}(1)} \min_{x \in S_n(1)} \langle \omega, \tilde{A}x \rangle = \min_{x \in S_n(1)} \max_{\omega \in S_{2n}(1)} \langle \omega, \tilde{A}x \rangle. \text{ (теорема фон Неймана о минимаксе)}$$

Пару векторов (ω, x) , доставляющих решение этой минимаксной задачи (т.е. седловую точку), назовем равновесием Нэша. По определению (это неравенство восходит к Ханнани [26])

$$\min_{x \in S_n(1)} \frac{1}{N} \sum_{k=1}^N f_k(x) \leq C.$$

Тогда, если мы (игрок Б) будем придерживаться следующей рандомизированной стратегии (см., например, [6], [19], [20]), выбирая $\{x^k\}$:

- 1) $p^1 = (n^{-1}, \dots, n^{-1})$;
- 2) Независимо разыгрываем случайную величину $j(k)$ такую, что $P(j(k) = j) = p_j^k$;
- 3) Полагаем $x_{j(k)}^k = 1$, $x_j^k = 0$, $j \neq j(k)$;
- 4) Пересчитываем

$$p_j^{k+1} \sim p_j^k \exp \left(- \sqrt{\frac{2 \ln n}{N}} \cdot \tilde{a}_{i(k)j} \right), \quad j = 1, \dots, n,$$

где $i(k)$ — номер стратегии, которую выбрал игрок А на шаге k ;³

то с вероятностью $\geq 1 - \sigma$

$$\frac{1}{N} \sum_{k=1}^N f_k(x^k) - \min_{x \in S_n(1)} \frac{1}{N} \sum_{k=1}^N f_k(x) \leq \sqrt{\frac{2}{N}} \left(\sqrt{\ln n} + 2\sqrt{2 \ln(\sigma^{-1})} \right),$$

т.е. с вероятностью $\geq 1 - \sigma$ наши (игрока Б) потери ограничены

$$\frac{1}{N} \sum_{k=1}^N f_k(x^k) \leq C + \sqrt{\frac{2}{N}} \left(\sqrt{\ln n} + 2\sqrt{2 \ln(\sigma^{-1})} \right).$$

Самый плохой для нас случай (с точки зрения такой оценки) — это когда игрок А тоже действует (выбирая $\{\omega^k\}$) согласно аналогичной стратегии (только игрок А решает задачу максимизации).⁴ Если и А и Б будут придерживаться таких стратегий, то они сойдутся к равновесию Нэша (седловой точке), причем довольно быстро [20]: с вероятностью $1 - \sigma$

$$0 \leq \|A\bar{x}^N\|_\infty = \max_{\omega \in S_{2n}(1)} \langle \omega, \tilde{A}\bar{x}^N \rangle - \max_{\omega \in S_{2n}(1)} \min_{x \in S_n(1)} \langle \omega, \tilde{A}x \rangle \leq \max_{\omega \in S_{2n}(1)} \langle \omega, \tilde{A}\bar{x}^N \rangle - \min_{x \in S_n(1)} \langle \bar{\omega}^N, \tilde{A}x \rangle \leq$$

³Заметим, что эта стратегия имеет естественную интерпретацию. Мы (игрок Б) описываем на текущем шаге игрока А вектором, компоненты которого — сколько раз игрок А использовал до настоящего момента соответствующую стратегию. Согласно этому вектору частот мы рассчитываем вектор своих ожидаемых потерь (при условии, что игрок А будет действовать согласно этому частотному вектору). Далее, вместо того, чтобы выбирать наилучшую (для данного вектора частот А) стратегию, дающую наименьшие потери, мы используем распределение Гиббса с параметром $\sqrt{2 \ln n / N}$ (экспоненциально взвешиваем с этим параметром вектор ожидаемых потерь с учетом знака). С наибольшей вероятностью будет выбрана наилучшая стратегия, но с ненулевыми вероятностями могут быть выбраны и другие стратегии.

⁴Игрок А пересчитывает

$$p_i^{k+1} \sim p_i^k \exp \left(\sqrt{\frac{2 \ln(2n)}{N}} \cdot \tilde{a}_{ij(k)} \right), \quad i = 1, \dots, 2n,$$

где $j(k)$ — номер стратегии, которую выбрал игрок Б на шаге k .

$$\begin{aligned}
&\leq \max_{\omega \in S_{2n}(1)} \langle \omega, \tilde{A}\bar{x}^N \rangle - \frac{1}{N} \sum_{k=1}^N \langle \omega^k, \tilde{A}x^k \rangle + \frac{1}{N} \sum_{k=1}^N \langle \omega^k, \tilde{A}x^k \rangle - \min_{x \in S_n(1)} \langle \bar{\omega}^N, \tilde{A}x \rangle \leq \\
&\leq \sqrt{\frac{2}{N}} \left(\sqrt{\ln(2n)} + 2\sqrt{2\ln(2/\sigma)} \right) + \sqrt{\frac{2}{N}} \left(\sqrt{\ln n} + 2\sqrt{2\ln(2/\sigma)} \right) \leq \\
&\leq 2\sqrt{\frac{2}{N}} \left(\sqrt{\ln(2n)} + 2\sqrt{2\ln(2/\sigma)} \right)
\end{aligned}$$

где

$$\bar{x}^N = \frac{1}{N} \sum_{k=1}^N x^k, \quad \bar{\omega}^N = \frac{1}{N} \sum_{k=1}^N \omega^k.$$

Таким образом, чтобы с вероятностью $\geq 1 - \sigma$ иметь $\|A\bar{x}^N\|_\infty \leq \varepsilon$, достаточно сделать

$$N = 16 \frac{\ln(2n) + 8\ln(2/\sigma)}{\varepsilon^2} \text{ — итераций;}$$

$$O\left(n + \frac{s \ln n (\ln n + \ln(\sigma^{-1}))}{\varepsilon^2}\right) \text{ — общее число арифметических операций,}$$

где число $s \ll \sqrt{n}$ — ограничивает сверху, число ненулевых элементов в строках и столбцах матрицы P .

