LA-UR-18-20279
Accepted Manuscript

Efficient numerical methods to solve sparse linear equations with application to PageRank

Maximov, Yury
Anikin, Anton
Gasnikov, Alexander
Gornov, Alexander
Kamzolov, Dmitry
Nesterov, Yurii

Provided by the author(s) and the Los Alamos National Laboratory (2022-07-06).

To be published in: Optimization Methods and Software

DOI to publisher's version: 10.1080/10556788.2020.1858297

Permalink to record:
http://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-18-20279







Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy . By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Noname manuscript No.
(will be inserted by the editor)

Efficient Numerical Methods to Solve Sparse Linear Equations with Application to PageRank

Anton Anikin · Alexander Gasnikov · Alexander Gornov · Dmitry Kamzolov · Yury Maximov · Yurii Nesterov

Received: date / Accepted: date

Abstract Over the last two decades, the PageRank problem received increased interest from the academic community as an efficient tool to estimate web-page importance in information retrieval. In spite of numerous developments, the design of efficient optimization algorithms for the PageRank problem is still problem is still ongoing.In this paper, we propose three new algorithms with linear-time convergence for solving the problem over bounded-degree graphs. The idea behind the algorithms is to set up the PageRank problem as a convex minimization problem over a unit simplex and then solve it using iterative methods with small iteration complexity. Our theoretical results are supported by extensive empirical evidence using real-world and simulated data.

Keywords PageRank · Sparsity · Randomization · Frank-Wolfe algorithm · ℓ_1 optimization

Anton Anikin and Alexander Gornov
Institute of System Dynamics and Control Theory (ISDCT SB RAS), Russia

Alexander Gasnikov
Institute for Information Transmission Problems (IITP RAS), and Moscow Institute of Physics and Technology, Russia

Dmitry Kamzolov
Université Grenoble-Alpes, France, and Moscow Institute of Physics and Technology, Russia

Yury Maximov
Center for Non-linear Studies and Theoretical Division T-4, Los Alamos National Laboratory, USA, and Center for Energy Systems, Skolkovo Institute of Science and Technology
E-mail: yury@lanl.gov, corresponding author

Yurii Nesterov
Center for Operations Research and Econometrics (CORE), Catholic University of Louvain (UCL), Belgium, and the National Research University Higher School of Economics, Russia

arXiv:1508.07607v5 [math.OC] 12 Jul 2018

1 Introduction

In this paper, we address the problem of finding a solution to the system of linear equations $Ax = 0$ for sufficiently sparse matrix A, $A \in \mathbb{R}^{m \times n}$. Our research is primarily motivated by the problem of finding stationary distributions of Markov Chains over sparse large-scale communication graphs. This problem, also known as the PageRank problem, was pioneered by Brin and Page in [5,33] in the early nineties, and it is still attracting significant it still has attracted a lot of interest from both the academic community and industry. In the PageRank problem, it is assumed that the (asymmetric) transition probability matrix P, associated with the web-graph, is known and that the problem is to find the stationary distribution x over the graph with nodes corresponding to the expected time that a random walk spends at a particular node, $x = P^\top x$.

The problem becomes especially challenging in high dimensions since direct computations become unreliable due to non-linear time and memory efforts. Many studies have been devoted to the approximation of the PageRank vector based on random walks analysis and Markov Chain Monte Carlo methods [2, 13,19,20,35,37]. Those methods are very attractive, both theoretically and practically, unless the spectral gap, i.e., the difference between the two largest eigenvalues of the transition matrix, is sufficiently large. The latter is often not the case for sparse graphs with complex topologies, and, moreover, estimating the gap requires significant time and effort, as well [22].

Another line of research addresses the problem of finding the stationary distribution of a Markov chain as a convex optimization problem [14,24,29, 31]. According to these papers, the PageRank problem can be equally stated as an ℓ_p norm, $p \geq 1$, minimization problem on a unit simplex:

$$\|P^\top x - x\|_p \to \min_{x \in \Delta_1^n},$$

where $\Delta_1^n = \{x : \sum_{i=1}^n x^i = 1, x^i \geq 0\}$.

From the convex optimization perspective, similar problems appear in applied mathematics, statistics, and machine learning. Among these are LASSO [12], traffic matrix estimation in large-scale IP networks [38], phase estimation in a linearized model of electric current [36], and the finite element method [23].The high-dimensional nature of the problems above call into question the utility of traditional approaches, which do not devote sufficient attention to the problem structure and require non-linear time and memory efforts.

In this paper we focus on the influence of the transition probability matrix sparsity on the computational complexity of the PageRank problem. We advocate particular efficiency of convex optimization methods for the simplex constrained $\|Ax\|_2^2$ and $\|Ax\|_\infty$ minimization problems [6,7]. In particular, we prove that the time complexity of these problems is linear in the problem dimension if the number of non-zeros in each row/column is bounded above by some constant d. The key contribution is a set of efficient algorithms to update a function value, a gradient, and an argument in an (almost)

dimension-independent manner. Later in the paper, we extend our results to a more general setup, where a limited number of dense rows or columns in the transition matrix is allowed.

1.1 Contribution

Our contribution can be summarized as follows. We propose:

1. The `NL1` algorithm, a ℓ_1-proximal gradient descent method that supports sparse updates of the gradient and the function value. It allows us to solve the problem with overall time complexity $O(n + d^2 \log^2(n/\varepsilon)/\varepsilon^2)$, where d is the maximal in- and out- vertexes' degree of the graph, ε is required accuracy, and $\|Px - x\|_2 \leq \varepsilon$;
2. The `S-FW` method, an extension of the Frank-Wolfe algorithm that allows efficient gradient and function value update. The resulting algorithm `S-FW` has much better time-complexity estimates than the `NL1` method, $O(n + d^2 \log(2+n/d^2)/\varepsilon^2)$, but the algorithms often have comparable performance in practice;
3. The `GK` algorithm, which aims to minimize $\|P^\top x - x\|_\infty$ over the unit simplex. We provide an equivalent saddle-point setup for this problem and, subsequently, solve it subsequently by a version of the mirror descent with a randomized projecting. We prove that a randomized projection with KL-divergence guarantees the total running time of the algorithm to be bounded from above by $O(n + d \log n \log(n/\delta)/\varepsilon^2)$ with probability at least $1 - \delta$, for any $0 < \delta < 1$;
4. And we extend the linear-time complexity estimates to the sparse graphs with a small number of dense rows/columns containing more than d non-zeros.

Let us emphasize that a d-sparse matrix could have as many non-zero elements as $n \cdot d$. Time-complexity estimates of the proposed algorithms are yet sub-linear in the number of non-zero elements of the transition matrix if its sparsity pattern is known beforehand. The last statement implies that there is no need to read all non-zero elements of a stochastic matrix to arrive at approximate solution of the PageRank problem. In Table 1.1 we summarize the best known results for solving the PageRank problem using optimization techniques.

The Google page condition required in [31] is the existence of a column j such that all $P_{ij} \geq \alpha$, $\alpha > 0$ for any $i : 1 \leq i \leq n$; and β corresponds to the spectral gap of the matrix P, e.g. $\beta = \lambda_1(P) - \lambda_2(P) = 1 - \lambda_2(P)$, the difference between the largest and the second-largest eigenvalues of the transition matrix P. The estimate [27] runs on average for the randomized coordinate descent algorithm while the estimates [13, 24, 25] are correct with probability at least $1 - \delta$, for any $\delta : 0 < \delta < 1$.

Algorithm	Constraint	Time Complexity	Objective
Nazin - Polyak [24]	no	$O\left(\frac{n\log(n/\delta)}{\varepsilon^2}\right)$	$\|P^\top x - x\|_2 \le \varepsilon$
Nesterov [27]	d-sparse	$O\left(n + \frac{d^2\log n}{\varepsilon^2}\right)$	$\mathbb{E}\|P^\top x - x\|_2 \le \varepsilon$
Nesterov [29]	d-sparse	$O\left(dn\frac{\log n}{\varepsilon^2}\right)$ $O\left(\frac{d^{1/2}n^{3/2}\log n}{\varepsilon}\right)$	$\|P^\top x - x\|_\infty \le \varepsilon$
Juditsky et al. [18,25]	no	$O\left(\frac{n\log(n/\delta)}{\varepsilon^2}\right)$	$\|P^\top x - x\|_\infty \le \varepsilon$
Nesterov - Nemirovski [31]	d-sparse Google page	$O\left(\frac{nd}{\alpha}\log 1/\varepsilon\right)$	$\|x - x^*\|_1 \le \varepsilon$
Polyak - Tremba [34]	d-sparse on average	$O\left(\frac{dn}{\varepsilon}\right)$	$\|P^\top x - x\|_1 \le \varepsilon$
Gasnikov - Dmitriev, [13]	d-sparse spectral gap β	$O\left(n + \frac{d\log n\log(n/\delta)}{\beta\varepsilon^2}\right)$	$\mathbb{E}\|x - x^*\|_2 \le \varepsilon$
Gasnikov - Dmitriev, [13]	m non-zeros	$O\left(m + \left(n + \frac{m^2}{n^2}\right)\frac{\log n}{\varepsilon^2}\right)$	$\|x - x^*\|_\infty \le \varepsilon$
Langville - Meyer, [21]	d-sparse spectral gap β	$O\left(\frac{dn}{\beta}\log\frac{n}{\varepsilon}\right)$	$\|x - x^*\|_1 \le \varepsilon$
Cohen et al. [9]	m non-zeros	$O(m+n)\log^{O(1)}\frac{n}{\beta\varepsilon}$	$\mathbb{E}\|P^\top x - x\|_2 \le \varepsilon$
This paper, Section 2	d-sparse	$O\left(n + \frac{d^2\log(n/d^2+1)}{\varepsilon^2}\right)$	$\|P^\top x - x\|_2 \le \varepsilon$
This paper, Section 3	d-sparse	$O\left(n + \frac{d^2\log(n/d^2+2)}{\varepsilon^2}\right)$	$\|P^\top x - x\|_2 \le \varepsilon$
This paper, Section 4	d-sparse	$O\left(n + \frac{d\log n\log(n/\delta)}{\varepsilon^2}\right)$	$\|P^\top x - x\|_\infty \le \varepsilon$

Table 1 Time complexity of the PageRank problem. Time complexity of the algorithms proposed in this paper along with results of Nesterov [27] are the only sub-linear algorithms known to the authors.

