Random Projections for Linear Programming: An Improved Retrieval Phase

Leo Liberti, Benedetto Manca, Pierre-Louis Poirion

Random projections for Linear Programming: an improved retrieval phase*

LEO LIBERTI, LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, France
BENEDETTO MANCA, Dipartimento di Matematica e Informatica, Università di Cagliari, Italy
PIERRE-LOUIS POIRION, RIKEN Center for Advanced Intelligence Project, Japan

One way to solve very large linear programs in standard form is to apply a random projection to the constraints, then solve the projected linear program [63]. This will yield a guaranteed bound on the optimal value, as well as a solution to the projected linear program. The process of constructing an approximate solution of the original linear program is called solution retrieval. We improve theoretical bounds on the approximation error of the retrieved solution obtained as in [42], and propose an improved retrieval method based on alternating projections. We show empirical results illustrating the practical benefits of the new approach.

CCS Concepts: • **Mathematics of computing → Combinatorial optimization**; **Probabilistic algorithms**; **Dimensionality reduction**.

Additional Key Words and Phrases: optimization, linear programming, Johnson-Lindenstrauss Lemma

1 INTRODUCTION

Random Projections (RP) are random matrices that act as linear transformations on sets of vectors. They map vectors in $\mathbb{R}^n$ to $\mathbb{R}^k$ with $k \ll n$, and guarantee that certain features of the original vector set remain approximately invariant with high probability in the image of the transformation.

We look at RPs that guarantee approximate congruence, and consider their applications to optimization problems. We formulate such problems using the Mathematical Programming (MP) formal language, and solve them by passing the MP formulation as an input to an appropriate solver, e.g. CPLEX [26] for Linear Programming (LP). The MP formulation of a general optimization problem is:

$$\min\{f_p(x) \mid g_p(x) \leq 0 \wedge x \in Z\},$$

where p is an array of numerical parameters encoding the input of the problem, x is an n-vector of decision variables that will contain the output after the solver has processed the formulation, f_p is a function $\mathbb{R}^n \to \mathbb{R}$, g_p a map $\mathbb{R}^n \to \mathbb{R}^m$, and Z a possibly mixed-discrete subset of $\mathbb{R}^n$.

In general, the application of RPs to MPs of any type requires the following process:

(1) sample a RP matrix T from the appropriate distribution;
(2) project the given MP formulation using T;

*This paper is an extension of the conference paper [39]. The improved solution retrieval and its accompanying theory is new. The computational results refer to larger instances, and show definite improvements w.r.t. [39]. The text was re-written to a large extent.

Authors' addresses: Leo Liberti, liberti@lix.polytechnique.fr, LIX CNRS, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France, 91128; Benedetto Manca, bmanca@unica.it, Dipartimento di Matematica e Informatica, Università di Cagliari, Cagliari, Italy; Pierre-Louis Poirion, pierre-louis.poirion@riken.jp, RIKEN Center for Advanced Intelligence Project, Tokyo, Japan.

1084-6654/2023/8-ART $15.00
https://doi.org/10.1145/3617506

(3) solve the projected MP formulation, collect the optimal solution $\bar{x}$ and its value $\bar{f}$;
(4) in cases where $\bar{x}$ is infeasible w.r.t. the original constraints, perform a *solution retrieval* phase.

In this paper we make two theoretical contributions: we tighten two results about the accuracy of the solution retrieval algorithm proposed in [42] by specializing them to the LP case, and we generalize this solution retrieval algorithm to an Alternating Projection Method (APM). Lastly, we present new computational results on the application of RPs to LPs. The main content differences between this paper and [39] are: (i) a considerable amount of additional theoretical and computational material about the APM, (ii) the proofs of the theoretical results, (iii) the computational results.

1.1 Relevant literature and background

1.1.1 Linear programming. This paper is about efficiently finding approximate solutions of large LPs in standard form — see Eq. (LP) in Sect. 2 below. LP is the most technologically advanced subfield of Mathematical Programming, so most LP instances are routinely solved, exactly, in a matter of seconds (or less) by means of state-of-the-art solvers (e.g. [22, 24, 25]). The algorithms implemented by such solvers are very advanced versions of the simplex algorithm [17] and the interior point method (IPM) [33]. LP solvers are usually tuned to work with extremely sparse constraint matrices, since most LPs modelled by humans have this property. There are three reasons why we propose fast approximate methods for LP: (i) huge LP instances, albeit sparse, may take a long time to solve; (ii) "a few seconds or less" may be too long if a large number of LP solutions are needed to solve a more difficult problem; (iii) even modestly-sized dense LPs may be problematic to solve for many good quality solvers.

A lot of work has been devoted to improving LP algorithms such as the simplex method or the IPM. While the earlier appears to have stabilized [6] (recent advances are more technological than scientific), a lot of research is still ongoing in the latter, some of which also concerns RPs, or more exactly *matrix sketching* (see Sect. 1.1.3 below). Specifically, RPs are applied to the system of complementarity constraints (or their linearization) that often turns out to be the computational bottleneck of IPMs [11, 48, 52–54].

1.1.2 Random projections. The fundamental result in the area of RPs is the Johnson-Lindenstrauss Lemma (JLL) [31], which guarantees approximate congruence with probability tending to one exponentially fast. More precisely, the JLL proves that, given an $m \times n$ matrix A and $\epsilon > 0$, there is an integer $k = O(\frac{1}{\epsilon^2} \ln n)$ such that, if T is a random $k \times m$ matrix sampled componentwise from $\mathbf{N}(0, 1/\sqrt{k})$, where $\mathbf{N}(\mu, \sigma)$ is a Gaussian distribution with mean μ and standard deviation σ, then

$$\mathbf{Prob}\big(\forall i < j \leq n\ (1-\epsilon)\|A_i - A_j\|_2 \leq \|TA_i - TA_j\|_2 \leq (1+\epsilon)\|A_i - A_j\|_2\big) \geq 1 - O(e^{-k}). \tag{1}$$

The kind of probabilistic behaviour exhibited by Eq. (1) is referred to as "with arbitrarily high probability" (wahp).

The application of RPs to raw numerical data is directly justified by the JLL. In the last 30 years, its application to algorithms involving Euclidean distances on large input data has had a considerable impact in the theoretical computer science community [27–30, 44–46, 57] and, a little later, in the numerical algebra community [7, 55, 65]. The application of RPs to optimization problems requires further theoretical efforts in order to turn the approximate congruence invariant to approximate feasibility and optimality: examples in this sense are [10, 15, 38–42, 47, 48, 62–64].

The main reference for RPs and LP in standard form is [63], which presents the theory addressing the solution process given above, a computational study that focuses on dense random LP instances, and a section about the application to error correcting codes. RPs were also applied to some specific LP problems: PAC learning [50] and quantile regression [66], with dimensional reduction techniques tailored to the corresponding LP structure. LPs in canonical form (inequalities instead of equations) are treated in [49].