Нетривиальным местом здесь является оценка сложности одной итерации $O(s \ln n)$. Получается эта оценка исходя из оценки наиболее тяжелых действий шаг 2 и 4. Сложность тут в том, что чтобы сгенерировать распределение дискретной случайной величины, принимающей n различных значений (в общем случае) требуется $O(n)$ арифметических операций — для первого генерирования (приготовления памяти). Последующие генерирования могут быть сделаны эффективнее — за $O(\ln n)$. Однако в нашем случае есть специфика, заключающаяся в том, что при переходе на следующий шаг s вероятностей в распределении могли как-то поменяться. Если не нормировать распределение вероятностей, то можно считать, что остальные вероятности остались неизменными. Оказывается, что такая специфика позволяет вместо оценки $O(n)$ получить оценку $O(s \ln n)$.

Замечание. Опишем точнее эту процедуру. У нас есть сбалансированное двоичное дерево высоты $O(\log_2 n)$ с n листом (дабы не вдаваться в технические детали, считаем что число n — есть степень двойки, понятно, что это не так, но можно взять, скажем, наименьшее натуральное m такое, что $2^m > n$ и рассматривать дерево с 2^m листом) и с $O(n)$ общим числом вершин. Каждая вершина дерева (отличная от листа) имеет неотрицательный вес равный сумме весов двух ее потомков. Первоначальная процедура приготовления дерева, отвечающая одинаковым весам листьев, потребует $O(n)$ операций. Но сделать это придется всего один раз. Процедура генерирования дискретной случайной величины с распределением, с точностью до нормирующего множителя, соответствующим весам листьев может быть осуществлена с помощью случайного блуждания из корня дерева к одному из листьев. Отметим, что поскольку дерево двоичное, то прохождение каждой его вершины при случайном блуждании, из которой идут два ребра в вершины с весами $a > 0$ и $b > 0$, осуществляется путем подбрасывания монетки (“приготовленной” так, что вероятность пойти в вершину с весом a — есть $a/(a+b)$). Понятно, что для осуществления этой процедуры нет необходимости в дополнительном условии нормировки: $a+b=1$. Если вес какого-то листа алгоритму необходимо поменять по ходу работы, то придется должным образом поменять дополнительно веса тех и только тех вершин, которые лежат на пути из корня к этому листу. Это необходимо делать, чтобы поддерживать свойство: каждая вершина дерева (отличная от листа) имеет вес равный сумме весов двух ее потомков.

Итак, выше в этом разделе была описана процедура генерирования последовательности x^k со сложностью

$$O(n + s \ln n \ln(n/\sigma)/\varepsilon^2),$$

на основе которой можно построить такой частотный вектор

$$\bar{x}^N = \frac{1}{N} \sum_{k=1}^N x^k,$$

компоненты — частоты, с которыми мы использовали соответствующие стратегии, что $\|A\bar{x}^N\|_\infty \leq \varepsilon$ с вероятностью $\geq 1 - \sigma$.

5. Программная реализация

Рассматриваемые в работе методы реализованы авторами в рамках единой программной системы на языке C++. Работоспособность и корректность реализации проверена с использованием компиляторов GCC (GNU Compiler Collection, версии: 4.8.4, 4.9.3, 5.2.1), clang (C language family frontend for LLVM, версии: 3.4.2, 3.5.2, 3.6.1, 3.7.0), icc (Intel C Compiler, версия 15.0.3) на операционных системах GNU/Linux, Microsoft Windows и Mac OS X.

Для удобства введем следующие обозначения рассматриваемых в работе методов: NL1 — метод из раздела 2, FW — метод из раздела 3 и GK — метод из раздела 4.

5.1. Хранение разреженной матрицы

Для хранения данных разреженной матрицы A используется широко распространенный формат CSR (Compressed Sparse Row), позволяющий работать с матрицами произвольной структуры. Поскольку для рассматриваемых методов существует необходимость обращения к элементам как матрицы A (вычисление функции) так и матрицы A^T (вычисление градиента), в текущей программной реализации осуществляется хранение в памяти обеих матриц. Это, естественно, влечет за собой удвоение объема потребляемой оперативной памяти, но радикальным образом улучшает быстродействие соответствующих вычислительных процедур.

5.2. Поиск минимального/максимального компонента градиента

Методы NL1 и FW требуют наличия информации о минимальной/максимальной компоненте градиента — номере (индексе) соответствующей переменной. Поскольку идеология методов подразумевает выполнение большого числа “легких” итераций, поиск таких компонент должен быть эффективно реализован. С учетом того, что все рассматриваемые методы учитывают разреженность решаемой задачи и, в итоге, производят модификацию конечного (и весьма небольшого относительно n) числа компонент градиента, в текущей реализации применяется подход, предложенный в работе [4]. Основной его идеей является построение бинарного дерева, реализующего вычисление требуемой функции от n переменных (в нашем случае — \min/\max), и механизм быстрого обновления (за $O(\ln n)$) вычисленного ранее значения в случае изменения одной переменной (вместо “полного” поиска за $O(n)$).

Реализация таких деревьев выполнена в виде шаблонных классов C++, что позволяет задавать требуемую функцию-обработчик данных на этапе компиляции и обеспечивает максимальное быстродействие при вызове этой функции относительно других вариантов реализации: функторов (`std::function + lambda`), указателей на функцию или применение виртуального метода.

Длина обрабатываемого деревом вектора, в общем случае, не равна степени двойки, поэтому реализованные классы деревьев создают, при необходимости, дополнительные узлы на соответствующих уровнях и при обработке данных учитывают данное обстоятельство, напрямую “проталкивая” данные таких “некратных” узлов на вышестоящий уровень для последующей обработки.