1.2 Paper structure

In Section 2 we introduce the ℓ_1 proximal gradient descent algorithm. The key idea behind this time-efficient algorithm, referenced below as `NL1`, are sparse gradient and function value updates, $f(x) = \|Ax\|_2^2$, $A = P^\top - I$

$$\nabla f(x+h) = \nabla f(x) + A^\top Ah, \quad f(x+h) = f(x) + h^\top \nabla f(x) + \|Ah\|_2^2.$$

`NL1` does not require full gradient computations on each step if matrix A is sufficiently sparse. We show that the update vector h on each step, has only two non-zero coordinates, which specify the minimal and the maximal coordinates of the gradient. To update gradient coordinates and extract the minimal and the maximal value, we use a list of binary heaps [10], which admits logarithmic dependence of iteration complexity in dimension.

Algorithm `S-FW`, a revision of the Frank-Wolfe conditional gradient, is proposed in Section 3. The Frank-Wolfe algorithm has recently stimulated much interest, mainly due to the numerous Big Data problems to which it has been applied [16,17,30]. In this paper, we focus on efficient gradient and function value updates per iteration in order to reduce the time complexity of the algorithm with the same number of iterations. Accurate theoretical analysis provides us with a similar time complexity estimate as for `NL1`. Also, the `S-FW` and `NL1` algorithms appear to be comparable from a practical perspective.

In Section 4, we deal with the problem of the approximation of the PageRank vector in the ℓ_∞ norm, i.e., the minimization of $\|P^\top x - x\|_\infty$. Using the technique of Juditsky et al. [18], the problem could be equally stated as a saddle-point problem over the product of two unit simplexes. Randomized mirror descent seems to be one of the most popular tools for solving the problem efficiently. Its application to the PageRank problem leads to the time-complexity estimate $O(n \log(n/\delta)/\varepsilon^2)$ and does not depend on the problem sparsity [25].

In this paper we focus on the approach pioneered by Grigoriadis and Khachiyan in [15]. The idea behind the approach is the randomized projection of the gradient on the simplex rather than randomized approximation of the gradient itself. Projection of the gradient on the unit simplex is carried out in accordance with the Kullback-Leibler divergence, corresponding to the exponential weighting of the gradient coordinates. Randomized projection chooses one of the vertices of the unit simplex in such a way that the expected value gives an unbiased estimation of the projection [13].

We provide an algorithm to update the gradient vector on each iteration in an (almost) dimension-independent manner, which leads to a linear time-complexity estimate

$$O(n + d \log n \log(n/\sigma)/\varepsilon^2)$$

for d-sparse transition matrices, with $\|P^\top x - x\|_\infty \le \varepsilon$. Finally, we pay a special attention to its interpretation in terms of game theory.

In Sections 2 - 4, we also summarize sparsification techniques allowing to reduce ranking problems over dense graphs to sparse problems. The proposed algorithms deliver state-of-the-art time-complexity estimates for d-sparse optimization problems if d is sufficiently small. Faster estimates are possible via the Markov Chain Monte Carlo method for some problem setups. We refer to recent results [13] for details.

We conclude in Section 5 with the implementation details and a case study. Finally, some technical proofs are given in the appendix.

In this paper, we use the following notation. By $\|x\|_p$, we denote the ℓ_p norm of vector $x \in \mathbb{R}^n$, in particular, $\|x\|_2 = x^\top x$ denotes the Euclidean norm, $\|x\|_1 = \sum_{i=1}^n |x_i|$, and $\|x\|_\infty = \max\{|x_1|, \dots, |x_n|\}$. By $O(1)$, we denote positive absolute constants.

2 ℓ_1-Gradient descent for d-sparse problems

The PageRank problem can be equally stated as the following convex optimization problem

$$f_\gamma(x) = \frac{1}{2}\|Ax\|_2^2 + \sum_{i=1}^{n} \frac{\gamma}{2}(-x^i)_+^2 \to \min_{x^\top e=1}, \tag{1}$$

where $A = I - P^\top$, I is the $n \times n$ identity matrix, $P \in \mathbb{R}_+^{n\times n}$ is a column stochastic transition matrix, e is an n-dimensional vector of all ones, $f_\gamma(x)$ is convex and increasing function with $f_\gamma(x_*) = 0$, and $z_+ = \max\{z, 0\}$.

To solve the problem, we use the proximal gradient descent with an ℓ_1 setup:

$$x_{k+1} = x_k + \operatorname*{argmin}_{h:h^\top e=0} \left\{ f_\gamma(x_k) + h^\top \nabla f(x_k) + \frac{L_1}{2}\|h\|_1^2 \right\}, \tag{2}$$

where $f_\gamma(x)$ is L_1-smooth in ℓ_1 norm

$$\|\nabla f_\gamma(x) - \nabla f_\gamma(y)\|_\infty \le L_1\|x - y\|_1, \quad L_1 = 1 + \gamma.$$

Let us emphasize that vector h_k which solves the problem 2 is always sparse whatever the function $f_\gamma(x)$ is. In particular, it has only two non-zero coordinates corresponding to:

$$i_+ = \operatorname*{argmax}_{1\le i\le n} \partial f_\gamma(x_k)/\partial x^i \quad \text{and} \quad i_- = \operatorname*{argmin}_{1\le i\le n} \partial f_\gamma(x_k)/\partial x^i$$

so that

$$h_k^{i_+} = \frac{1}{4L_1}\left(\frac{\partial f_\gamma(x_k)}{\partial x^{i_+}} - \frac{\partial f_\gamma(x_k)}{\partial x^{i_-}}\right) \text{ and } h_k^{i_-} = -\frac{1}{4L_1}\left(\frac{\partial f_\gamma(x_k)}{\partial x^{i_+}} - \frac{\partial f_\gamma(x_k)}{\partial x^{i_-}}\right) \tag{3}$$

That is why the gradient $\nabla f_\gamma(x_{k+1})$ is

$$\begin{aligned}\nabla f_\gamma(x_{k+1}) =& \nabla f(x_k + h_k) = \\ & A^\top A x_k + A^\top A h_k + \gamma \sum_{i=i_+,i_-} (-x_k^i + h_k^i)_+^2 - (-x_k^i)_+^2\end{aligned} \tag{4}$$

The update vector has at most $O(\min(n, d^2))$ non-zero coordinates. To efficiently update the gradient, we will use a doubly-linked list of binary heaps, so that the j-th heap is used to extract the minimal value and update coordinates from $(j-1)\lfloor n/d^2\rfloor + 1$ to $\min\{j\lfloor n/d^2\rfloor, n\}$ inclusively. We refer to a binary heap as a *Max-Heap*, if the key stored in each node is greater than or equal to the keys in the node's children. We require both the minimal and the maximal coordinates of the gradient for an iteration of the algorithm. Using *Min-Heap* (*Max-Heap*) extracting the minimal (maximal) element from a heap of m items requires $O(\log m)$ time. An update of a single element to preserve

Algorithm 1: NL1: ℓ_1 Gradient Descent for PageRank

Input: d-sparse transition matrix P, starting point x_0
objective $f_\gamma(x) = \frac{1}{2}\|Ax\|_2^2 + \frac{\gamma}{2}\sum_{i=1}^n (-x_i)_+^2$ with $A = I - P^\top$, and $\gamma > 0$
number of iterations k required by Theorem 1, and accuracy ε.
Output: x_k

while $\kappa \le k$ *and* $f_\gamma(x) > \varepsilon^2$ **do**
  $i_+ = \operatorname{argmax}_{1\le i\le n} \partial f_\gamma(x_k)/\partial x^i, \quad i_- = \operatorname{argmin}_{1\le i\le n} \partial f_\gamma(x_k)/\partial x^i$
  $$h_\kappa^i = \begin{cases} \frac{1}{4L_1}\left(\frac{\partial f_\gamma(x_k)}{\partial x^{i_+}} - \frac{\partial f_\gamma(x_k)}{\partial x^{i_-}}\right), & i = i_+ \\ -\frac{1}{4L_1}\left(\frac{\partial f_\gamma(x_k)}{\partial x^{i_+}} - \frac{\partial f_\gamma(x_k)}{\partial x^{i_-}}\right), & i = i_- \\ 0, & \text{otherwise} \end{cases}$$
  Update argument: $x_{\kappa+1} = x_\kappa + h$
  Update gradient: $\nabla f_\gamma(x_{\kappa+1}) = \nabla f_\gamma(x_\kappa) + A^\top A h_\kappa + \gamma(-x_k + h_k)_+ - \gamma(-x_k)_+$
  Update $f_\gamma(x_{\kappa+1})$:
  $f_\gamma(x_{\kappa+1}) = f_\gamma(x_\kappa) + h^\top A^\top A x + \|Ah\|_2^2/2 + \frac{\gamma}{2}\sum_{i:\, h^i\ne 0}(-x^i)_+^2 - (-x_\kappa^i - h_\kappa^i)_+^2$
  $\kappa = \kappa + 1$
return x_κ

the keys' order requires $O(\log m)$ time, as well [10]. That is why a gradient update 4 requires $O(d^2 \log(2 + n/d^2))$ time in total.

Algorithm 1 presents the NL1 algorithm with convergence rate established in Theorem 1.

Theorem 1 *For a starting point x_0 in one of the vertices of the unit simplex, Algorithm 1 converges to $f_\gamma(x) \le \varepsilon^2$ for any constant $\gamma > 0$, with the overall time complexity*

$$T = O\left(n + d^2\log(n/d^2+2)\left(\frac{1}{\varepsilon^2} + \min_{\delta: n\delta^2>1}\left[\frac{1}{\delta^2\log(n\delta^2)}\right]\right)\right).$$

The bound established in Theorem 1 gives $T = O\left(nd^2\log(n/d^2+2)/\varepsilon^2\right)$, for the case $n\varepsilon^2 = o(1)$, and sub-linear time complexity

$$T = \left(n + d^2\log(n/d^2+2)/\varepsilon^2\right)$$

for the most practical case, $n\varepsilon^2 = \Omega(1)$.