We note that most of the early papers in RP are purely theoretical, with the exception of [58]. More recent literature includes computational results, either on specific algorithms [8, 48], or specific problems [12, 38, 41], or problem classes [15, 40, 62, 63]. On problem classes, the quality of results can range from excellent to catastrophic, depending on the instance [39].

The first of many computational difficulties (see Sect. 3.1 below) is that, in writing $k = O(\epsilon^{-2} \ln n)$, we are neglecting a constant multiplicative coefficient C_0 related to the "big oh", the appropriate value of which is usually the fruit of guesswork. Another difficulty is that the the theoretical results in this area apply to "high dimensions", without specifying a minimum dimension above which they hold. In catastrophic cases, the theory ensures that results would improve for larger instance sizes, but just how large is unknown. At this time, in our opinion, it is very hard to foresee whether RPs will be useful or not on a given problem instance. This justifies the continuing interest in improving the projection process, and in verifying the empirical advantage of each improvement, both of which are at the core of the contributions of the present paper.

1.1.3 Matrix sketching. Matrix sketching is a technique for applying RPs to whole subspaces instead of finite sets of vectors. The algorithmic treatment is often the same: pre- or post-multiply a given data matrix by a random Gaussian matrix. The theoretical analysis, however, leads to a more powerful result (subspaces instead of finite sets) in exchange for a smaller decrease in size. A clear and unified treatment of JLL and matrix sketching is given in [43, §8.5-8.7]: the projected dimension k goes from $O(\epsilon^{-2} \ln(n))$ in the JLL to $O(\epsilon^{-2} s \log(s))$ in matrix sketching [43, §9.6], where s is the dimension of the subspace we project from. If we are projecting from $\mathbb{R}^n$, we need $k = O(\epsilon^{-2} n \log(n))$. This obviously only helps if $Ax = b$ is overconstrained (yielding columns of A in $\mathbb{R}^s$ with $s \gg n$), which is not the case we consider in this paper. Indeed, matrix sketching is often applied to overconstrained linear systems [65], arising e.g. in linear regression.

The research carried out in matrix sketching and LP follows different directions from the one taken in this paper (i.e. reducing the size of an LP). We look in particular at three papers [11, 53, 54] that apply matrix sketching to the a step of the IPM in order to solve a linear system more efficiently.

In IPMs for LP, the main issue is that of solving complementary constraint systems, which are bilinear constraints (eventually reformulated to linear systems). Quadratic forms are therefore the main object of interest: instead of looking at $AA^\top$ for some matrix A, one considers instead $AT^\top TA^\top$, where T is an appropriately sized RP. In [11], in order to ensure a small value of the bound in the right hand side (rhs) of

$$\|AT^\top TA^\top - AA^\top\|_2 \leq \frac{\zeta}{4}\left(\|A\|_2^2 + \|A\|_F^2/u\right), \tag{2}$$

which holds with probability $\geq 1 - \eta$ for any $u \geq 1$, the estimate of the projected dimension k in [11] is $O(u \ln(u/\eta))$. This result is attributed to [13] by the authors of [11]. It is difficult to compare this result and the JLL since the two statements are about different expressions.

We note that, in order to ensure a small rhs in Eq. (2), ζ should be small. But the authors of [11] do not mention ζ in their order-of-magnitude expression for k because they consider it constant (in [13], on the other hand, it plays a considerable role, as it appears in many complexity expressions as an additional multiplicative factor ζ^{-2}). Moreover, for the term $\|A\|_F^2/u$ to be small, u should be large. Lastly, for Eq. (2) to hold with high probability, η should be small, which makes u/η large. All in all, the JLL expression $k = O(\epsilon^{-2} \ln(n))$ must be compared with $O(\zeta^{-2} u \ln(u/\eta))$ in [11].

A term-by-term comparison becomes possible if we set $u = 1$, but this potentially makes the term $\|A\|_F^2/u$ large. While ϵ and ζ have comparable meanings (i.e., decreasing the error bound), in the JLL expression for k the logarithm is applied to n, which is a part of the instance, whereas η in [11] controls the probability of success of the statement, and is not part of the instance, but rather an algorithmic parameter. By contrast, the JLL parameter that plays the same role as η has the form $O(e^{-k})$, which depends on k, which in turn depends on ϵ and n. Aside

from the fact that k partly depends on the instance in the JLL, while it only depends on algorithmic parameters in [11], it is hard to push the comparison further without making undue assumptions.

The papers [53, 54] also make use of matrix sketching within the IPM algorithm (in addition to several other IPM improvements): the progress along the central path is stochastic, based on sketched matrices. The direction is approximate, but the error is controlled using randomized error bounds, and the number of iterations turns out to be lower than with the classic IPM. This approach does not, by itself, yield an exact solution, but [53] argues that its error guarantees are small enough that a crossover phase [51] will provide the exact solution. In [54] a new type of analysis leads to a matrix distribution $\mathsf{T}(k, n)$ s.t. for a $k \times n$ matrix T with $k \ll n$ sampled from $\mathsf{T}(k, n)$ and any two vectors g and h in $\mathbb{R}^n$, the quantity $g^\top T^\top T g$ approximates $g^\top h$ well. While $TT^\top$ is a full-rank $k \times k$ matrix close to the identity I_k with high probability as shown in [67, Cor. 7], $T^\top T$ is an $n \times n$ matrix, evidently of deficient rank k, which looks componentwise like the identity I_n, and acts like the identity for a given, fixed vector [36, §7.3.2]. Matrices sampled from $\mathsf{T}(k, n)$ are called *coordinate-wise embeddings*. The difference between the main results of [53] and [54] is in the type of error bound they afford: based on Frobenius and ℓ_2 norms in [53], and on ℓ_1 norms in [54].

1.1.4 The formulation and the algorithm. This paper is a follow-up to [42, 63]. In those papers, as well as the present one, we have pursued a formulation-based (rather than algorithmic) line of research. We apply RPs directly to the formulation, rather than to the vectors or matrices that appear in the steps of some solution algorithm. In other words, we use RPs to derive *approximating reformulations* [35]. This is a more general approach than improving the efficiency of a given LP algorithm. Working at the formulation level yields algorithm-independent results. Generality is usually at trade-off with bound quality: this is indeed the case in the present situation, as discussed in Sect. 3.1. It also explains why our results do not always yield accurate solutions of the original problem (but then, even in the case of [53], it is the crossover phase that outputs the exact solution rather than the IPM itself).

On the other hand, unlike the application of RPs to specific algorithms, our results apply to all LP formulations in standard form, independently of the way one chooses to solve them: they say something about the abstract formulation entity rather than a chosen algorithm. For example, an obvious corollary to our general results is that a "randomly projected" simplex method yields an approximate solution with an error bound, which appears to be a new result obtained practically for free. Such overarching consequences cannot be derived from algorithmic papers such as [11, 53, 54], which on the other hand, thanks to their specificity, may obtain better error bounds.