Поскольку используемые в работе деревья должны реализовывать не только поиск минимального/максимального значений градиента, но и соответствующего им индекса, то в каждом узле такого дерева хранятся и обрабатываются пары “индекс-значение”. Такой подход увеличивает объем памяти, занимаемый деревом, но существенным образом улучшает его быстродействие, поскольку исключается необходимость обращения к, в общем случае случайным, элементам вектора градиента, что весьма затратно с точки зрения работы кэша современных процессоров. Такое “совместное” хранение пар “индекс-значение” существенно улучшает “локальность” данных для процессора при обработке узлов.

Также для рассматриваемых деревьев реализован механизм отсечений — если получаемый функцией-обработчиком результат от данных с “дочерних” узлов не меняет содержимое текущего узла, то, очевидно, дальнейшее “продвижение” на вышестоящие уровни не имеет смысла и будет бесполезной тратой времени и вычислительных ресурсов. Применение этого механизма на практике показало его полную оправданность — например, при решении методом FW задачи web-Google (см. ниже) с числом переменных $n = 875713$, полная высота дерева поиска минимального компонента градиента равна 20 ($2^{19} < n < 2^{20}$), а средняя “высота работы” алгоритма обновления этого дерева при наличии механизма отсечений — 5.

5.3. Генерация дискретной случайной величины с требуемой вероятностью

Для работы метода GK требуется наличие генератора дискретной случайной величины с требуемой вероятностью (алгоритм такой генерации подробно изложен в разделе 4). Реализация требуемого дискретного генератора выполнена с использованием вычислительных деревьев, описанных в разделе 5.2. Отличие в реализации состоит в том, что дерево для рассматриваемого генератора реализует функцию суммирования вероятностей и работает не с парами “индекс-значение”, а напрямую со значениями (вероятностями). Также выполнена оптимизация алгоритма генерации — достаточно всего лишь одного подбрасывания монетки для поиска нужного листа (переменной): генерируем случайную величину $0 \leq r \leq \tilde{p}$, где \tilde{p} — значение вероятности в корневом узле; начиная с корневого узла, проверяем условие $r \leq p_a$, где p_a — значение вероятности в “дочернем” узле a ; если условие выполняется — переходим на узел a , в противном случае вычисляем $r = r - p_a$ и переходим на “дочерний” узел b ; процедуру выполняем до тех пор, пока не достигнем требуемый нам лист (переменную).

5.4. Вычисление значения функции

Наличие значения функции в текущей точке для рассматриваемых методов, вообще говоря, не требуется. Но данная информация может быть полезна в качестве критерия останова, для построения графика сходимости метода и т.п. Для вычисления (обновления) значения функции можно применить подход с использованием деревьев из раздела 5.2, реализующих операцию суммирования, но это влечет за собой дополнительные вычислительные расходы ($O(\ln n)$ на каждое обновление). Поэтому в текущей реализации для методов NL1 и FW используется прямое обновление суммы:

$$f_k = \sum_{i=1}^n b_i^2, \quad b = Ax_k.$$

Рассматриваемые методы в процессе своей работы производят изменение отдельных элементов вектора x_k на некоторую величину Δx , что влечет за собой изменение s элементов вектора b :

$$\Delta b_i = A_{ij} \cdot \Delta x_j.$$

Зная эту величину мы можем обновить значение оптимизируемой функции:

$$f_k^+ = f_k + 2b_i \Delta b_i - (\Delta b_i)^2.$$

Для метода FW требуется сделать уточнение. Нетривиальным моментом, позволившем существенно увеличить скорость его работы, стало избавление от линейной (за $O(n)$) операции масштабирования вектора x_k на каждой итерации (см. раздел 3). Рассмотрим этот подход более подробно. Пусть x_1 — стартовая точка, тогда

$$\begin{aligned} x_2 &= (1 - \gamma_1)x_1 + \gamma_1 y_1 = y_1 \text{ — т.к. } \gamma_1 = 1 \\ x_3 &= (1 - \gamma_2)x_2 + \gamma_2 y_2 = (1 - \gamma_2)(x_2 + \frac{\gamma_2}{1 - \gamma_2} y_2) \\ x_4 &= (1 - \gamma_3)x_3 + \gamma_3 y_3 = (1 - \gamma_2)(1 - \gamma_3) \left(x_2 + \frac{\gamma_2}{1 - \gamma_2} y_2 + \frac{\gamma_3}{(1 - \gamma_2)(1 - \gamma_3)} y_3 \right) \\ &\dots \\ x_{k+1} &= \hat{\beta}_k \left(x_2 + \sum_{i=2}^k \hat{\gamma}_i y_i \right) = \hat{\beta}_k \left(y_1 + \sum_{i=2}^k \hat{\gamma}_i y_i \right) = \hat{\beta}_k \hat{x}_k \end{aligned}$$

где $\hat{\beta}_k = \prod_{r=2}^k (1 - \gamma_r)$, $\hat{\gamma}_k = \gamma_k / \hat{\beta}_k$

Вполне очевидно, что мы можем (разреженно) обновлять содержимое вектора \hat{x}_k указанным образом на каждой итерации. Актуальным становится вопрос такого учета коэффициента $\hat{\beta}_k$, который не нарушает разреженность и позволяет осуществлять вычисление (пересчет) функции/градиента на каждой итерации алгоритма. Считаем, что у нас уже есть вычисленное значение функции/градиента (рассматриваем задачу $\|Ax\|_2^2$) в некоторой точке, тогда