Matrix Sparsification. In a number of practical problems, both column and row sparsity of the transition probability matrix seems to be very restrictive. Indeed, for the PageRank problem, search engines such as Google or Yahoo! may refer to a large number of sites simultaneously. In particular, that means that matrix $P^\top$ may have a few dense columns. Below we propose a method to sparsify the transition probability matrix P in order to improve the convergence of optimization methods without loss of quality.

Consider a single linear equation $a^\top x = b$, $x \in \mathbb{R}^n$, where a is a dense vector. The system can be equally stated as

$$a_i^\top x = bz_i, \quad 1 \le i \le \lceil n/d\rceil, \quad \sum_{i=1}^{\lceil n/d\rceil} a_i = a, \quad \sum_{i=1}^{\lceil n/d\rceil} z_i = 1,$$

such that each a_i has no more than d non-zero elements. In order to guarantee that $\|a^\top x - b\|_2^2 \le \varepsilon^2$, one needs to find a pair (x, z) such that

$$\sum_{i=1}^{\lceil n/d \rceil} \lceil n/d \rceil \|a_i^\top x - z_i\|_2^2 \le \varepsilon^2.$$

The last corresponds to a system of linear equations $\tilde{A}(x\,z) = 0$ of the size $(n + d \cdot \lceil n/d \rceil) \times (n + \lceil n/d \rceil)$ with no more than $d+1$ non-zero elements in each row. In order to solve the problem, we apply the `NL1` algorithm, although it requires a minor correction:

$$f_\gamma(x, z) = \frac{1}{2}\|\tilde{A}(x\,z)\|_2^2 + \frac{\gamma}{2}\sum_{i=1}^{n}(-x_i)_+^2 \to \min_{\substack{x^\top e = 1,\, x \ge 0 \\ z^\top e = 1}}$$

A step of the `NL1` algorithm is

$$\begin{pmatrix} x \\ z \end{pmatrix}_{k+1} = \begin{pmatrix} x \\ z \end{pmatrix}_k + \operatorname*{argmin}_{\substack{h_x :\, h_x^\top e = 0 \\ h_z :\, h_z^\top e = 0}} \left\{ f_\gamma(x_k, z_k) + \nabla f_\gamma(x_k, z_k) \cdot \begin{pmatrix} x \\ z \end{pmatrix}_k + \frac{L}{2}\|(h_x\,h_z)\|_1^2 \right\}$$

Recall that the optimal $h = (h_x\,h_z)$ has only two non-zero coordinates, as was true for the `NL1` algorithm over the simplex, corresponding to

$$i_x^+ = \operatorname*{argmax}_{1 \le i \le n} \frac{\partial f(x_k, z_k)}{x^i} \quad \text{and} \quad i_x^- = \operatorname*{argmin}_{1 \le i \le n} \frac{\partial f(x_k, z_k)}{x^i},$$

if

$$\max_{1 \le i \le n} \frac{\partial f_\gamma(x_k, z_k)}{x^i} - \min_{1 \le i \le n} \frac{\partial f_\gamma(x_k, z_k)}{x^i} > \max_{1 \le j \le \lceil \frac{n}{d} \rceil} \frac{\partial f_\gamma(x_k, z_k)}{z^j} - \min_{1 \le j \le \lceil \frac{n}{d} \rceil} \frac{\partial f_\gamma(x_k, z_k)}{z^j},$$

while

$$i_z^+ = \operatorname*{argmax}_{1 \le j \le \lceil \frac{n}{d} \rceil} \frac{\partial f_\gamma(x_k, z_k)}{z^j} \quad \text{and} \quad i_z^- = \operatorname*{argmin}_{1 \le j \le \lceil \frac{n}{d} \rceil} \frac{\partial f_\gamma(x_k, z_k)}{z^j},$$

otherwise. Now, for efficient implementation of the `NL1` algorithm, one needs to store the values $\partial f_\gamma / \partial z^j$ and $\partial f_\gamma / \partial x^i$ in separate binary heaps. A similar strategy can be used if sparsification of multiple rows is required. Finally, we remind the reader that single-row sparsification increases the problem dimension by n, at most.

Column sparsification is slightly more involved. Consider a function $f(x)$ in more details:

$$\begin{aligned} f_\gamma(x) &= \frac{1}{2}\|Ax\|_2^2 + \frac{\gamma}{2}\sum_{i=1}^{n}(-x)_+^2, \qquad x = (x_1, x_2, \dots x_n) \\ \bar{f}_\gamma(x_1) &= \min_{x_{2:n}} \left\{ f(x) + I_{\{x \in \Delta_1^n\}} \right\}. \end{aligned}$$

The function $\bar{f}_\gamma(x_1)$ is convex in x_1 for any fixed $\gamma > 0$ [4, Section 3.2.5]. That is why the value of $\bar{f}_\gamma(x_1)$ can be computed using the `NL1` algorithm while the value

$$f^*_\gamma = \min_{0 \le x_1 \le 1} f_\gamma(x_1)$$

requires a one dimensional binary search so that the overall time complexity T of the `NL1` algorithm used to solve the PageRank problem with a single dense column is

$$T = O\left(n + d^2 \log\left(\frac{n}{d^2} + 2\right) \log \frac{n}{\varepsilon} \min\left\{\frac{n}{\varepsilon^2}, \min_{\delta : n\delta^2 > 1} \left[\frac{\log \frac{1}{\varepsilon}}{\delta^2 \log(n\delta^2)}\right]\right\}\right).$$

The same approach can be applied to improving the efficiency of the `S-FW` and `GK` algorithms proposed later in Sections 3 and 4 on the PageRank instances with a few dense rows or columns.

3 Frank-Wolfe algorithm with sparse updates

The PageRank problem, according to Eq. 1, can be stated as

$$f(x) = \frac{1}{2}\|Ax\|_2^2 \to \min_{x \in \Delta_1^n}$$

We use the Frank-Wolfe conditional gradient [11,17] to solve the problem above.

We choose the starting point x_0 of the algorithm in one of the vertices of the unit simplex. Then on each step we solve

$$h_k^\top \nabla f(x_k) \to \min_{y \in \Delta_1^n} \tag{5}$$

Furthermore the solution y_k of Eq. 5 has only one non-zero coordinate $y_k^{i_k}$, corresponding to

$$i_k = \operatorname*{argmin}_{1 \le i \le k} \partial f(x)/\partial x^i.$$

Then, the update rule is

$$x_{k+1} = (1 - \gamma_k)x_k + \gamma_k h_k, \quad \gamma_k = \frac{2}{k+1}, \quad k \ge 1.$$

According to [28], $f(x_k) - f(x)$ is bounded from above as:

$$f(x_k) - f^* = f(x_k) \le \frac{2L_1 \max_{x,y \in \Delta_1^n} \|x - y\|_1^2}{k+1} \le \frac{8L_1}{k+1}, \tag{6}$$

where $L_1^2 = \max_{x \in \Delta_1^n} \|Ax\|_2^2 \le 2$. Thus, in order to guarantee $f(x_k) \le \varepsilon^2/2$, one needs at most $32\varepsilon^{-2}$ iterations.

Algorithm 2: S-FW: Frank-Wolfe algorithm for PageRank

Input: d-sparse transition matrix P, starting point x_0 in one of the vertices of unit simplex, and $A = P^\top - I$
Output: $x_k : f(x_k) = \|Px_k - x_k\|_2/2 \le \varepsilon$

while $f(z_k) > \beta_k^2 \varepsilon^2$ **do**
  $i_k = \operatorname{argmin}_{1 \le i \le n} \partial f(x_k)/\partial x^i, \quad \tilde{\gamma}_k = 2/(k+1)$
  $h_k = \tilde{\gamma}_k \cdot \delta$, where $\delta^{i_k} = 1$, and $\delta^i = 0$, $i \ne i_k$
  Update argument: $z_{k+1} = z_k + \tilde{\gamma}_k h_k$
  Update gradient: $\frac{\nabla f(x_{k+1})}{\beta_k} = \frac{\nabla f(x_k)}{\beta_k} + \tilde{\gamma}_k A^\top A h_k$
  Update function value $f(x_{k+1})$: $\frac{f(x_{k+1})}{\beta_k^2} = \frac{f(x_k)}{\beta_k^2} + \tilde{\gamma}_k h_k^\top \frac{\nabla f(x)}{\beta_k} + \|Ah_k\|_2^2 \tilde{\gamma}_k^2/2$
  $k = k + 1$
return $x_k = z_k \beta_k^2$

Consider a step of the algorithm in more detail. Denote β_k as

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

The the step of the Frank-Wolfe algorithm is

$$z_{k+1} = z_k + \tilde{\gamma}_k h_k,$$

where h_k is a solution of Eq. 5. Moreover, only one coordinate of h_k:

$$i_k = \operatorname*{argmin}_{1 \le i \le k} \partial f(x_k)/\partial x^i = \operatorname*{argmin}_{1 \le i \le k} A^\top A z_k^i.$$

is other than zero. Therefore, the update of the S-FW algorithm is similar to the Gauss-Southwell rule studied in detail in [32] for minimization of the strongly convex functions.

Using the doubly-linked list of binary heaps, described in Section 2, the minimal coordinate of $A^\top A z_k$ can be computed in $O(d^2 \log(2 + n/d^2))$ time. Then $A^\top A z_{k+1}$ is

$$A^\top A z_{k+1} = A^\top A z_k + \tilde{\gamma}_k A^\top A y_k. \tag{7}$$

and

$$\frac{\nabla f(x_{k+1})}{\beta_k} = \frac{\nabla f(x_k)}{\beta_k} + \frac{\tilde{\gamma}_k \beta_k A^\top A h_k}{\beta_k}$$

and, also

$$\frac{f(x_{k+1})}{\beta_k^2} = \frac{f(x_k)}{\beta_k^2} + \tilde{\gamma}_k h_k^\top \frac{\nabla f(x)}{\beta_k} + \|Ah_k\|_2^2 \tilde{\gamma}_k^2/2$$

For a d-sparse matrix A, the time required to compute $A^\top A z_{k+1}$ from $A^\top A z_k$ is $O(d^2 \log(2 + n/d^2))$, as well. One can compute x_k having z_k as $x_k = \beta_k x_k$ in $O(n)$ time.

Combining 6 and 7, we have the following complexity estimate for the algorithm.