More formally, if $\mathcal{A}$ denotes an LP algorithm and F an LP formulation, the universal approach taken in this paper is:

$$\exists T \; \forall \mathcal{A} \quad \mathcal{A}(\mathsf{F}) \approx \mathcal{A}(T\mathsf{F}), \tag{3}$$

whereas the existential approach is:

$$\exists T \; \exists \mathcal{A} \quad \mathcal{A}(\mathsf{F}) \approx \mathcal{A}(T\mathsf{F}). \tag{4}$$

The approach in Eq. (3) is also known as "sketch-and-solve" in the matrix sketching community. An example of existential approach is given by the "iterate-and-sketch" method employed in the papers [11, 53, 54] discussed above. The fact that, in both Eq. (3) and (4) the RP T is existentially quantified before the algorithm derives from the *oblivious* properties of RPs, namely the fact that that they are sampled from a distribution without any input (except for size) from the set of vectors to be projected.

Even if existential approaches find better solutions than universal ones, we think universal approaches may also be used to construct solutions in practice, as we argue in this paper. For this purpose, we restrict our attention to a specific algorithm, i.e. the APM proposed in Sect. 2.3. But this algorithm acts as a post-processing phase and does not make use of RPs. So we can claim that we remain in the universal framework of Eq. (3).

1.2 Summary

The rest of this paper is organized as follows. In Sect. 2 we present new algorithmic and theoretical results, which concern tightened error bounds, a new retrieval method based on the APM, and some results about its convergence. Sect. 3 describes the computational benchmark: issues in estimating RP-related parameters and how we addressed them, description of the four LP problem structures we consider (maximum flow, diet, quantile regression, basis pursuit) and the corresponding parameters, test pipeline, performance measures. Sect. 4 presents and discusses the computational results obtained for each of our chosen LP structures. Sect. 5 concludes the paper.

2 THEORETICAL RESULTS

An LP formulation in standard form is provided in Eq. (LP), where $c \in \mathbb{R}^n$, A is an $m \times n$ matrix, $b \in \mathbb{R}^n$. Given a $k \times m$ RP matrix T, the projected LP formulation is given in Eq. (T LP).

$$\left.\begin{array}{lrcl} \min & cx & & \\ & Ax & = & b \\ & x & \geq & 0, \end{array}\right\} \quad \text{(LP)} \qquad\qquad \left.\begin{array}{lrcl} \min & cx & & \\ & TAx & = & Tb \\ & x & \geq & 0. \end{array}\right\} \quad (T\,\text{LP})$$

Eq. (T LP) is still an LP in standard form, but it has fewer constraints than Eq. (LP). Since the worst-case running time of all LP solvers also depends on the number of constraints m, solving the latter takes less time than the former. If $\bar{x}$ is an optimal solution of Eq. (T LP) and x^* of Eq. (LP), then $c\bar{x} \leq cx^*$, and there is a $\gamma > 0$ such that $cx^* \leq c\bar{x} + \gamma$ wahp [63].

Both $S = \{x \mid Ax = b\}$ and $TS = \{x \mid TAx = Tb\}$ are affine subspaces of $\mathbb{R}^n$. In general we have $S \subseteq TS$ and $\dim S = n - m < n - k = \dim TS$, which implies that $\bar{x} \in S$ with probability zero. This justifies the need for a post-processing algorithm that takes $\bar{x}$ for input, and provides a solution that is feasible in (LP). Obviously, solving Eq. (LP) directly (i.e. with an LP solver) is not a viable alternative, since otherwise we would not even need to bother with the projected LP. In [42, 63] we proposed instead a simple projection of $\bar{x}$ on the subspace S, which can be achieved with the pseudoinverse of A:

$$\tilde{x} = \bar{x} + A^\top (AA^\top)^{-1}(b - A\bar{x}) \tag{5}$$

The main issue with Eq. (5) is that $\tilde{x}$ may not satisfy the non-negativity constraints $x \geq 0$. We prove in [42] that the negativity error is bounded above wahp. An empirical verification is provided in [63]. We address this issue by iterating an alternation of Eq. (5) and a projection on the variable bounds (Sects. 2.3, 3.4).

The second issue with Eq. (5) is that, since the pseudoinverse is usually dense, its direct application is inefficient. We address this issue by replacing Eq. (5) with an approximation thereof (Sect. 3.5).

2.1 Approximate feasibility invariance

The dual view of the feasible set $F = \{x \mid Ax = b \wedge x \geq 0\}$ of (LP) provides the following geometric interpretation: F describes the set of conic combinations of the columns A^j of A. More precisely, any x in F yields a conic combination $\sum_j x_j A_j$ that is equal to b. We consider $\mathrm{cone}(A)$, the cone spanned by the columns of A, as well as the convex hull $\mathrm{conv}(A)$ of the same. Moreover, for any $y \in \mathrm{cone}(A)$, we let $\|y\|_A = \min\{\sum_j x_j \mid y = \sum_j x_j A^j \wedge x \geq 0\}$ be the *A-norm* of y.

We note that F is invariant w.r.t. the application of T to (LP): if $x \in F$ then $TAx = Tb$ by linearity of T. On the other hand, it is generally false that if $x \geq 0$ but $x \notin F$, then $TAx \neq Tb$. The following approximate feasibility statement

$$b \notin \mathrm{cone}(A) \Rightarrow \mathbf{Prob}\big(Tb \notin \mathrm{cone}(TA)\big) \geq 1 - 2(n+1)(n+2)e^{-C_1(\epsilon^2-\epsilon^3)k} \tag{6}$$

is proved in [63, Thm. 3] for all $\epsilon \in (0, \Delta^2/(\mu_A + 2\mu_A\sqrt{1-\Delta^2}+1))$, where C_1 is the universal constant of the JLL, $\mu_A = \max\{\|x\|_A \mid x \in \mathrm{cone}(A) \wedge \|x\|_2 \leq 1\}$, and Δ is a lower bound to $\min_{x \in \mathrm{conv}(A)} \|b - x\|_2$.

Let $\text{val}(\cdot)$ indicate the optimal objective function value of a MP formulation. The approximate optimality statement for (LP) derived in [63, Thm. 4] is conditional to the LP formulation being feasible and bounded, so that, if x^* is an optimal solution, there is a scalar θ (assumed w.l.o.g. ≥ 1) such that $\sum_j x^*_j < \theta$. Given $\gamma \in (0, \text{val(LP)})$,

$$\textbf{Prob}\big(\text{val(LP)} - \gamma \leq \text{val}(T\text{LP}) \leq \text{val(LP)}\big) \geq 1 - \eta, \tag{7}$$

where $\eta = 4ne^{-C_1(\epsilon^2-\epsilon^3)k}$, $\epsilon = O(\gamma/(\theta^2\|y^*\|_2))$, and y^* is an optimal dual solution of Eq. (LP). Like other approximate optimality results in this field, some quantities in the probabilistic statement depend on the norm of a dual optimal solution. This adds a further difficulty to computational evaluations, since they cannot be computed prior to solving the problem.