- 1) В случае изменения (путем умножения на некоторое α) коэффициента $\hat{\beta}$, значение функции пересчитывается как

$$f^+ = \alpha^2 f$$

Значение компонент градиента, соответственно, должно быть масштабировано на α , но, т.к. метод не требует собственно значений компонент, а только индекс минимального элемента, эту процедуру можно опустить. Если же требуется значение нормы градиента (например, в качестве одного из критерия остановки), то она пересчитывается очевидным образом

$$\|\alpha g\|_2 = \alpha \|g\|_2.$$

- 2) В случае изменения какой-либо компоненты вектора \hat{x} (без изменения значения $\hat{\beta}$) на некоторое Δx_j , производится обновление s элементов вектора $b = Ax$ на величину

$$\Delta b_i = A_{ij} \cdot \Delta x_j = A_{ji}^T \cdot \Delta x_j$$

для каждого такого Δb_i обновляется значение функции

$$f^+ = f + \hat{\beta}^2 (2b_i \Delta b_i - (\Delta b_i)^2)$$

и s компонент градиента g

$$\Delta g_k = A_{ki}^T \cdot \Delta b_i = A_{ik} \cdot \Delta b_i$$

Таким образом, в процессе работы метода непосредственное умножение \hat{x} на коэффициент $\hat{\beta}$ (что требует $O(n)$ операций) не производится, эта операция осуществляется только в конце, когда необходимо вернуть “истинное” значение оптимизируемых переменных $x^* = \hat{\beta}_N \cdot \hat{x}_N$. Применение описанного подхода позволило существенным образом увеличить быстродействие реализованного алгоритма.

Проведенные вычислительные эксперименты показали, что представленные схемы прямого обновления суммы (вместо применения деревьев) не ведут к каким-либо значительным накоплениям ошибок округления. После завершения процедуры минимизации для проверки производилось “честное” вычисление $f(x^*)$.

6. Численные эксперименты

Приводимые в работе результаты численных экспериментов получены на вычислительной системе, имеющей следующие характеристики:

- ubuntu server 14.04, x86_64
- Intel Core i5-2500K, 16 Гб ОЗУ
- используемый компилятор — gcc-5.2.1,
параметры сборки: `-std=c++11 -O2 -mcmmodel=small -DNDEBUG`

Поведение рассматриваемых методов исследовалось на задачах PageRank различной размерности с матрицами 3-х типов:

- 1) Диагональная, с задаваемым числом диагоналей: $n_d = 1, 3, 5, \dots$. Каждая строка/столбец матрицы содержит $(n_d - 1)/2 + 1 \leq s \leq n_d$ ненулевых элементов.
- 2) Со случайно генерируемой структурой. Каждая строка/столбец матрицы содержит ровно s ненулевых элементов.
- 3) Построенная из web-графов, загруженных с сайта Стэнфордского университета [27]. Строки/столбцы матрицы A содержат произвольное число ненулевых элементов (см. табл. 1).

Для всех проводимых экспериментов задавалось значение $\varepsilon = 10^{-4}$. Для методов NL1 и FW в качестве стартовой точки выбрана первая вершина единичного симплекса $x_0 = (1, 0, \dots, 0)$, критерий остановки — $f(x_k) = 1/2 \|Ax_k\|_2^2 \leq \varepsilon^2/2$. Метод GK останавливался по достижению требуемого числа итераций (N), после завершения алгоритма производилась проверка $\|A\bar{x}^N\|_\infty \leq \varepsilon$.

Указанное в результатах время отражает только работу метода оптимизации, т.е. не включает в себя загрузку/генерацию A/A^T , инициализацию деревьев, первоначальное “полное” вычисление функции/градиента и т.п.

Таблица 1. Характеристики матрицы A для задач с web-графами

web-graph		Число ненулевых элементов				
		в строке		в столбце		среднее
		мин.	макс.	мин.	макс.	
Stanford,	$n = 281903$	2	38607	1	256	9.2
NotreDame,	$n = 325729$	2	10722	1	3445	5.51
BerkStan,	$n = 685230$	1	84209	1	250	12.09
Google,	$n = 875713$	1	6327	1	457	6.83

Проведенные вычислительные эксперименты позволили сделать следующие выводы:

- 1) К сожалению, реализация метода GK показала неудовлетворительные результаты — авторам так и не удалось достичь стабильного выполнения условия $\|A\bar{x}^N\|_\infty \leq \varepsilon$. Поиск причин подобного поведения привел к интересным наблюдениям. Добавление вычисления функции на итерациях (что изначально не требуется и существенно снижает быстродействие) показало, что метод довольно быстро спускается в точку, которая в ряде случаев удовлетворяет требуемому условию, но после этого значение функции начинает расти и метод приходит в точку \bar{x}^N , где требуемое условие нарушено. Попытки разобраться с причиной такого поведения выявили, что в процессе оптимизации ряд компонент векторов p_i и p_j принимают очень большие значения, вплоть до машинной бесконечности в случае выполнения достаточно большого числа итераций. Очевидно, что такие значения вероятностей приводят к тому, что генератор псевдослучайных чисел просто не может “попасть” в остальные “маленькие” компоненты, которые в итоге оказываются исключены из процесса оптимизации, а метод оперирует переменными только из весьма небольшого множества. Выполнение периодического перемасштабирования векторов p_i и p_j (так, чтобы сумма их компонент стала равна 1) не изменили наблюдаемое поведение. Так же было замечено, что рассматриваемый эффект несколько ослабевает с увеличением значения N , но не исчезает полностью, а лишь сдвигается во времени — см. рис. 1.