Theorem 2 *Algorithm 2 requires at most $k = 32\varepsilon^{-2}$ iterations to guarantee $\|P^\top x - x\|_2 \leq \varepsilon$ for any d-sparse transition matrix P. The overall time complexity of the algorithm does not exceed*

$$T = O\left(n + \frac{d^2 \log(2 + n/d^2)}{\varepsilon^2}\right).$$

Discussion. Theorem 2 implies sub-linear convergence in the number of non-zero elements of the transition matrix P. This is not surprising, since we assume that the underlying graph structure, along with required smoothness, are known a priori.

It remains an open question for the authors to improve convergence rate in terms of the accuracy and maximal degree of the transition graph while preserving linear dependence in the problem dimension.

4 Saddle point setup for PageRank

In the PageRank problem, a user often requires accurate approximation of a few of the largest coordinates representing the most relevant websites rather than the full PageRank vector. To this end, we propose an algorithm to approximate the PageRank vector in ℓ_∞-norm. Below, we consider the problem

$$f(x) = \|(P^\top - I)x\|_\infty = \|Ax\|_\infty \to \min_{x \in \Delta_1^n}.$$

Following [25], we set up the problem as

$$\min_{x \in \Delta_1^n} \max_{\|y\|_1 \leq 1} \langle Ax, y\rangle = \min_{x \in \Delta_1^n} \max_{y \in \Delta_1^{2n}} \langle Ax, Jy\rangle = \min_{x \in \Delta_1^n} \max_{y \in \Delta_1^{2n}} \langle x, \tilde{A}y\rangle, \tag{8}$$

where $\tilde{A} = A^\top J$, $J = [I_n, -I_n]$, and I_n is the $n \times n$ identity matrix.

We propose a sub-linear-time algorithm to approximate a bilinear matrix game representing the PageRank, Problem 8. Let $\tilde{A}_{ij}$ be a gain for Player A (loss of player B), if A plays strategy i and B plays strategy j, $1 \leq i \leq n$, and $1 \leq j \leq 2n$. Consider the loss function for Player B at step k:

$$f(x, y_k) = x^\top \tilde{A} y_k, \quad x \in \Delta_1^n,$$

where $y_k \in \Delta_1^{2n}$ is a vector with a single non-zero coordinate corresponding to the strategy of Player A. We also emphasize that y_k depends on the whole history of the game. Let C be the cost of the matrix game:

$$C = \max_{y \in \Delta_1^{2n}} \min_{x \in \Delta_1^n} y^\top \tilde{A} x = \min_{x \in \Delta_1^n} \max_{y \in \Delta_1^{2n}} y^\top \tilde{A} x = \min_{x \in \Delta_1^n} \|Ax\|_\infty = 0.$$

and

$$\min_{x \in \Delta_1^n} \frac{1}{N} \sum_{i=1}^N f(x, y_i) \geq C \geq \max_{y \in \Delta_1^{2n}} \frac{1}{N} \sum_{i=1}^N f(x_i, y), \tag{9}$$

Algorithm 3: UPDATE RULE FOR A PROBABILITY DISTRIBUTION

Input: (unnormalized) probability distribution p given by a binary tree with leafs values p_i such that the probability of any leaf j is $p_j / \sum_{i=1}^{n} p_i$, $p_i > 0$ for any $i:\ 1 \le i \le n$, $\sum_{i=1}^{n} p_i > 0$, with and update rule of coordinate k according to Eq. 10

$$\tilde{p}_k \propto p_k \exp\left(-\psi_k\right),$$

Output: $x \sim \tilde{p}$, a sample x follows the updated distribution $\tilde{p}$

1 // Distribution update
2 $u = k$ // Start with the leaf k
3 **while** $u \neq$ *root* **do**
4 $\quad p_u \leftarrow p_u + p_k(\exp(-\psi_k) - 1)$;
5 $\quad u \leftarrow$ parent of u
6 // Sampling $x \sim \tilde{p}$
7 $u = k$ // Start with the leaf k
8 **while** $u \neq$ *leaf* **do**
9 $\quad$ Let ν, ω be children of u
10 $\quad$

$$u = \begin{cases} \nu, & \text{with probability } p_\nu/(p_\nu + p_\omega), \\ \omega, & \text{otherwise.} \end{cases}$$

11 **return** u

for any sequences $\{x_i\}_{i=1}^N$, $\{y_i\}_{i=1}^N$ if for any i: $x_i, \in \Delta_1^n$, $y_i \in \Delta_1^{2n}$, $1 \le i \le n$. In the subsequent of the section, we consider $\{(x_i, y_i)\}_{i=1}^N$ with a single non-zero coordinate each.

In order to solve the problem, assume the following randomized strategy for the Player B played against any strategy of the Player A:

1. Let $p_1 = (n^{-1}, \ldots, n^{-1})$;
2. Choose at random j_k, such that $\mathbb{P}(j_k = j) = p_j^k$;
3. Assume $x_{j_k}^k = 1$ and $x_j^k = 0$ for all $j \neq j_k$;
4. Update

$$p_j^{k+1} \propto p_j^k \exp\left(-\gamma \tilde{A}_{i_k, j}\right), \tag{10}$$

where i_k is a strategy that Player A chooses at step k.

The crucial gain in the efficiency of the algorithm is due to a time-efficient updates at stage 3. Indeed, consider a binary tree with its leaves corresponding to the variables, and constructed in such a way that the value p_v assigned to a node v is a total probability of all leaves having v as a predecessor. If we update the weight of a leaf according to Eq. 10, we also update each vertex u belonging to the path from the leaf to the root as

$$p_u = p_u + \xi, \quad \xi = p_j^{k+1} - p_j^k.$$

In order to sample $x \sim p^{k+1}$, we start from the root of the tree and proceed to its child a with probability $p_a/(p_a + p_b)$. Otherwise, we proceed to its sibling

Algorithm 4: Grigiriadis-Khachyan Randomization for ℓ_∞ approximation of the PageRank problem

Input: d-sparse transition matrix P, starting point x_0 in one of the vertices of the unit simplex, and learning rate γ

Output: x_k : $f(\bar{x}_k, \bar{y}_k) = \bar{x}_k \tilde{A} \bar{y}_k \leq \varepsilon^2$, implies $\|P^\top x_k - x_k\|_\infty \leq \varepsilon$

$\pi = (n^{-1}, \ldots, n^{-1}), \quad p = (n^{-1}, \ldots, n^{-1})$,

starting point (x_0, y_0) in one of the vertices of $\Delta_1^n \times \Delta_1^{2n}$

while $f(\bar{x}_k, \bar{y}_k) > \varepsilon^2$ **do**

  // Player A turn

  Choose at random i_k^A, such that $\mathbb{P}(i_k = j) = \pi_i$;

  Assume $y_k^{i_k^A} = 1$, and $y_k^i = 0$ if $i \neq i_k^A$;

  Update π_k// see Algorithm 3 for details

$$\pi_{k+1}^i \propto \pi_k^i \exp\left(-\gamma \tilde{A}_{i, j_k^B}\right)$$

  // Player B turn

  Choose at random j_k^B, such that $\mathbb{P}(j_k^B = j) = p_j$;

  Assume $x_k^{j_k^B} = 1$, and $x_k^j = 0$ if $j \neq j_k^B$;

  Update p_k // see Algorithm 3 for details

$$p_{k+1}^j \propto p_k^j \exp\left(-\gamma \tilde{A}_{i_k^A, j}\right)$$

  $k\bar{x}_k = \sum_{t \leq k} x_t$, $k\bar{y}_k = \sum_{t \leq k} y_t$

  $k^2 f(\bar{x}_k, \bar{y}_k) = (k\bar{x}_k)^\top \tilde{A}(k\bar{y}_k) =$
  $(k-1)^2 f(\bar{x}_{k-1}, \bar{y}_{k-1}) + \bar{x}_{k-1}\tilde{A}y_k + x_k \tilde{A}\bar{y}_k + x_k \tilde{A} y_k$

return $(\bar{x}_k, \bar{y}_k)$

b, where p_a, and p_b are the values assigned to a and b, respectively. We repeat the same procedure for each node one a path from the root to one of the leafs of the tree. Algorithm 3 formalizes this argument.

Using the same strategy for the Player A, we establish the convergence rate to the Nash equilibrium (x^*, ω^*), which solves problem 8 in Theorem 3. Algorithm 4 contains all necessary details. It is worth mentioning an interpretation of the algorithm: indeed on each iteration it make an update following to a sparse projection of the gradient to the unit simplex according to KL divergence.

Theorem 3 *Algorithm 4 after $N \geq 16\varepsilon^{-2}(\log 2n + 8\log(2/\delta))$ iterations with a constant step-size $\gamma = \sqrt{2(\log n)/N}$ results in a point $(\bar{x}_N, \bar{y}_N)$ such that, with probability at least $1 - \delta$, for any $\delta > 0$ one has:*

$$\|A\bar{x}_N\|_\infty \leq \bar{y}_N^\top \tilde{A} \bar{x}_N \leq \varepsilon.$$

Moreover, the total running time of the algorithm is bounded from above as

$$T = O\left(n + \frac{d \log n \log \frac{n}{\delta}}{\varepsilon^2}\right).$$

The proof of the theorem is provided in Appendix B.