Let $\bar{x}$ be a *projected solution*, i.e. an optimal solution of the projected formulation. In [63, Prop. 3], it is proved that $\bar{x}$ is feasible in the original formulation with zero probability. We therefore need to provide a way to construct a feasible solution of the original formulation, i.e. a *solution retrieval* method. A couple were proposed in [63], but the one found in [42, Eq. (6)] comes with an approximation guarantee and a good practical performance (see Sect. 4.5). As already mentioned, the retrieved solution $\tilde{x}$ is defined as the projection of $\bar{x}$ on the affine subspace $Ax = b$, and computed using the pseudoinverse, see Eq. (5). The fact that we only project on $Ax = b$ without enforcing $x \geq 0$ is necessary, since otherwise we would need to solve the whole high-dimensional LP. On the other hand, it causes potential infeasibility errors w.r.t. $x \geq 0$.

A probabilistic bound on this error is cast in general terms for Conic Programs (CP) in [42], where the analysis is conducted in the framework of Formally Real Jordan Algebras (FRJA). This allows the same analysis to hold for many types of CPs [2], including LPs. Specifically, the central result to the bound on the negativity error of the retrieved solution $\tilde{x}$ is the M^* bound theorem [59, Thm. 9.4.2], which bounds (wahp) the expected diameter of the set $\ker A \cap B$ in function of the radius and the Gaussian width of B, where B is any subset of $\mathbb{R}^n$. We recall that the radius of B is $\max_{x\in B}\|x\|_2$, and that the Gaussian width of B is $\mathbb{E}_g(\sup_{x\in B}\langle g, x\rangle)$, where g is a standard normal vector in $\mathbb{R}^n$, and $\mathbb{E}_g$ is the mean w.r.t. the random vector g. Both Prop. 4.2 and Thm. 4.4 in [42], which bound (wahp) the feasibility error of $\bar{x}$ and the negativity error of $\tilde{x}$, are consequences of the M^* bound theorem, by estimating radius and Gaussian width of an appropriately chosen set B. The foremost technical difficulty of applying the M^* bound theorem is to find good estimations of the Gaussian width.

The first theoretical contribution of this paper is a specialization of [42, Thm. 4.4] — originally cast in the general conic case — to the case of the non-negative orthant (i.e. the case of LP). This result bounds the negativity of the smallest component of $\tilde{x}$ in terms of $\bar{x}$ and the condition number κ of A.

Proposition 2.1. *There is a universal constant C_2 such that, for any $u \geq 0$, we have:*

$$\textbf{Prob}\big(\min_{j\leq n}\tilde{x}_j \geq \min_{j\leq n}\bar{x}_j - \epsilon\theta\kappa(C_2 + u\sqrt{2/\ln(n)})\big) \geq 1 - 2e^{-u^2}. \tag{8}$$

The proof is based on an improvement of [42, Eq. (7)] based on computing the Gaussian width and diameter of $\{x \geq 0 \mid \langle \mathbf{1}, x\rangle \leq 1\}$.

Proof. The inequality above is given in general terms in [42, Eq. (7)] as:

$$\lambda_{\min}(\tilde{x}) \geq \lambda_{\min}(\bar{x}) - \epsilon\theta\kappa\|Q^{1/2}\|_2(C_2 w(\mathbb{B}) + \Delta u)/\sqrt{\ln(n)},$$

where λ are generalized eigenvalues in a FRJA $\mathbb{A}$, Q is the realization of the bilinear form of $\mathbb{A}$, $w(\cdot)$ is the Gaussian width, $\mathbb{B} = \{x \in \mathcal{K}_{\mathbb{A}} \mid \langle \mathbf{e}, x\rangle \leq 1\}$ (with $\mathcal{K}_{\mathbb{A}}$ the cone of squares of $\mathbb{A}$), and Δ is the diameter of $\mathbb{B}$. For LP, $\mathcal{K}_{\mathbb{A}}$ is the non-negative orthant and $Q = I_n$, so $\lambda_i(v)$ is the i-th component of the vector v in order of increasing value, whence $\lambda_{\min}$ is the component of minimum value, $\|Q^{1/2}\|_2 = 1$, $\mathbf{e} = \mathbf{1}$, and $\mathbb{B} = \text{conv}(\{0, e_1, \ldots, e_n\})$. Now the Gaussian width of a finite set F is $w(F) = O(\sqrt{\ln|F|})$ and $w(F) = w(\text{conv}(F))$ [59], therefore $w(\mathbb{B}) = O(\sqrt{\ln(n)})$. Finally, $\|0 - e_i\|_2 = 1$ for all $i \leq n$, and $\|e_i - e_j\|_2 = \sqrt{2}$ for each $i < j \leq n$, so the diameter of $\mathbb{B}$ is $\sqrt{2}$. The result follows by replacement. □

2.2 Approximate optimality of the retrieved solution

As a corollary to Prop. 2.1, we also have the following result about the difference between objective function values of the retrieved and projected solutions.

Corollary 2.2. *Let $\tilde{f}$ be the objective function value of the retrieved solution $\tilde{x}$, and $\bar{f}$ be the optimal objective function value of the projected formulation. There is a universal constant C_2 such that, for any $u \geq 0$, we have:*

$$\mathbf{Prob}\big(|\tilde{f} - \bar{f}| \leq \epsilon\theta\kappa\|c\|_2(C_2 + u\sqrt{2/\ln(n)}) \big) \geq 1 - 2e^{-u^2}. \tag{9}$$

Proof. Note that $\tilde{f} = c^\top\tilde{x}$ and $\bar{f} = c^\top\bar{x}$. By Prop. 2.1, we have $\tilde{f} = c^\top\bar{x} + c^\top(A^\top(AA^\top)^{-1}(b - A\bar{x}))$. By Cauchy-Schwartz inequality, this yields $|\tilde{f} - \bar{f}| \leq \|c\|_2\|A^\top(AA^\top)^{-1}\|_2\|b - A\bar{x}\|_2$. The result follows from [42, Prop. 4.2] and [42, Thm. 4.4]. □

2.3 Improved solution retrieval

Our proposed improvement to the solution retrieval method shown in Eq. (5) and analysed in Prop. 2.1 and Cor. 2.2 is based on alternating two projection steps: the aforementioned projection on $Ax = b$ (see Eq. (5)), and a projection on the non-negative orthant $x \geq 0$, which simply consists in zeroing all negative components of $\tilde{x}$.