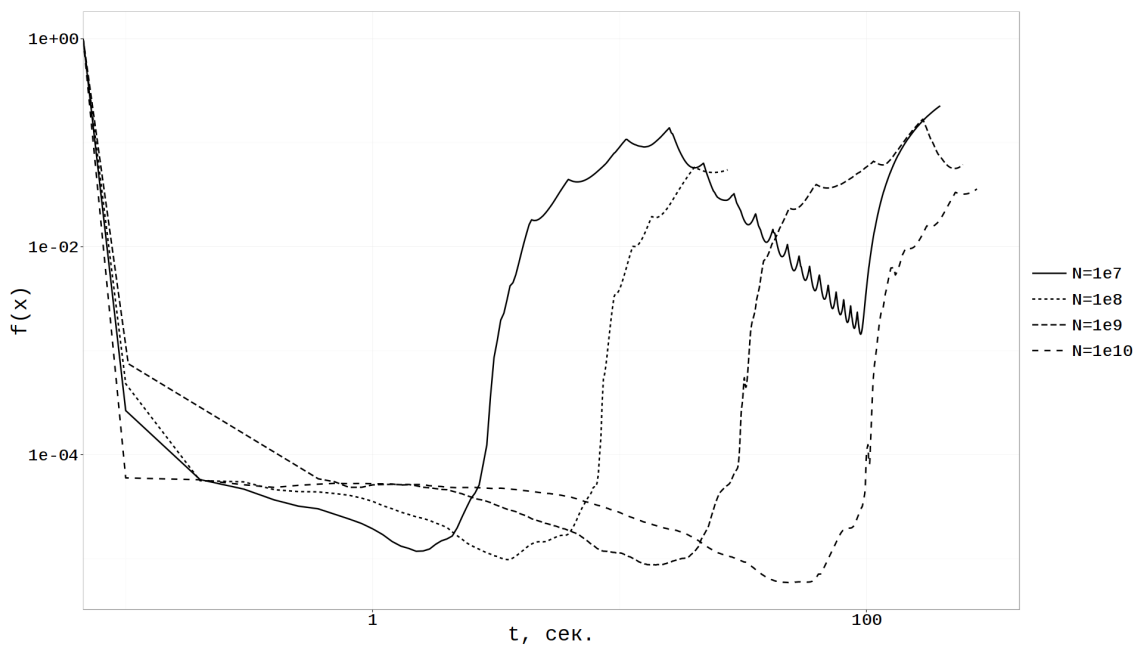


Рис. 1. Поведение метода GK при различном значении N , A — случайная, $n = 10^2$, $s = 3$

Разумным объяснением этому неприятному факту, на наш взгляд, может быть машинная арифметика, вносящая свой вклад в виде накопления погрешностей, а также “неидеальная” (с точки зрения математики) работа генератора псевдослучайных чисел (хотя в реализации используется хорошо проверенный на практике генератор MersenneTwister). Мы вынуждены констатировать, что хорошие теоретические оценки для метода GK пока дали весьма посредственные практические результаты.

- 2) Быстродействие методов на задачах с диагональной матрицей A (см. табл. 2, рис. 2) существенно выше (особенно это касается метода FW), чем в случае постановок со случайной A (см. табл. 3, рис. 3). Это связано с тем, что кэш центрального процессора при работе с диагональной матрицей работает в существенно более “комфортных” условиях, поскольку производится последовательные операции чтения/записи смежных элементов векторов x и $b = Ax$. Операции обновления деревьев поиска минимального/максимального компонент градиента так же осуществляются в смежных компонентах. Структура диагональных матриц такова, что обработав одну строку матрицы, мы получаем в кэше такой набор компонент векторов x и b , что обработка следующей строки почти не потребует обращения к ячейкам оперативной памяти, причем этот эффект практически не зависит от размерности задачи (при фиксированном n_d). Это хорошо видно на примере постановок с $s = 3$ — при росте размерности задачи на 5 порядков имеем рост времени работы алгоритма не более чем на 50%.

Таблица 2. Время (сек.) решения задачи PageRank, A — диагональная

n	метод NL1		метод FW	
	время	итерации	время	итерации
$n_d = 3; 2 \leq s \leq 3$				
10^2	4.089	3948632	0.007	14142
10^3	4.221	3950392	0.008	14142
10^4	4.575	3950392	0.009	14142
10^5	4.814	3950392	0.010	14142
10^6	5.143	3950392	0.010	14142
10^7	5.566	3950392	0.010	14142
10^8	6.021	3950392	0.010	14142
$n_d = 11; 6 \leq s \leq 11$				
10^2	14.655	2100964	0.041	14749
10^3	37.796	5101072	0.041	16956
10^4	39.170	5101072	0.062	19995
10^5	39.897	5101072	0.064	24495
10^6	41.004	5101072	0.065	24495
10^7	43.917	5101072	0.068	24495
$n_d = 51; 26 \leq s \leq 51$				
10^3	529.240	5216119	1.552	46447
10^4	535.348	5216119	1.045	29991
10^5	537.419	5216119	1.741	49235
10^6	549.782	5216119	1.758	49235
10^7	552.271	5216119	1.789	49235
$n_d = 101; 51 \leq s \leq 101$				
10^4	1935.198	5175085	6.464	49925
10^5	1962.307	5175085	9.097	68646
10^6	1940.331	5175085	9.134	68646