Discussion. The mirror descent and the dual averaging perspective. The proposed algorithm is essentially a mirror descent with randomized projection of the gradient on a unit simplex according to KL divergence. Let us consider the problem of minimizing of the left hand-side of Eq. 9 in more details:

$$\frac{1}{N}\sum_{i=1}^{N} f(x, y_i) = \frac{1}{N}\sum_{i=1}^{N} \langle x, A^\top J y_i \rangle \to \min_{x \in \Delta_1^n} \tag{11}$$

where $\{y_i\}_{i=1}^n$ is a sequence of unit coordinate vectors. Denote $f_i(x) \doteq f(x, y_i)$ for $i \geq 1$. Recall the setup of the mirror descent algorithm [25]. Let $\omega(x) = \sum_{i=1}^n x^i \log x^i$ be the distance-generating function, which is is 1-strongly convex with respect to the ℓ_1 norm. A step of the dual averaging algorithm [26] with step-size γ is:

$$y_{k+1} = y_k - \gamma \nabla f_k(x_k), \quad x_{k+1} = \nabla \omega^*(y_{k+1}), \tag{12}$$

where $\omega^*(y) = \sup_{x \in \Delta_1^n} \left\{ y^\top x - \omega(x) \right\} = \log \left\{ \sum_{j=1}^n \exp(y_j) \right\}$. An update of x_{k+1} in Eq. 12 can be also viewed as a projection of y_{k+1} to the unit simplex in accordance with Kullback-Leibler divergence. Indeed Eq. 12 is a step of mirror descent algorithm for simplex constrained problems as well [1, Appendix A], [3], [18]. A randomized version of the update is then

$$x_{k+1} = e_i, \text{ with probability } p_{k+1}^i = \frac{p_k^i \exp\left(-\gamma \frac{\partial f_k(x_k)}{\partial x^i}\right)}{\sum_{t=1}^n p_k^t \exp\left(-\gamma \frac{\partial f_k(x_k)}{\partial x^t}\right)}, \tag{13}$$

where e_i is a unit vector with a single non-zero coordinate corresponding to index i. Since $f(x, y) = y^\top \tilde{A} x$, update 13 is the same as the update in Algorithm 4.

5 Implementation details and case study

All algorithms proposed in the paper are implemented in `C++`. We test our code with `GCC` (GNU Compiler Collection, v.: 4.8.4, 4.9.3, 5.2.1), `clang` (C language family front-end for LLVM, v.: 3.4.2, 3.5.2, 3.6.1, 3.7.0), and `icc` (Intel C Compiler, v. 15.0.3) under GNU/Linux, Microsoft Windows, and Mac OS X. We conduct the experiments using:

- Ubuntu server 14.04, x86_64
- Intel Core i5-2500K, 16 Gb RAM
- `GCC`-5.2.1 to compile `C++` code,
- Assembly parameters: `-std=c++11 -O2 -mcmodel=small -DNDEBUG`

We test our algorithms in different dimensions using the following three test beds:

1. d-diagonal matrix for $n_d = 1, 3, 5, \ldots$. Each row/column of these matrices contains $(n_d - 1)/2 + 1 \leq d \leq n_d$ non-zero elements;

2. randomly-generated matrices with d non-zero elements (on average);
3. and web-graphs from the Stanford University graph collection [1]

We use accuracy $\varepsilon = 10^{-4}$ in each of our experiments; $x_0 = (1, 0, \ldots, 0)$ is used as a starting point for the `NL1` and `S-FW` algorithms, and we terminate the algorithms if $f(x_k) = \|Ax\|_2^2/2 \leq \varepsilon^2/2$. Computational time reported for the case study includes time required by optimization method, $A/A^\top$ generating time, initialization of all data structures used, as well as initial gradient/function computation.

web-graph		# non zeros				
		in a row		in a column		average
		min	max	min	max	
Stanford,	$n = 281\,903$	2	38 607	1	256	9.20
NotreDame,	$n = 325\,729$	2	10 722	1	3 445	5.51
BerkStan,	$n = 685\,230$	1	84 209	1	250	12.09
Google,	$n = 875\,713$	1	6 327	1	457	6.83

Table 2 Structure of the matrix A for the graphs from the Stanford web-graph collection. Columns of the transition matrix are more dense than the rows and on average both columns and rows contain a few non-zero elements only.

The numerical experiments described below allow the following conclusions to be drawn:

1. Implementation of the `GK` algorithm leads to unfavorable results, since the authors have not guaranteed $\|A\bar{x}\|_\infty \leq \varepsilon$ for sufficiently small ε. In our experiments (see Figure 1) the algorithm converges sufficiently fast for the desired precision. But after a large number of iterations, the value of $f(\bar{x})$ starts to grow, and the resulting point does not satisfy the accuracy condition. The reason for this is that the values of several probabilities p_i become extremely large and out of range after a number of iterations. That leads to significant numerical errors in estimating residual small probabilities and overall unsatisfactory performance of the algorithm. Rescaling the probability vectors does not change the behavior of the algorithm. The described effect decreases for larger n; refer to 1 for details. To summarize, the theoretical bounds for the `GK` algorithm differ markedly from from those seen in practical performance.
2. Computational time for d-diagonal matrices A is much smaller than that for random matrices (see Figures 2 and 3 for details).This is due to the fast cache operations, which require far fewer memory reads for sequential data. Updates to computational trees/heaps are also performed in sequential elements, which improves time performance as well. Also, this permits the dramatic reduction of the dependence on the actual problem dimension in practice. Thus in Table 2 for $d = 3$ and accuracy $\varepsilon = 10^{-4}$ the

[1] `http://snap.stanford.edu/data/#web`

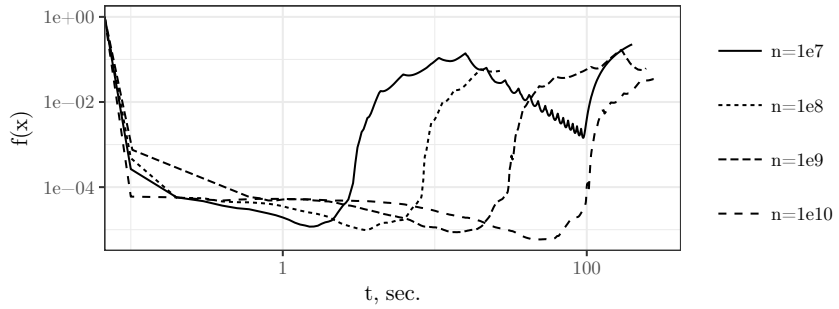


Fig. 1 Convergence of the `GK` algorithm for various dimensions n. A is a random $n \times n$ matrix, with a number of non-zeros in each row and column $d = 3$. Practical performance of the `GK` algorithm is limited due to unavoidable errors in estimating small probabilities.

computational time has increased just twice for $n = 10^{10}$ compared with $n = 10^7$.

	`NL1`		`S-FW`	
n	time, sec.	iteration	time	iterations
$n_d = 3$; $2 \le d \le 3$				
10^2	4.089	3 948 632	0.007	14 142
10^3	4.221	3 950 392	0.008	14 142
10^4	4.575	3 950 392	0.009	14 142
10^5	4.814	3 950 392	0.010	14 142
10^6	5.143	3 950 392	0.010	14 142
10^7	5.566	3 950 392	0.010	14 142
10^8	6.021	3 950 392	0.010	14 142
$n_d = 11$; $6 \le d \le 11$				
10^2	14.655	2 100 964	0.041	14 749
10^3	37.796	5 101 072	0.041	16 956
10^4	39.170	5 101 072	0.062	19 995
10^5	39.897	5 101 072	0.064	24 495
10^6	41.004	5 101 072	0.065	24 495
10^7	43.917	5 101 072	0.068	24 495
$n_d = 51$; $26 \le d \le 51$				
10^3	529.240	5 216 119	1.552	46 447
10^4	535.348	5 216 119	1.045	29 991
10^5	537.419	5 216 119	1.741	49 235
10^6	549.782	5 216 119	1.758	49 235
10^7	552.271	5 216 119	1.789	49 235
$n_d = 101$; $51 \le d \le 101$				
10^4	1935.198	5 175 085	6.464	49 925
10^5	1962.307	5 175 085	9.097	68 646
10^6	1940.331	5 175 085	9.134	68 646

Table 3 Time in seconds required to solve the PageRank problem. A is a d-diagonal matrix. The `S-FW` algorithm outperforms the `NL1` algorithm for most of the instances and have better scalability with the dimension of the problem.

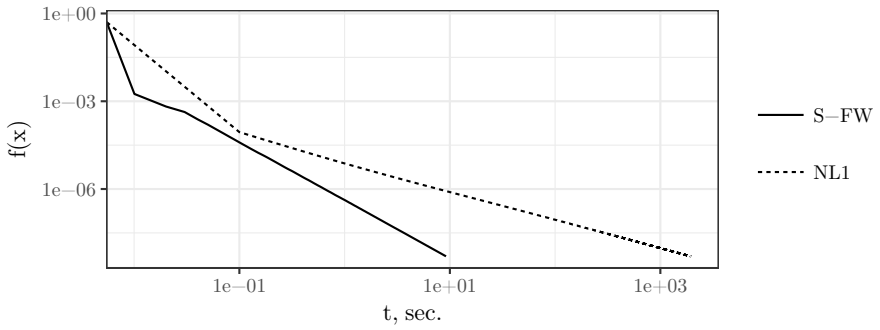


Fig. 2 Computational time for the PageRank problem, A is a n_d-diagonal matrix, dimension $n = 10^6$, number of non-zero diagonals $n_d = 101$. The `S-FW` algorithm significantly outperforms the `NL1` algorithms for large scale problems.

Conversely, for the random matrices, caching does not give the same improvement in speed. This significantly decreases the actual performance of the algorithms; see Table 2 for details.

	`NL1`		`S-FW`	
n	time	iterations	time	iterations
$d = 3$				
10^2	0.003	1 999	0.023	39 734
10^3	0.031	17 748	0.118	190 601
10^4	0.233	141 739	0.414	632 954
10^5	2.374	840 617	2.107	2 009 854
10^6	16.171	4 020 388	9.355	6 203 826
10^7	56.694	11 669 495	32.442	17 916 520
10^8	173.070	19 988 053	121.258	43 390 838
$d = 11$				
10^2	0.013	590	0.173	44 706
10^3	0.072	5 106	0.593	142 109
10^4	0.568	40 029	2.123	450 873
10^5	6.342	299 382	10.374	1 482 735
10^6	78.383	2 025 423	60.715	4 753 809
10^7	503.385	11 272 158	219.988	14 693 667
$d = 51$				
10^3	0.891	3 851	11.681	162 015
10^4	8.383	31 372	42.824	510 444
10^5	77.137	241 191	164.751	1 621 686
10^6	1 300.194	1 683 845	1 152.805	5 082 774
10^7	11 250.461	10 627 974	5 432.107	17 479 622
$d = 101$				
10^4	29.540	29 127	168.124	529 685
10^5	304.419	225 146	650.878	1 696 708
10^6	4 692.729	1 607 834	4 619.220	5 267 738

Table 4 Time in seconds required to solve the PageRank problem. A is a random matrix. The `NL1` algorithm outperforms the `S-FW` algorithm in most of the sparse and low-dimensional instances, while the `S-FW` algorithm is preferable for large scale cases.