This suggests using the well-known APM [61, Thm. 13.10]. For two closed convex sets S, U with non-empty intersection, the APM projects the current point on S, then the result on U, and repeats these two steps. The APM converges (possibly in infinite time) to a point in the intersection $S \cap U$. In the current set-up, we have $S = \{x \mid Ax = b\}$ and $U = \{x \mid x \geq 0\}$. The first projection is implemented as in Eq. (5). The second zeroes all the negative components of the input vector.

There are many variants of the APM. We present the most basic one (von Neumann's [61]) in this section. We introduce and discuss two others in Sect. 3.4.

The new retrieval method is presented in Alg. 1. It takes as input: (i) the problem data A, b, (ii) the projected solution $\bar{x}$, and (iii) two algorithmic parameters: the maximum number N of iterations, and a numeric threshold $\delta > 0$ for the error $\|A\tilde{x} - b\|_2$. The loop in Alg. 1 is executed at most N times. It alternates between achieving

Algorithm 1 APM$(A, b, \bar{x}, \mathsf{N}, \delta)$

```
x̃ ← x̄
for i ≤ N do
  x̃ ← x̃ + Aᵀ(AAᵀ)⁻¹(b − Ax̃)    // project on Ax = b
  if x ≥ 0 then
    return x̃    // x̃ feasible
  end if
  for j ∈ {h ≤ n | x̃_h < 0} do
    x̃_j ← 0    // project on x ≥ 0
  end for
  if ‖Ax̃ − b‖₂ ≤ δ then
    return x̃    // x̃ almost feasible
  end if
end for
return x̃
```

feasibility w.r.t. $Ax = b$ and w.r.t. $x \geq 0$. Alg. 1 returns a solution $\tilde{x} \leftarrow \text{APM}(A, b, \tilde{x}, \delta)$ which is closer to the feasible set of Eq. (LP) than the projected solution $\bar{x}$.

2.4 Error and termination of Alg. 1

The APM in Alg. 1 converges to a point $\hat{x}$ in the intersection of $Ax = b$ and $x \geq 0$. More in general, von Neumann's APM converges to a point in the intersection of two closed convex sets S, U if the intersection is non-empty [18]. It is also well known that it may take infinite time to reach $\hat{x}$, and that convergence can be slow. The APM can be extended to any finite number of closed convex sets $S_1, \ldots, S_q$ [23]. Alg. 1 can therefore be generalized accordingly, for example in order to deal with different sets in SDPs that have equality, inequality, and range constraints ($q = 3$ in this case). In this section we analyse the termination of Alg. 1.

Let us define, for notational convenience, $S = \{x \in \mathbb{R}^n \mid Ax = b\}$ and $U = \mathbb{R}^n_+$ (the non-negative orthant). For any $x \in \mathbb{R}^n$ we let $x^+ = (\max(0, x_j) \mid j \leq n)$, $x^- = (\min(0, x_j) \mid j \leq n)$, and $\mathrm{neg}(x) = |\mathrm{supp}(x^-)|$. We denote by $d(S, T)$ the minimum Euclidean distance between a set S and a set T in the same Euclidean space. If either $S = \{s\}$ or $U = \{t\}$ are singletons, we indicate them by s, t instead of $\{s\}, \{t\}$ in the $d(\cdot, \cdot)$ expression (thus we write $d(s, T)$, $d(S, t)$, or $d(s, t)$).

Assuming that the original LP is feasible, the APM defines a sequence of points $\bar{x} = x^0, x^1, \ldots, \hat{x}$, where x^k is the orthogonal projection of x^{k-1} on S for odd k, and on U for even k. For general APMs this sequence may be infinite. Alg. 1, however, terminates in finite time with $\hat{x} = \tilde{x}$ when the infeasibility w.r.t. $Ax = b$ becomes smaller than a given $\delta > 0$. So we assume that there is a last index N of the sequence. We note that $x^{2h+1} = x^{2h} + A^\top (AA^\top)^{-1}(b - Ax^{2h})$ for all $h \geq 0$, and that $x^{2h} = (x^{2h-1})^+$ for all $h \geq 1$. Since projections are orthogonal, if k is odd we have $d(x^{k-1}, S) = d(x^{k-1}, x^k)$ and $d(x^k, U) = d(x^k, x^{k+1})$. Since APMs ensure that distances between successive pairs of iterates decrease, we have:

$$\forall k \geq 1 \quad d(x^{k+1}, x^k) \leq d(x^k, x^{k-1}). \tag{10}$$

We now find bounds on δ in Alg. (1) and u (appearing in Eq. (8)-(9)) in terms of several expressions, among which

$$\mathcal{A} = \sqrt{\ln(n)/2}\left(C_2 - \frac{\min_h \bar{x}_h}{\epsilon\theta\kappa}\right), \quad \mathcal{B} = \sqrt{\frac{\ln(n)}{2\mathrm{neg}(x^1)}}\epsilon\theta\kappa,$$

and $\mathcal{D}_u = \sqrt{\mathrm{neg}(x^1)}\left|\min_h \bar{x}_h - \epsilon\theta\kappa(C_2 + u\sqrt{2/\ln(n)})\right|$.

Proposition 2.3. *Let $u \geq 0$, $\delta \geq \mathcal{D}_u$, and $x^{\mathsf{N}} = \tilde{x}$ be the last iterate of the sequence given by Alg. 1. If $\mathsf{N} > 1$, then $\mathcal{A} - \mathcal{B}\delta \leq u \leq \mathcal{A} + \mathcal{B}\delta$.*

Proof. Consider the first point $x^1 \in S$ obtained by projecting the point $\bar{x}$ onto S. Since $\mathsf{N} > 1$, we must have $\min_j x^1_j < 0$, otherwise Alg. 1 would terminate. We can bound the distance between x^1 and the set U: using (8), for all $u \geq 0$ we have

$$\begin{aligned} d(x^1, U)^2 &= d(x^1, x^2)^2 = \|x^1 - (x^1)^+\|^2 = \sum_{\substack{j \leq n \\ x^1_j < 0}} (x^1_j)^2 \\ &\leq \sum_{\substack{j \leq n \\ x^1_j < 0}} \left(\min_h \bar{x}_h - \epsilon\theta\kappa(C_2 + u\sqrt{2/\ln(n)})\right)^2 = \mathcal{D}_u^2 \end{aligned} \tag{11}$$

with probability at least $1 - 2^{-u^2}$. Now, let k be an even index of the sequence. Using (10) and (11) we infer:

$$d(x^k, S) = d(x^k, x^{k+1}) \leq d(x^1, x^2) = d(x^1, U) \leq \mathcal{D}_u$$

with probability at least $1 - 2^{-u^2}$. Therefore, since $\mathcal{D}_u \leq \delta$, Alg. (1) terminates at iteration $\mathsf{N} = k$ and $\tilde{x} = x^{\mathsf{N}}$ is the retrieved solution. The last inequality gives us a bound on u as follows:

$$
\begin{aligned}
& \mathcal{D}_u \leq \delta \\
\Rightarrow \quad & \left| u\left(\sqrt{\frac{2}{\ln(n)}}\epsilon\theta\kappa\right) + \min_h \bar{x}_h - \epsilon\theta\kappa C_2 \right| \leq \frac{\delta}{\sqrt{\mathrm{neg}(x^1)}} \\
\Rightarrow \quad & \frac{\delta}{\sqrt{\mathrm{neg}(x^1)}} - \min_h \bar{x}_h + \epsilon\theta\kappa C_2 \leq u\left(\sqrt{\frac{2}{\ln(n)}}\epsilon\theta\kappa\right) \leq \frac{\delta}{\sqrt{\mathrm{neg}(x^1)}} + \min_h \bar{x}_h - \epsilon\theta\kappa C_2 \\
\Rightarrow \quad & \sqrt{\frac{\ln(n)}{2}}\left(C_2 - \frac{\min_h \bar{x}_h}{\epsilon\theta\kappa}\right) + \frac{\delta\sqrt{\ln(n)}}{\sqrt{2\mathrm{neg}(x^1)}\epsilon\theta\kappa} \leq u \leq \sqrt{\frac{\ln(n)}{2}}\left(\frac{\min_h \bar{x}_h - C_2}{\epsilon\theta\kappa}\right) + \frac{\delta\sqrt{\ln(n)}}{\sqrt{2\mathrm{neg}(x^1)}\epsilon\theta\kappa} \\
\Rightarrow \quad & \mathcal{A} - \mathcal{B}\delta \leq u \leq \mathcal{A} + \mathcal{B}\delta,
\end{aligned}
$$

where $\mathcal{A}$ and $\mathcal{B}$ are as defined above. □

We observe that the quantity $\mathcal{D}_u$ can be relatively large and give a loose upper bound for $d(x^1, U)$. However, if we consider $\mathcal{D}_u$ as a function of only u and impose $\mathcal{D}_u \geq 0$ we can find the value of u for which $\mathcal{D}_u$ reach its minimum:

$$
\mathcal{D}_u = \sqrt{\mathrm{neg}(x^1)}\left|\min_h \bar{x}_h - \epsilon\theta\kappa(C_2 + u\sqrt{2/\ln(n)})\right| \geq 0 \Leftrightarrow u \leq \left(\frac{\min_h \bar{x}_h}{\epsilon\theta\kappa} - C_2\right)\sqrt{\frac{\log(n)}{2}} \tag{12}
$$

As a corollary of Prop. 2.3, since Alg. 1 terminates when $d(x^{\mathsf{N}}, U) \leq \delta$ and $\mathcal{D}_u$ is an upper bound for $d(x^1, U)$ (and thus for $d(x^2, S)$), we have the following

Corollary 2.4. *Under the same assumptions of Prop. 2.3, Alg. 1 terminates with* $\mathsf{N} = 2$.

The following result shows that, by fixing the parameter δ (resp. N) in Alg. 1, it is possible to obtain an upper (resp. lower) bound on the parameter N (resp. δ). These bounds will give the possibility to choose the maximum number of iterations needed to reach a certain feasibility error δ w.r.t. the set S, or to choose the required feasibility error δ to reach in a certain number of iterations N.

The aforementioned bounds are constructed by considering the convergence rate of the APM algorithm, i.e. the constant $\mu \in (0, 1)$ such that

$$
d(x^k, \hat{x}) \leq \mu d(x^{k-1}, \hat{x}) \quad \forall k > 0, \tag{13}
$$

where $\hat{x}$ is the accumulation point of the sequence $\{x^k\}$, the fact that the distance between x^1 and x^2 is bounded (cf. Eq. (11)) and the worst case scenario for the APM algorithm with convergence rate μ (for more details on the convergence rate of the APM algorithm we refer to [18]).

Theorem 2.5. *Let* $S = \{x \in \mathbb{R}^n \mid Ax = b\}$, $U = \{x \in \mathbb{R}^n \mid x \geq 0\}$, $\mu \in (0, 1)$ *be the convergence rate of Alg. 1,* $\mathcal{D}_u$ *the quantity defined in Eq.* (11) *and* $\alpha = \arccos(\mu)$. *For* $\mathsf{N} \in \mathbb{N}$ *fixed and even, Alg. 1 terminates after* N *iterations if the feasibility error* δ *w.r.t. the set* S *satisfies*

$$
\delta \geq \mu^{\mathsf{N}-1}\frac{\mathcal{D}_u}{\tan(\alpha)}. \tag{14}
$$

For $\delta > 0$ *fixed, the number of iterations* N *required to terminate Alg. 1 with feasibility error* δ *w.r.t.* S *satisfies*

$$
\mathsf{N} \leq \frac{\ln(\tan(\alpha)\delta/D)}{\ln(\cos(\alpha))} + 1. \tag{15}
$$

Proof. We consider any line $\sigma \subset \mathbb{R}^n$ passing through x^2 perpendicular to the vector $x^2 - x^1$ and another line τ passing through x^1 intersecting σ with angle α satisfying $\cos(\alpha) = \mu$. By construction, the APM algorithm applied to σ and τ starting from x^1, satisfies

$$\forall k > 1 \quad d(y^k, \hat{y}) = \cos(\alpha) d(y^{k-1}, \hat{y}) = \mu d(y^{k-1}, \hat{y}), \tag{16}$$

where $\{y^k\}$ is the sequence of points generated by the algorithm with $y^i = x^i$, $i = 1, 2$, and $\hat{y}$ is its accumulation point.

From Eq. (16) we can conclude that the sequence $\{y^k\}$ converges more slowly than the sequence $\{x^k\}$ generated by Alg. 1. Therefore, if we bound the index N such that the sequence $\{y^k\}$ satisfies $d(y^{\mathsf{N}}, \tau) \leq \delta$, the same bound holds for the sequence $\{x^k\}$.

In order to find such bound, let $\mathsf{N} \in \mathbb{N}$ be even, then

$$d(y^{\mathsf{N}}, \tau) \leq d(y^{\mathsf{N}}, \hat{y}) = \mu^{\mathsf{N}-1} d(y^1, \hat{y}) = \mu^{\mathsf{N}-1} \frac{d(y^1, y^2)}{\tan(\alpha)} = \mu^{\mathsf{N}-1} \frac{d(x^1, x^2)}{\tan(\alpha)} \leq \mu^{\mathsf{N}-1} \frac{\mathcal{D}_u}{\tan(\alpha)}, \tag{17}$$

where, in the last step, we have used Eq. (11). Since the APM algorithm terminates when $d(y^{\mathsf{N}}, \tau) \leq \delta$, we obtain the required bound on N as follows:

$$\mu^{\mathsf{N}-1} \frac{\mathcal{D}_u}{\tan(\alpha)} \leq \delta \Leftrightarrow \mathsf{N} \leq \frac{\ln(\tan(\alpha)\delta/\mathcal{D}_u)}{\ln(\cos(\alpha))} + 1, \tag{18}$$

where we recall that $\cos(\alpha) = \mu$. We also observe that the left hand side of (18) gives the required lower bound on the feasibility error δ for fixed N. □

3 COMPUTATIONAL BENCHMARK

As mentioned in the introduction, the computational application of RPs to problem classes (such as LP) is questionable, because performance varies from problem to problem, as well as from instance to instance. Our goal is to provide a computational comparison: how much precision do we lose, and how much time do we gain, by solving projected instead of original instances?