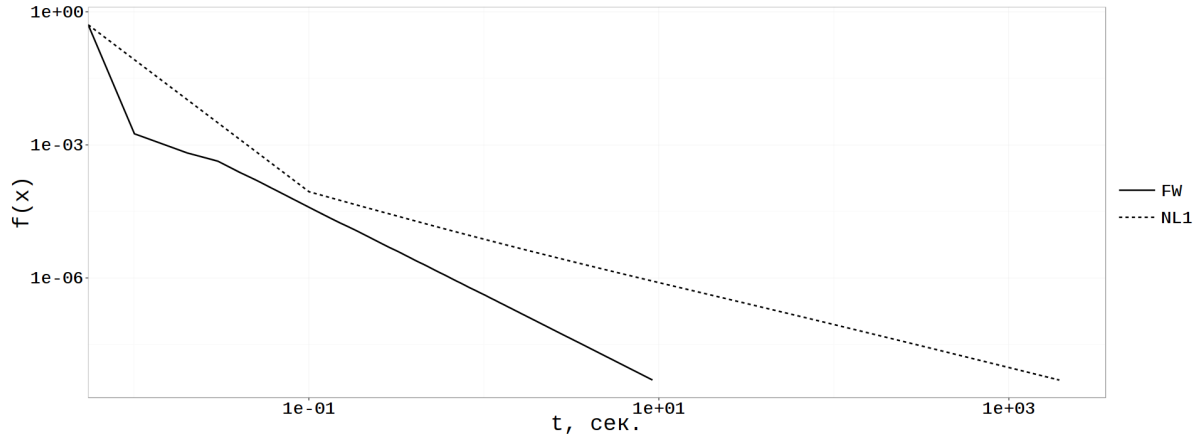


Рис. 2. Решение задачи PageRank, A — диагональная, $n = 10^6$, $n_d = 101$

Для матриц со случайной структурой начинаются “хаотичные” запросы к ячейкам памяти, что резко снижает эффективность работы кеша, которая, в общем случае, падает при увеличении n , поскольку растет размер обрабатываемых векторов x и b , которые кэшируются при таких “хаотичных” запросах всё хуже и хуже.

Таблица 3. Время (сек.) решения задачи PageRank, A — случайная

n	метод NL1		метод FW	
	время	итерации	время	итерации
$s = 3$				
10^2	0.003	1999	0.023	39734
10^3	0.031	17748	0.118	190601
10^4	0.233	141739	0.414	632954
10^5	2.374	840617	2.107	2009854
10^6	16.171	4020388	9.355	6203826
10^7	56.694	11669495	32.442	17916520
10^8	173.070	19988053	121.258	43390838
$s = 11$				
10^2	0.013	590	0.173	44706
10^3	0.072	5106	0.593	142109
10^4	0.568	40029	2.123	450873
10^5	6.342	299382	10.374	1482735
10^6	78.383	2025423	60.715	4753809
10^7	503.385	11272158	219.988	14693667
$s = 51$				
10^3	0.891	3851	11.681	162015
10^4	8.383	31372	42.824	510444
10^5	77.137	241191	164.751	1621686
10^6	1300.194	1683845	1152.805	5082774
10^7	11250.461	10627974	5432.107	17479622
$s = 101$				
10^4	29.540	29127	168.124	529685
10^5	304.419	225146	650.878	1696708
10^6	4692.729	1607834	4619.220	5267738

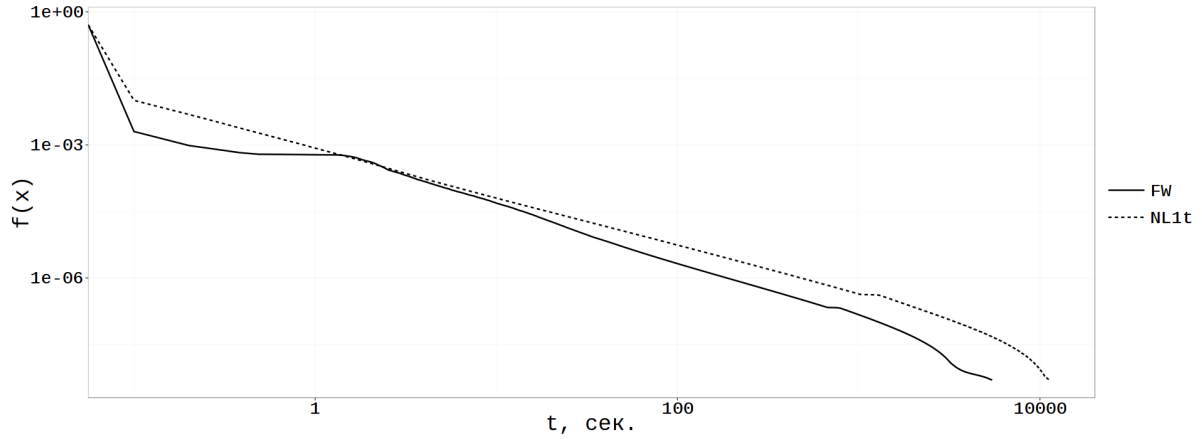


Рис. 3. Решение задачи PageRank, A — случайная, $n = 10^7$, $s = 51$

- 3) Весьма неожиданным эффектом стало быстроедействие метода FW для задач с web-графами (см. табл. 4, рис. 4 и рис. 5). Проведенные эксперименты показали, что число итераций, потребовавшихся для получения решения с требуемой точностью, для всех таких постановок оказалось *меньше*, чем размерность задачи.

Метод NL1 на 2-х задачах этого типа, к сожалению, показал весьма посредственные результаты, что, на наш взгляд связано с тем, что он модифицирует 2 переменных за итерацию (в отличие от метода FW), причем выбирает эти переменные таким образом, что регулярно попадает на “неудачные” строки/столбцы матрицы A . Поясним этот момент более подробно. В табл. 1 приведены характеристики разреженности матрицы A , где отражено, что существуют “длинные” строки/столбцы (с большим s), выбор которых на итерации влечет за собой существенный объем вычислений. Метод NL1 в процессе работы регулярным образом выбирает эти “длинные” строки/столбцы, что дает очень высокую стоимость соответствующей итерации. Обозначим s_r — число ненулевых элементов в текущей (обрабатываемой методом) строке матрицы A , s_c — в текущем столбце. Очевидно, что чем меньше эти значения, тем “легче” итерация и выше быстроедействие метода. В табл. 5 показаны величины этих параметров, которые показывают, что для задачи web-BerkStan средняя стоимость итерации (оценивается по значению $s_r \cdot s_c$, т.е. числу обрабатываемых за итерацию элементов матрицы A) для метода NL1 оказалась на порядок выше, чем у метода FW. При этом для задачи web-Stanford метод NL1 показывает намного более удачный выбор строк/столбцов, показывая результаты, вполне сопоставимые с методом FW. Заметим так же, что при решении подобных постановок с “реальной” структурой матрицы A , отдельные итерации рассматриваемых методов (максимальное значение $s_r \cdot s_c$ в табл. 5) по стоимости могут быть сопоставимы с полным вычислением функции/градиента.