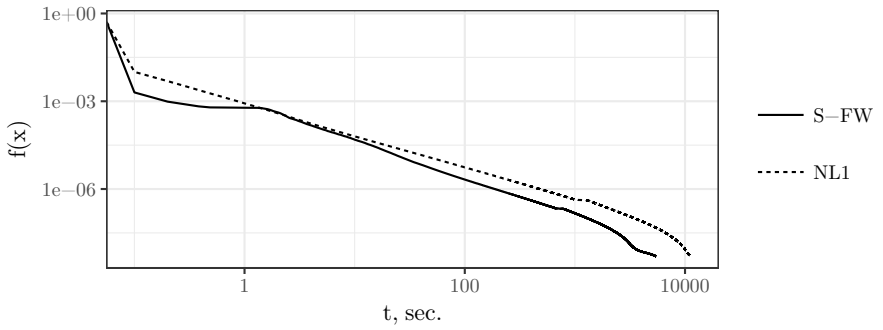


Fig. 3 Time complexity for the PageRank problem. A is a random matrix, dimension $n = 10^7$, average number of non-zeros in each row and column $d = 51$. The `NL1` and `S-FW` algorithms have almost the same computational time.

3. Surprisingly, the time complexity of the `S-FW` algorithm for the Stanford web-graph collection is much less than that for the `NL1` algorithm (see Table 3). Unfortunately, for two problems on the list, the `NL1` algorithm performance is not sufficiently high. We propose that this is due to the fact that the `NL1` algorithm modifies two variables per iteration and often involves very expensive, dense updates compared with to the `S-FW` algorithm. See Table 5 for the information about the sparsity of the transition matrices. Table 3 contains information about the average iteration complexity of the `NL1` algorithm; it is much higher than that for the `S-FW` algorithm, which supports our conjecture, particularly for the web-BerkStan dataset. Recall that this property is true without additional matrix sparsification (refer to Section 2). The time complexity of the `S-FW` and `NL1` algorithms are shown in Figures 4 and 5.

		`NL1`		`FW`	
web-graph	n	time	iterations	time	iteration
Stanford	281 903	0.145	93 152	0.008	14 142
NotreDame	325 729	700.810	3 816 436	0.526	38 014
BerkStan	685 230	38 161.847	12 315 700	0.536	19 990
Google	875 713	113.643	1 083 996	0.278	37 313

Table 5 Time in seconds required to solve the PageRank problem for web-graphs from the Stanford graph collection. The `S-FW` algorithm achieves significantly better time performance compared to the `NL1` algorithm.

6 Conclusion

In this paper, we have proposed three novel algorithms to solve the PageRank problem. All the algorithms can be viewed as guided versions of coordinate or

		Stanford		BerkStan	
		NL1	FW	NL1	FW
d_r	min	1.0	1.0	1.0	1.0
	max	34.0	4.0	84 209.0	84 209.0
	average	3.9	3.9	2 278.4	148.6
d_c	min	2.0	2.0	1.0	1.0
	max	37.0	3.0	244.0	83.0
	average	2.9	2.8	15.7	6.2
$d_r \cdot d_c$	min	3.0	3.0	2.0	2.0
	max	1 258.0	12.0	15 494 456.0	6 989 347.0
	average	11.7	11.3	84 304.3	7 507.5

Table 6 Iteration complexity for the Stanford graph collection. The `S-FW` algorithm has performed a much fewer number of updates, resulting in a higher performance compared to the `NL1` algorithm.

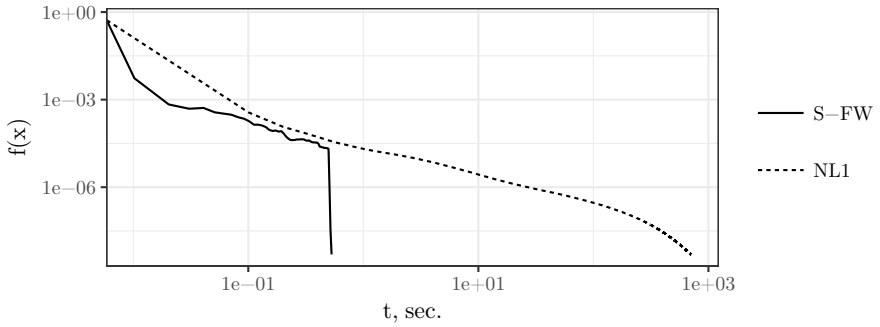


Fig. 4 Time complexity for PageRank over the web-NotreDame dataset. Time complexity for PageRank over the web-BerkStan dataset. The `FW` algorithm significantly outperforms the `NL1` algorithm and has a significant gain in the vicinity of the optimal point.

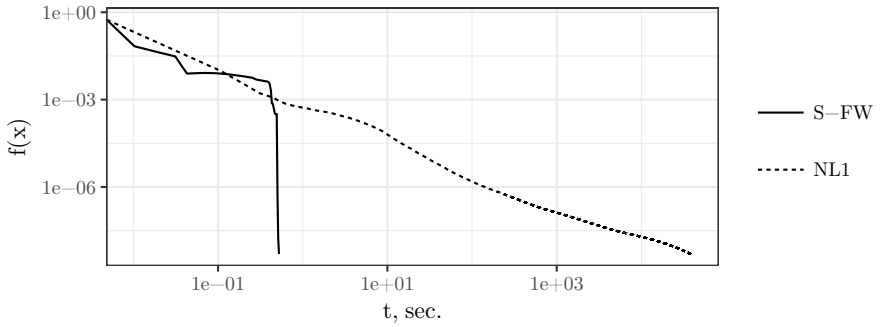


Fig. 5 Time complexity for PageRank over the web-BerkStan dataset. The `FW` algorithm significantly outperforms the `NL1` algorithm and has a significant gain in the vicinity of the optimal point.

block-coordinate descent and demonstrate superior practical performance. In

further works, the authors intend to devote more attention to sparsification techniques and interplay between the problem sparsity, dimension, and desired accuracy.

References

1. Allen-Zhu, Z., Orecchia, L.: Linear coupling: An ultimate unification of gradient and mirror descent. In: Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS 2017), p. 3:13:22 (2017)
2. Avrachenkov, K., Litvak, N., Nemirovsky, D., Osipova, N.: Monte carlo methods in pagerank computation: When one iteration is sufficient. SIAM Journal on Numerical Analysis **45**(2), 890–904 (2007)
3. Banerjee, A., Merugu, S., Dhillon, I.S., Ghosh, J.: Clustering with bregman divergences. Journal of machine learning research **6**(Oct), 1705–1749 (2005)
4. Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
5. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer networks **56**(18), 3825–3833 (2012)
6. Bubeck, S., et al.: Convex optimization: Algorithms and complexity. Foundations and Trends® in Machine Learning **8**(3-4), 231–357 (2015)
7. Candes, E.J., Wakin, M.B., Boyd, S.P.: Enhancing sparsity by reweighted 1 minimization. Journal of Fourier analysis and applications **14**(5), 877–905 (2008)
8. Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge university press (2006)
9. Cohen, M.B., Kelner, J., Peebles, J., Peng, R., Rao, A.B., Sidford, A., Vladu, A.: Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, pp. 410–419. ACM (2017)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, second edition (2001)
11. Frank, M., Wolfe, P.: An algorithm for quadratic programming. Naval Research Logistics (NRL) **3**(1-2), 95–110 (1956)
12. Friedman, J., Hastie, T., Tibshirani, R.: The elements of statistical learning, vol. 1. Springer series in statistics. New York (2001)
13. Gasnikov, A.V., Dmitriev, D.Y.: On efficient randomized algorithms for finding the pagerank vector. Computational Mathematics and Mathematical Physics **55**(3), 349 (2015)
14. Gasnikov, A.V., Nesterov, Y.E., Spokoiny, V.G.: On the efficiency of a randomized mirror descent algorithm in online optimization problems. Computational Mathematics and Mathematical Physics **55**(4), 580–596 (2015)
15. Grigoriadis, M.D., Khachiyan, L.G.: A sublinear-time randomized approximation algorithm for matrix games. Operations Research Letters **18**(2), 53–58 (1995)
16. Harchaoui, Z., Juditsky, A., Nemirovski, A.: Conditional gradient algorithms for norm-regularized smooth convex optimization. Mathematical Programming **152**(1-2), 75–112 (2015)
17. Jaggi, M.: Revisiting frank-wolfe: projection-free sparse convex optimization. In: Proceedings of the 30th International Conference on International Conference on Machine Learning-Volume 28, pp. I–427. JMLR. org (2013)
18. Juditsky, A., Nemirovski, A.: First order methods for nonsmooth convex large-scale optimization, i: general purpose methods. Optimization for Machine Learning pp. 121–148 (2011)