3.1 The inapplicability of RP theory

Most theoretical results using RPs are of the "wahp" type: a certain statement occurs with probability tending to one exponentially fast in terms of the projected size (see e.g. Eq. (6)). The statements themselves are often bounds to a feasibility or optimality error expressed in terms of certain features of the problem instance (see e.g. Eq. (7) and (8)): for example the number n of variables and the condition number κ of the constraint matrix.

Among these features, some relate to the solution of the original problem: for example the boundedness assumption (encoded by θ) and the norm of the solution vector y^* of the dual. Evidently, such information is not available: if it were, the original problem would already have been solved, and there would be no need to solve the projected problem. Another issue is raised by the presence of the universal constants C_0, C_1, C_2: while for specific instances or problem structures one may be able to compute reasonable estimates [5] for the constants related to measure concentration, to the best of our knowledge, their estimation over general problem classes such as LP is useless.

This situation prevents a straightforward application of RP theory to practical cases. As mentioned in [39, §3.3], easy bounds for θ and $\|y^*\|$ lead to tiny values for ϵ, which imply huge values for k. In turn, this means that the smallest sizes n to be considered for the applications of RPs to LP should be exponentials of huge values: practically impossible. One might then question the significance of these results. Our answer is that they have an epistemological value: they say that inserting randomness in an LP before solving it is not as crazy as it would

appear. Unfortunately, they do not say just how to go about choosing the parameters for the obtained solution to be meaningful.

3.2 Choice of parameters

Our computational set-up reflects the need for solving both the original and the projected instances, so as to be able to compare results. Accordingly, we choose a range of sizes n, k so that our solver of choice can solve both original and projected instances.

We arbitrarily fix $C_0 = 1$ so as to be able to derive ϵ as $\sqrt{\ln m/k}$. Since we do not know how to choose C_1, C_2, and we can only obtain useless estimates for θ, $\|y^*\|_2$, we give up entirely on γ, δ, and therefore also on estimating the probability of success *a priori*.

In the implementation of the APM (Alg. 1), we chose a termination condition based on the maximum number N of iterations rather than on the error threshold δ w.r.t. $\|A\tilde{x} - b\|_2$, since if the APM takes too long, the entire endeavor of projecting instances becomes moot. In our computational experiments, we set $\mathsf{N} = 30$ and $\delta = 0.01$. We also added a further termination test to the APM, namely that if $\|x^k - x^{k+1}\|_2 < \delta$ at iteration k, then the APM terminates prematurely.

3.3 Density of the projection matrix

All componentwise sampled sub-Gaussian distributions [19] can be used to prove the results cited in this paper. Some sparse variants also exist, along the lines of [1, 32]. We use the sparse RPs described in [15, §5.1]. For a given density $\sigma \in (0, 1)$ and standard deviation $\sqrt{1/(k\sigma)}$, with probability σ we sample a component of the $k \times m$ RP T from the distribution $\mathsf{N}(0, \sqrt{1/(k\sigma)})$, and set it to zero with probability $1 - \sigma$. In our computational study, we set $\sigma = 0.5 d_A$, where d_A is the density of the constraint matrix A (the multiplicative constant 0.5 was chosen empirically on the basis of a small subset of instances).

3.4 Alternating projection variants

We considered three APM variants: (i) von Neumann's basic one [61] given in Alg. 1 and used in [39], (ii) Dykstra's one [20], and (iii) the Averaged Alternating Reflections Method (AARM) [4]. While von Neumann's APM is designed to simply find any point in the intersection of convex sets $S, U \subset \mathbb{R}^n$, Dykstra's variant and AARM find the projection of a given point x_0 on $S \cap U$, i.e. they find $x \in S \cap U$ such that $\|x - x_0\|_2$ is minimum.

Von Neumann's APM and the AARM can be described as operators $\mathbb{R}^n \to S \cap U$ composed of two distinct projection operators $P_S : \mathbb{R}^n \to S$ and $P_U : \mathbb{R}^n \to U$. We have:

$$\begin{aligned} T_{\text{VN}} &= P_S \circ P_U \\ T_{\text{AARM}} &= (1 - \alpha)I + \alpha(2\beta P_S - I) \circ (2\beta P_U - I), \end{aligned}$$

where $\alpha, \beta \in [0, 1]$ were both set to 0.9. The Dykstra variant makes use of extra variables p, q, as shown in Alg. 2. All three algorithms come with some form of convergence guarantee: for details, we refer the reader to the corresponding references [4, 20, 61], as well as [14].

In the context of Alg. 1, S is a (possibly unbounded) hyper-rectangle $x^L \le x \le x^U$, where x^L, x^U are lower and upper ranges for the decision variables (LP), and U is the affine subspace $Ax = b$.

In order to choose the best variant for our purposes, we ran some preliminary tests on a small part of our benchmark instances to compare these three variants. Dykstra's APM gave the best trade-off between solution quality and speed.

Algorithm 2 The Dykstra APM

Input: starting point $x \in \mathbb{R}^n$
Output: candidate $x \in S \cap U$
  $p = 0, q = 0$
  while $\|A(x+p) - b\|_2 \geq$ tolerance **do**
    $y = P_S(x+p)$
    $p \leftarrow x + p - y$
    $x = P_U(y+q)$
    $q \leftarrow y + q - x$
  end while

3.5 Approximating the pseudoinverse

Although Alg. 1 makes use of the pseudo-inverse of A in order to compute the projection P_U of the current iterate $\tilde{x}$ on $Ax = b$, the computation of the pseudo-inverse of a large-sized matrix is costly, even if A is sparse (since the pseudoinverse is usually dense), and even if it need be computed only once, at the beginning of Alg. 1. In this section we present a fast heuristic method for computing $\arg\min\{\|x - x_0\|_2 \mid Ax = b\}$ for any given x_0 (we note that here x_0 plays the role of the current iterate $\tilde{x}$ in Alg. 1).

Specifically, by "fast" we mean "fast in our Python implementation" based on `scipy.sparse` [60]. This library offers two relevant functions:

(1) `linalg.lsmr`, an iterative solver for least-squares problems: it solves $\arg\min_x \|Ax - b\|_2$, and provides a solution x of minimum ℓ_2 norm if there are multiple solutions to $Ax = b$;
(2) `linalg.lsqr`, which solves $\arg\min_x(\|Ax - b\|_2 + \mu\|x - x_0\|_2)$ where μ is given by the user: our preliminary tests with varying μ never managed to provide us with satisfactory results, as either $\|Ax - b\|_2$ was too large, or $\|x\|_2$ was close to zero (and generally far from x_0).