Таблица 4. Время (сек.) решения задачи PageRank для web-графов

web-граф	n	метод NL1		метод FW	
		время	итерации	время	итерации
Stanford	281903	0.145	93152	0.008	14142
NotreDame	325729	700.810	3816436	0.526	38014
BerkStan	685230	38161.847	12315700	0.536	19990
Google	875713	113.643	1083996	0.278	37313

Таблица 5. Стоимость итерации для задач с web-графами

		Stanford		BerkStan	
		NL1	FW	NL1	FW
s_r	мин.	1.0	1.0	1.0	1.0
	макс.	34.0	4.0	84209.0	84209.0
	среднее	3.9	3.9	2278.4	148.6
s_c	мин.	2.0	2.0	1.0	1.0
	макс.	37.0	3.0	244.0	83.0
	среднее	2.9	2.8	15.7	6.2
$s_r \cdot s_c$	мин.	3.0	3.0	2.0	2.0
	макс.	1258.0	12.0	15494456.0	6989347.0
	среднее	11.7	11.3	84304.3	7507.5

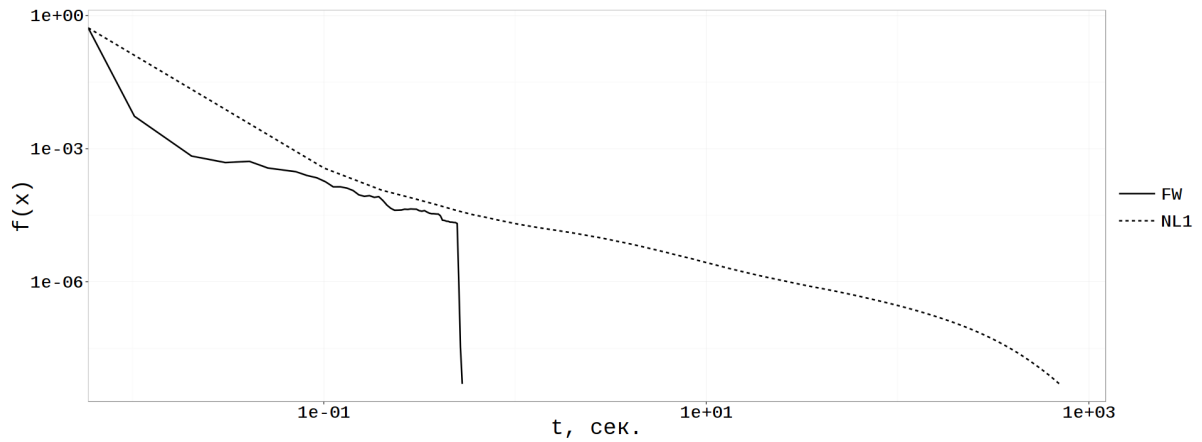


Рис. 4. Решение задачи web-NotreDame

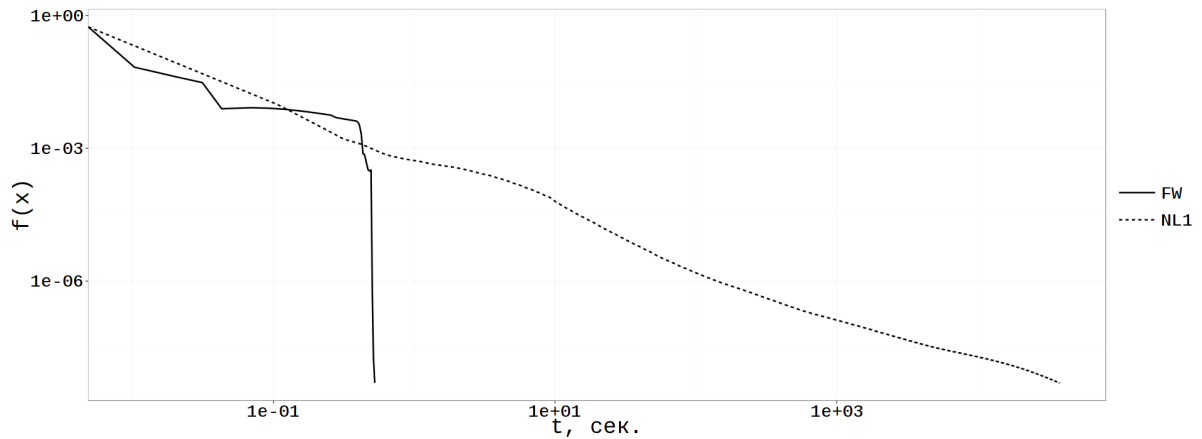


Рис. 5. Решение задачи web-BerkStan

7. Заключение

Данная статья представляет собой запись доклада А.В. Гасникова на Традиционной математической школе Б.Т. Поляка в июне 2015 года. Метод из раздела 2 был предложен Ю.Е. Нестеровым в ноябре 2014 года. Этот метод впоследствии дорабатывался А.В. Гасниковым, А.С. Аникиным, А.Ю. Горновым, Д.И. Камзоловым и Ю.В. Максимовым. Подход

из раздела 3 был предложен в апреле 2015 А.В. Гасниковым и А.Ю. Горновым. Подход дорабатывался Ю.В. Максимовым. Поход из раздела 4 был предложен А.В. Гасниковым в 2012 году (по-видимому, в это время и был введен класс дважды разреженных матриц в задачах huge-scale оптимизации, при попытке перенести результаты Григориадиса–Хачияна [19] на разреженный случай). Подход дорабатывался А.С. Аникиным. Численные эксперименты проводились А.С. Аникиным и Д.И. Камзоловым.