19. Kamvar, S., Haveliwala, T., Golub, G.: Adaptive methods for the computation of pagerank. Linear Algebra and its Applications **386**, 51–65 (2004)
20. Kamvar, S.D., Haveliwala, T.H., Manning, C.D., Golub, G.H.: Extrapolation methods for accelerating pagerank computations. In: Proceedings of the 12th international conference on World Wide Web, pp. 261–270. ACM (2003)
21. Langville, A.N., Meyer, C.D.: Google's PageRank and beyond: The science of search engine rankings. Princeton University Press (2011)
22. Levin, D.A., Peres, Y.: Markov chains and mixing times, vol. 107. American Mathematical Soc. (2017)
23. Logan, D.L.: A first course in the finite element method. Cengage Learning (2011)
24. Nazin, A.V., Polyak, B.T.: Randomized algorithm to determine the eigenvector of a stochastic matrix with application to the pagerank problem. Automation and Remote Control **72**(2), 342–352 (2011)
25. Nemirovski, A., Juditsky, A., Lan, G., Shapiro, A.: Robust stochastic approximation approach to stochastic programming. SIAM Journal on optimization **19**(4), 1574–1609 (2009)
26. Nesterov, Y.: Primal-dual subgradient methods for convex problems. Mathematical programming **120**(1), 221–259 (2009)
27. Nesterov, Y.: Efficiency of coordinate descent methods on huge-scale optimization problems. SIAM Journal on Optimization **22**(2), 341–362 (2012)
28. Nesterov, Y.: Introductory lectures on convex optimization: A basic course, vol. 87. Springer Science & Business Media (2013)
29. Nesterov, Y.: Subgradient methods for huge-scale optimization problems. Mathematical Programming **146**(1-2), 275–297 (2014)
30. Nesterov, Y.: Complexity bounds for primal-dual methods minimizing the model of objective function. Mathematical Programming pp. 1–20 (2015)
31. Nesterov, Y., Nemirovski, A.: Finding the stationary states of markov chains by iterative methods. Applied Mathematics and Computation **255**, 58–65 (2015)
32. Nutini, J., Schmidt, M., Laradji, I., Friedlander, M., Koepke, H.: Coordinate descent converges faster with the gauss-southwell rule than random selection. In: International Conference on Machine Learning, pp. 1632–1641 (2015)
33. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
34. Polyak, B.T., Tremba, A.A.: Regularization-based solution of the pagerank problem for large matrices. Automation and Remote Control **73**(11), 1877–1894 (2012)
35. Sarma, A.D., Molla, A.R., Pandurangan, G., Upfal, E.: Fast distributed pagerank computation. Theoretical Computer Science **561**, 113–121 (2015)
36. Stott, B., Jardim, J., Alsaç, O.: Dc power flow revisited. IEEE Transactions on Power Systems **24**(3), 1290–1300 (2009)
37. Tong, H., Faloutsos, C., Pan, J.Y.: Fast random walk with restart and its applications (2006)
38. Zhang, Y., Roughan, M., Duffield, N., Greenberg, A.: Fast accurate computation of large-scale ip traffic matrices from link loads. In: ACM SIGMETRICS Performance Evaluation Review, vol. 31, pp. 206–217. ACM (2003)

A Missing proof of section 2

Recall the definition of $f_\gamma(x)$:

$$f_\gamma(x) = \frac{1}{2}\|Ax\|_2^2 + \sum_{i=1}^{n} \frac{\gamma}{2}\left(-x^i\right)_+^2.$$

Lemma 1 *Let x_* satisfies $f_\gamma(x_*) \leq \varepsilon^2$ for some $\gamma > 0$. Then for $\hat{x} = (x_*)_+ / e^\top (x_*)_+$ we have*

$$\|A\hat{x}\|_2^2 \leq 4(1+\gamma^{-1})\varepsilon^2,$$

and $\hat{x} \in \Delta_1^n = \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1,\ x_i \geq 0\}$

Proof By the conditions of the lemma

$$\frac{1}{2}\|Ax_*\|_2^2 + \sum_{i=1}^{n} \frac{\gamma}{2}\left(-x_*^i\right)_+^2 \leq \varepsilon^2. \tag{14}$$

Let $x = (x_*)_+ - (-x_*)_+$, then by the triangle inequality we have

$$\|(A(x_*)_+)\|_2 \leq \|Ax_*\|_2 + \|(A(-x_*)_+)\|_2 \leq \sqrt{2}\varepsilon + \|(A(-x_*)_+)\|_2 \tag{15}$$

Since $\lambda_{\max}(A) = \lambda_{\max}(I - P^\top) = 1$ we have by Inequality 14

$$\|A(-x_*)_+\|_2 \leq \|(-x_*)_+\|_2 \leq \varepsilon\sqrt{\frac{2}{\gamma}} \tag{16}$$

Using Inequalities 14, 15 and 16 we have the final estimate

$$\|A(x_*)_+\|_2 \leq \sqrt{2}\varepsilon + \|A(-x_*)_+\|_2 \leq \sqrt{2}\varepsilon + \sqrt{\frac{2}{\gamma}}\varepsilon.$$

By definition of $\hat{x}$, we have $\|A\hat{x}\|_2((x_*)_+^\top e) = \|A(x_*)_+\|_2$. Since $e^\top((x_*)_+ - (-x_*)_+) = 1$ and $(-x_*)_+^\top e \geq 0$ we have $(x_*)_+^\top e \geq 1$ and $\|A\hat{x}\|_2 \leq \|A(x_*)_+\|_2$. An application of the Cauchy-Schwartz inequality concludes the proof of the lemma.

Proposition B.1 of [1] establishes convergence rate of the gradient descent in arbitrary norm.

Proposition 1 (Proposition B.1 of [1]) *Let $f(x)$ be a convex, differentiable function that is L-smooth with respect to $\|\cdot\|$ on $Q = \mathbb{R}^n$, and x_0 any initial point in Q. Consider the sequence of T gradient steps $x_{k+1} = \operatorname{argmin}_{y \in Q}\left\{\frac{L}{2}\|y - x\|^2 + \nabla f(x_k)^\top (y - x)\right\}$, then the last point x_T satisfies*

$$f(x_T) - f(x_*) \leq 2\frac{LR^2}{T}$$

where $R = \max_{x:\, f(x) \leq f(x_0)} \|x - x_*\|$, and x_* is any minimizer of f.

Theorem 1 *For a starting point x_0 in one of the vertices of the unit simplex, Algorithm 1 converges to $f_\gamma(x) \leq \varepsilon^2$ for any constant $\gamma > 0$, with the overall time complexity*

$$T = O\left(n + d^2\log(n/d^2 + 2)\left(\frac{1}{\varepsilon^2} + \min_{\delta : n\delta^2 > 1}\left[\frac{1}{\delta^2\log(n\delta^2)}\right]\right)\right).$$

Proof First, we establish the upper bound of the form:

$$T = O\left(n + \frac{nd^2 \log(n/d^2 + 2)}{\varepsilon^2}\right). \tag{17}$$

A single iteration of Algorithm 1 results in a sparse update vector h containing at most two non-zero coordinates. Then the gradient update

$$\nabla f_\gamma(x_k + h) = \nabla f_\gamma(x_k) + \eta A^\top Ah + \gamma(-x_k + h)_+ - \gamma(-x_k)_+$$

requires $O(d^2 \log(n/d^2 + 2))$ operations by using a set of $\lceil n/d^2 \rceil$ binary heaps described in the Section 2. The function value update

$$f_\gamma(x_k + \eta h) = f_\gamma(x_k) + 2h^\top Ax_k + \|Ah\|_2^2/2 + \frac{\gamma}{2}(-x + h)_+^2 - \frac{\gamma}{2}(-x)_+^2$$

similarly requires at most $O(d)$ operations. The size of the level set

$$R = \max_{x:\, f(x) \le f(x_0)} \|x - x_*\|_1$$

at a point x is bounded from above as $R \le 2\sqrt{2nf_\gamma(x_0)/\gamma} + 2$ because

$$\frac{\gamma}{2}\frac{\|(-x)_+\|_1^2}{n} \le \frac{\gamma}{2}\|(-x)_+\|_2^2 = \frac{\gamma}{2}\sum_{i=1}^n (-x^i)_+^2 \le \frac{1}{2}\|Ax\|_2^2 + \frac{\gamma}{2}\sum_{i=1}^n (-x^i)_+^2 \le f_\gamma(x_0) \tag{18}$$

and

$$R \le \max_{x: f_\gamma(x) \le f_\gamma(x_0)} \{\|x\|_1 + \|x_*\|_1\} \le 2\|(-x)_+\|_1 + 2 \le 2\sqrt{\frac{2nf_\gamma(x_0)}{\gamma}} + 2, \tag{19}$$

where $\|x\|_1 \le \|(-x)_+\|_1 + \|(x)_+\|_1 \le 2\|(-x)_+\|_1 + 1$ since $\sum_{i=1}^n x^i = 1$.

Fix any $\delta : n\delta^2 < 1$, and let $\varepsilon_1^2 = \delta^2 n$. In order to achieve $f_\gamma(x_{(1)}) \le \varepsilon_1^2$ one needs

$$T_{(1)} = 16\frac{(1+\gamma)(1 + 2nf_\gamma(x_0)/\gamma)}{\delta^2 n} \le 16\frac{(1+\gamma)(1 + 2f_\gamma(x_0)/\gamma)}{\delta^2}$$

iterations. Let $\varepsilon_k^2 = \delta^2 n f_\gamma(x_{(k-1)}) = \delta^{2k} n^k$ for any $k \ge 1$. If $2nf_\gamma(x_{(k)})/\gamma \ge 1$ we have:

$$T_{(k)} \le 16\frac{(1+\gamma)(1 + 2nf_\gamma(x_{(k)}))}{\delta^{2k+2} n^{k+1}} \le 64\frac{(1+\gamma)n^{k+1}\delta^{2k}}{n^{k+1}\delta^{2k+2}} = \frac{64(1+\gamma)}{\delta^2}$$

Then $T_{(j)} = O(1/\delta^2)$, $1 \le j \le k$, and $2nf_\gamma(x_{(k)}) \le 1$ in $k = O(\log(f_\gamma(x_0))/\log(n\delta^2))$ steps. Thus the overall number of iterations for this step is $\sum_{i=1}^k T_{(i)} = O(1/\delta^2 \log(n\delta^2))$. The remaining time required to solve the problem starting with $x_{(k)}$ is bounded from above as $32(1+\gamma)/\varepsilon^2$ which completes the proof of the theorem.

B Missing proof of section 4

Theorem 3 provides us with an upper bound on the efficiency of this strategy. Here we assume that the number of iterations N is known in advance. Let us emphasize that Algorithm 4 is a version of the Mirror Descent algorithm with randomized projecting aims to support sparse updates (see Section 4 for details).

Our main tool below is Proposition 2, which establishes the convergence rate of stochastic online optimization for linear functions f_k linear in x. We refer to recent results in [14] for a more general problem setup.

In the proof of the following proposition we mostly follow [8] and [14]. Our proof strongly relies to the recent results for dual averaging method [26] leads to the same sequence of steps as the mirror descent for simplex constrained convex optimization problems [1].