Исследование авторов в части 4 выполнено в ИППИ РАН за счет гранта Российского научного фонда (проект 14-50-00150), исследование в части 3 выполнено при поддержке гранта РФФИ 15-31-20571-мол_а_вед, исследование в части 2 при поддержке гранта РФФИ 14-01-00722-а.

Список литературы

- [1] *Brin S., Page L.* The anatomy of a large-scale hypertextual web search engine // Comput. Network ISDN Syst. — 1998. — V. 30(1–7). — P. 107–117.
- [2] *Langville A.N., Meyer C.D.* Google’s PageRank and beyond: The science of search engine rankings. — Princeton University Press, 2006.
- [3] *Назин А.В., Поляк Б.Т.* Рандомизированный алгоритм нахождения собственного вектора стохастической матрицы с применением к задаче PageRank // Автоматика и телемеханика. — 2011. — № 2. — С. 131–141.
- [4] *Nesterov Y.E.* Subgradient methods for huge-scale optimization problems // CORE Discussion Paper. — 2012. — V. 2012/2.
- [5] *Nesterov Y.E.* Efficiency of coordinate descent methods on large scale optimization problem // SIAM Journal on Optimization. — 2012. — V. 22. No. 2. — P. 341–362.
- [6] *Гасников А.В., Дмитриев Д.Ю.* Об эффективных рандомизированных алгоритмах поиска вектора PageRank // ЖВМиМФ. — 2015. — Т. 55. № 3. — С. 355–371.
- [7] *Hastie T., Tibshirani R., Friedman R.* The Elements of statistical learning: Data mining, Inference and Prediction. — Springer, 2009.
- [8] *Zhang Y., Roughan M., Duffield N., Greenberg A.* Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads // In ACM Sigmetrics. — 2003. San Diego, CA.
- [9] *Bubeck S.* Bubeck S. Convex optimization: algorithms and complexity // In Foundations and Trends in Machine Learning. — 2015. — V. 8. no. 3-4. P. — 231–357. arXiv:1405.4980
- [10] *Candes E.J., Wakin M.B., Boyd S.P.* Enhancing Sparsity by Reweighted l1 Minimization // J. Fourier Anal. Appl. — 2008. — V. 14. — P. 877–905.
- [11] *Allen-Zhu Z., Orecchia L.* Linear coupling: An ultimate unification of gradient and mirror descent // e-print, 2014. arXiv:1407.1537.
- [12] *Гасников А.В., Двуреченский П.Е., Нестеров Ю.Е.* Стохастические градиентные методы с неточным оракулом // arXiv:1411.4218.
- [13] *Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.* Introduction to Algorithms, Second Edition. — The MIT Press, 2001.
- [14] *Frank M., Wolfe P.* An algorithm for quadratic programming // Naval research logistics quarterly. — 1956. — V. 3. No. 1–2. — P. 95–110.

- [15] *Jaggi M.* Revisiting Frank-Wolfe: Projection-free sparse convex optimization // Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. <https://sites.google.com/site/frankwolfegreedytutorial>
- [16] *Harchaoui Z., Juditsky A., Nemirovski A.* Conditional gradient algorithms for norm-regularized smooth convex optimization // Math. Program. Ser. B. — 2015. http://www2.isye.gatech.edu/~nemirovs/ccg_revised_apr02.pdf
- [17] *Nesterov Yu.* Complexity bounds for primal-dual methods minimizing the model of objective function // CORE Discussion Papers. — 2015/03.
- [18] *Juditsky A., Nemirovski A.* First order methods for nonsmooth convex large-scale optimization, II. In: Optimization for Machine Learning. Eds. S. Sra, S. Nowozin, S. Wright. — MIT Press, 2012. <http://www2.isye.gatech.edu/~nemirovs/MLOptChapterII.pdf>
- [19] *Grigoriadis M., Khachiyan L.* A sublinear-time randomized approximation algorithm for matrix games // Oper. Res. Lett. — 1995. — V. 18. No. 2. — P. 53–58.
- [20] *Гасников А.В., Нестеров Ю.Е., Спокойный В.Г.* Об эффективности одного метода рандомизации зеркального спуска в задачах онлайн оптимизации // ЖВМ и МФ. — 2015. — Т. 55. № 4. — С. 55–71. [arXiv:1410.3118](https://arxiv.org/abs/1410.3118).
- [21] *Nesterov Yu.* Gradient methods for minimizing composite functions // Math. Prog. — 2013. — V. 140. No. 1. — P. 125–161.
- [22] *Shalev-Shwartz S., Zhang T.* Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization // Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. — P. 64–72. [arXiv:1309.2375](https://arxiv.org/abs/1309.2375).
- [23] *Zheng Q., Richtarik P., Zhang T.* Randomized dual coordinate ascent with arbitrary sampling // e-print, 2015. [arXiv:1411.5873](https://arxiv.org/abs/1411.5873).
- [24] *Fercoq O., Richtarik P.* Accelerated, Parallel and Proximal Coordinate Descent // e-print, 2013. [arXiv:1312.5799](https://arxiv.org/abs/1312.5799).
- [25] *Qu Z., Richtarik P.* Coordinate Descent with Arbitrary Sampling I: Algorithms and Complexity // e-print, 2014. [arXiv:1412.8060](https://arxiv.org/abs/1412.8060).
- [26] *Lugosi G., Cesa-Bianchi N.* Prediction, learning and games. — New York: Cambridge University Press, 2006.
- [27] Stanford Large Network Dataset Collection, Web graphs. <http://snap.stanford.edu/data/#web>.