Proposition 2 *Let $\{f_k(x) = x^\top A y_k\}_{k=1}^N$ be a set of functions $f : \mathbb{R}^n \to \mathbb{R}$ of variable x such that $\|\nabla f_k(x)\|_\infty \le M$ almost surely. Then for constant step-size policy $\gamma = M\sqrt{2(\log n)/N}$ we have*

$$\psi_N \doteq \mathbb{E}_{x_1,\ldots,x_N}\left[\sum_{k=1}^N f_k(x_k)\right] - \min_{x\in\Delta_1^n} f_k(x) \le M\sqrt{2N\log n}.$$

Moreover, for any $\Omega > 0$

$$Prob\left[\psi_N > M\sqrt{2N\log n} + 2M\sqrt{2N\Omega}\right] \le \exp(-\Omega).$$

Proof Let $y_{k,t} = y_k t + (1-t)y_{k-1}$, and let $\mathbb{E}_{k|k-1}(\cdot)$ be a conditional expectation with respect to the randomized projection $x_{k+1} = \nabla\omega^*(y_{k+1})$ only. Then

$$\begin{aligned}
\mathbb{E}_{k|k-1}\,\omega^*(y_k) &= \omega^*(y_{k-1}) + \mathbb{E}_{k|k-1}\int_0^1 (y_k - y_{k-1})^\top \nabla\omega^*(ty_k + (1-t)y_{k-1})dt \\
&= \omega^*(y_{k-1}) - \gamma\nabla f_k(x_k)^\top\nabla\omega^*(y_{k-1}) \\
&\quad - \gamma\nabla f_k(x_k)\mathbb{E}_{k|k-1}\left\{\int_0^1 \left[\nabla\omega^*(y_{k,t}) - \nabla\omega^*(y_{k-1})\right]dt\right\} \\
&\le \omega^*(y_{k-1}) - \gamma\nabla f_k(x_k)^\top\nabla\omega^*(y_{k-1}) \\
&\quad + \gamma\|\nabla f_k(x_k)\|_\infty\mathbb{E}_{k|k-1}\left\{\int_0^1 \left\|\nabla\omega^*(y_{k,t}) - \nabla\omega^*(y_{k-1})\right\|_1 dt\right\}, \quad (20)
\end{aligned}$$

where the last is due to Hoelder's inequality. By the 1-strong convexity of $\omega(x)$ with respect to the ℓ_1-norm we have

$$\left\|\nabla\omega^*(y') - \nabla\omega^*(y)\right\|_1 \le \|y' - y\|_\infty.$$

Then by 20 we have

$$\mathbb{E}_{k|k-1}\ \{\omega^*(y_k)\} \le \omega^*(y_{k-1}) - \gamma\nabla f_k(x_k)^\top\nabla\omega^*(y_{k-1}) + \frac{\gamma^2\|\nabla f_k(x_k)\|_\infty^2}{2} \quad (21)$$

Taking expectation $\mathbb{E}_N(\cdot)$ with respect to $x_1,\ldots,x_N$ and summing up Ineq. 21 for all k, $1 \le k \le M$, we have

$$\sum_{k=1}^N \gamma\mathbb{E}_N\ \left\{(x_k - x)^\top\nabla f_k(x_k)\right\} \le \omega(x) - \omega(x_0) + \sum_{k=1}^N \frac{\gamma_k^2}{2}\mathbb{E}_N\ \left\{\|\nabla f_k(x)\|_\infty^2\right\} \quad (22)$$

since by Young's inequality $\omega(x) + \omega^*(y) \ge x^\top y$.

By the linearity of each $f_k(x)$ in x, we have:

$$\begin{aligned}
\gamma\sum_{k=1}^N \mathbb{E}_N\left[f_k(x_k) - f_k(x)\right] &\le \sum_{k=1}^N \gamma\mathbb{E}_N(x_k - x)^\top\nabla f_k(x_k) \\
&\le \omega(x) - \omega(x_0) + \sum_{k=1}^N \frac{\gamma^2}{2}\mathbb{E}_N\|\nabla f_k(x_k)\|_\infty^2 \quad (23)
\end{aligned}$$

To finish the proof it remains to note that $\|\nabla f_k(x_k)\|_\infty \le M$ and

$$\psi_N \le \min_{\gamma>0}\frac{\log n + M^2N\gamma^2/2}{N\gamma} = M\sqrt{2N\log n}.$$

The remainder of the proof relies on Azuma's inequality. Let

$$Z_j = \sum_{k=1}^j \gamma(x - x_k)^\top\nabla f_k(x_k)$$

is a Martingale satisfying $|Z_{j+1} - Z_j| \leq c_j \doteq 4M\gamma_k$ almost surely. By Azuma's inequality we have

$$Prob[Z_N \geq t] \leq \exp\left(-\frac{t^2}{2\sum_{j=1}^{N} c_j^2}\right).$$

Setting $t = 4M\gamma\sqrt{2\Omega N}$ finishes the proof of the proposition.

Theorem 3 *Algorithm 4 after $N \geq 16\varepsilon^{-2}(\log 2n + 8\log(2/\delta))$ iterations with a constant step-size $\gamma = \sqrt{2(\log n)/N}$ results in a point $(\bar{x}_N, \bar{y}_N)$ such that with probability at least $1-\delta$, for any $\delta > 0$, one has:*

$$\|A\bar{x}_N\|_\infty \leq \bar{y}_N^\top \tilde{A}\bar{x}_N \leq \varepsilon.$$

Moreover, the total running time of the algorithm is bounded from above as

$$T = O\left(n + \frac{d\log n \log\frac{n}{\delta}}{\varepsilon^2}\right).$$

Proof For a point $(\bar{x}_N, \bar{y}_N)$ defined by Algorithm 4 we have:

$$\begin{aligned}
0 \leq ||A\overline{x}_N||_\infty &= \max_{y\in\Delta_1^{2n}} \langle y, \widetilde{A}\overline{x}_N\rangle - \max_{y\in\Delta_1^{2n}} \min_{x\in\Delta_1^{n}} \langle y, \widetilde{A}x\rangle \\
&\leq \max_{\omega\in\Delta_1^{2n}} \langle y, \widetilde{A}\overline{x}_N\rangle - \min_{x\in\Delta_1^{n}} \langle \overline{y}_N, \widetilde{A}x\rangle \\
&\leq \left\{\max_{y\in\Delta_1^{2n}} \langle y, \widetilde{A}\overline{x}_N\rangle - \frac{1}{N}\sum_{k=1}^{N}\langle y_k, \widetilde{A}x_k\rangle\right\} + \left\{\frac{1}{N}\sum_{k=1}^{N}\langle y_k, \widetilde{A}x_k\rangle - \min_{x\in\Delta_1^{n}} \langle \overline{y}_N, \widetilde{A}x\rangle\right\} \\
&\leq \sqrt{\frac{2}{N}}\Big(\sqrt{\ln(2n)} + 2\sqrt{2\ln(2/\delta)}\Big) + \sqrt{\frac{2}{N}}\Big(\sqrt{\ln n} + 2\sqrt{2\ln(2/\delta)}\Big) \\
&\leq 2\sqrt{\frac{2}{N}}\Big(\sqrt{\ln(2n)} + 2\sqrt{2\ln(2/\delta)}\Big),
\end{aligned}$$

where the last estimate is accurate owing to Proposition 2. That is, with probability at least $1-\delta$, it is sufficient to have

$$N \leq 16\frac{\ln(2n) + 8\ln(2/\delta)}{\varepsilon^2}$$

iterations of the `GK` algorithm in order to guarantee $\|A\overline{x}_N\|_\infty \leq \varepsilon$. Each update of x or y involves an update of no more than d probabilities in vectors p and π corresponding to non-zeros in the gradient. Algorithm 3 requires $O(\log n)$ time to update each. Therefore the time-complexity of the algorithm is bounded from above as

$$O\left(n + \frac{d\ln n(\ln n + \ln(\sigma^{-1}))}{\varepsilon^2}\right).$$

C Missing implementation details

C.1 Sparse matrices storage

In our implementation, we use the popular `CSR` format (Compressed Sparse Row) for sparse matrices. Since we require fast access to the elements of A (function value update) and $A^\top$ (gradient update), we keep both of them in RAM. This doubles the RAM requirements, but significantly improves time complexity.

C.2 Search and update of minimal and maximal gradient coordinate

The most common operations in the `NL1` and `S-FW` algorithms are search, and calculating the minimal and maximal coordinates of the gradient, in addition to updating a small number of them. In order to enable maximal performance, we use binary heaps following to [29].

We implement heaps as binary trees using `C++` templates. This allows the specification of a function that handles data at the compilation stage and guarantees maximal performance compared to function objects (`std::function` + `lambda`), function pointers, or virtual functions.

Since the heaps used in this paper require not only the minimal/maximal coordinates of the gradient but also their indexes, we store a "key-value" pair in each node of the tree. This approach increases the memory consumption of the algorithm but significantly improves its time performance since we can access coordinates directly.

In our experiments, most of the key decrease/increase operations in the heap do not require a full pass from the leaf to the root. For instance, for the `Google` dataset representing the graph of $n = 875713$ vertices, $2^{19} < n < 2^{20}$, the longest path from a leaf to the root has 20 vertices, while, on average, the key increase/decrease operations perform fewer than 5 updates.

C.3 Generation of a random variable with a given distribution

The `GK` algorithm requires the efficient generation of a discrete random variable with prescribed probability distribution (see Section 4 for details). We use computational trees as described in Section C.2, with the difference that the tree implements summation. We optimize the generation method; specifically, a single random draw is sufficient to find the target list/variable:

1. Generate random variable $r : 0 \leq r \leq \tilde{p}$, where $\tilde{p}$ is a probability prescribed to the root;
2. Starting from the root check $r \leq p_a$, where p_a is an (unnormalized) probability value in the child node a;
3. If $r \leq p_a$, follow to node a; otherwise, follow to its sibling b;
4. Continue until a leaf is reached.

C.4 Function value update

None of the proposed algorithms requires the function value, in general. On the other hand, a function value can be useful for early termination of the algorithm. In our experiments, we observe that the function value updates proposed in Sections 2 – 4 are less efficient than the direct update of the sum:

$$f_k = \sum_{i=1}^{n} (b^i)^2, \quad \text{with} \quad b = Ax_k.$$

All methods considered above change a single coordinate x^j on Δx^j, resulting in an update of no more than d coordinates of vector b. Being aware of the values b, we update the function value as:

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

Our experiments also demonstrate that direct updates of the function $\phi(x, \xi)$ in ξ significantly improve the time performance of Algorithm 1. Also, sufficiently large values γ dramatically improve the `NL1` algorithm performance.