We therefore decided to focus on `lsmr` only. We computed $\check{x} = \texttt{lsmr}(A, b)$ once only before the APM loop, and $\hat{x} = \texttt{lsmr}(A, b + Ax_0)$ with varying x_0 at every iteration of the APM loop. We note that `lsmr` provides solutions having minimum ℓ_2 norm, so `lsmr` approximately solves $\arg\min\{\|x\|_2 \mid Ax = b\}$. Therefore $\hat{x}$ is an approximate solution of $\arg\min\{\|x - x_0\|_2 \mid A(x - x_0) = b\}$. Let $x' = \hat{x} - x_0$: then $Ax' = A(\hat{x} - x_0) = b$ by definition of $\hat{x}$. This implies that we now know two points on the affine subspace $Ax = b$, namely $\check{x}$ and x'. We look for a point $x^\dagger$ that minimizes $\|x - x_0\|_2^2$ over the line $L(\lambda) = \lambda\check{x} + (1 - \lambda)x'$: this can be done analytically, as it suffices to replace x with $L(\lambda)$ in $\|x - x_0\|_2^2$, take the derivative w.r.t. λ, and set it to zero, as the resulting function of λ is convex. This yields

$$x^\dagger = \frac{\langle \check{x} - x', x_0 - x' \rangle}{\|\check{x} - x'\|_2^2}(\check{x} - x') + x'. \tag{19}$$

By construction $x^\dagger$ is at least as close to x_0 as $\check{x}$ and $\hat{x}$, and generally closer.

As a final algorithmic post-processing, we implemented a bisection search over the segment between $x^\dagger$ (feasible in $Ax = b$) and the projected solution $\bar{x}$ of TLP (generally infeasible in $Ax = b$), aimed at optimizing the objective $\langle c, x \rangle$ of LP while keeping the iterates feasible w.r.t. $Ax = b$. We found that, while the improvements of this step were very small in general, the CPU time it took is negligible.

3.6 The problem set

A computational study of unstructured LPs was already provided in [63]. Here we study the same set of randomly generated structured LPs from the problems Max Flow, Diet, Quantile Regression, Basis Pursuit. This is the

same application set found in [39]; in this paper we present new results related to more (and larger) instances per problem.

3.6.1 Maximum flow. The Max Flow formulation is defined on a weighted digraph $G = (N, \mathcal{A}, u)$ with a source node $s \in N$, a target node $t \in N$ (with $s \neq t$) and $u : \mathcal{A} \to \mathbb{R}_+$, as follows:

$$\left.\begin{array}{rrcl} \max\limits_{x \in \mathbb{R}_+^{|\mathcal{A}|}} & \sum\limits_{\substack{i \in N \setminus \{s\} \\ (s,i) \in \mathcal{A}}} x_{si} & - & \sum\limits_{\substack{i \in N \setminus \{s\} \\ (i,s) \in \mathcal{A}}} x_{is} \\ \forall i \in N \setminus \{s,t\} & \sum\limits_{\substack{j \in N \\ (i,j) \in \mathcal{A}}} x_{ij} & = & \sum\limits_{\substack{j \in N \\ (j,i) \in \mathcal{A}}} x_{ji} \\ \forall (i,j) \in \mathcal{A} & 0 \leq & x_{ij} & \leq u_{ij}. \end{array}\right\} \quad \text{(MF)}$$

We generate random weighted digraphs $G = (N, \mathcal{A}, u)$ with the property that a single (randomly chosen) node s is connected (through paths) to all of the other nodes: we first generate a random tree on $N \setminus \{t\}$, orient it so that s is the root, add a node t with the same indegree as the outdegree of s, and then proceed to enrich this digraph with arcs generated at random using the Erdős-Renyi model with probability 0.05. We then generate the capacities u uniformly from $[0, 1]$. Finally, we compute the digraph's incidence matrix A, which has $m = |N| - 2$ rows and $|\mathcal{A}|$ columns. Instances are feasible because the graph always has a path from s to t by construction, and the zero flow is always feasible.

Although (MF) is an LP, it is not in standard form, because of the upper bounding constraints $x \leq u$. But, by [63, §4.2], we can devise a block-structured RP matrix that only projects the equations $Ax = b$, leaving the inequalities $x \leq u$ alone. In this case, A is a flow matrix with two nonzeros per column, one set to 1 the other to -1, aside from columns referring to source and target nodes s, t that only have one nonzero; and $b = 0$. The density of A is $d_A = \frac{2|\mathcal{A}|-2}{(m-2)|\mathcal{A}|} \approx 2/m$.

For our random (MF) instances, $\theta = |\mathcal{A}|$ is a valid upper bound to $\sum_{(i,j) \in \mathcal{A}} x^*_{ij}$, since $0 \leq x_{ij} \leq u_{ij} \leq 1$ for all $(i, j) \in \mathcal{A}$.

3.6.2 Diet problem. The Diet formulation is defined on an $m \times n$ nutrient-food matrix D, a food cost vector $c \in \mathbb{R}^n_+$, and a nutrient requirement vector $b \in \mathbb{R}^m$, as follows:

$$\left.\begin{array}{rcl} \min\limits_{q \in \mathbb{R}^n_+} & c^\top q & \\ & Dq & \geq \quad b. \end{array}\right\} \quad \text{(DP)}$$

We sample c, D, b uniformly componentwise in $[0, 1]$, and set the density of D to $d_D = 0.5$. Instances are feasible because one can always buy enough food to satisfy all nutrient requirements. If $\|D_i\|_0 = |\text{nonzeros of row } D_i|$, then $\hat{q} = \left(\max\limits_{i \leq m}(b_i/(\|D_i\|_0 D_{ij})) \mid j \leq n\right)$ is a feasible solution.

We remark that Diet is the only problem in our test set that does not natively include a system of linear equations. The transformation using slack variables $r_i \geq 0$ for $i \leq m$ is immediate. We let $A = (D \mid -I)$, where I is $m \times m$. The decision variable vector is $x = (q, r)$. The density of A is $d_A = (d_D mn + m)/(m(n+m)) = (d_D n + 1)/(n+m)$. The theory of RP would justify a fair comparison only between original and projected formulations both in standard form. Since, however, this paper is about a *practical* comparison, and since no-one would convert (DP) to standard form before solving it — because this type of transformations is carried out by solvers automatically, and in the best way — we chose to compute objective function values and feasibility errors of the projected formulation on the space of the original formulation variables q. Thus, for a retrieved solution $\tilde{x} = (\tilde{q}, \tilde{r})$ we only considered $\tilde{q}$ in order to compute the objective function value corresponding to $\tilde{x}$.

Considering only the q variables is unproblematic if applied to the optimal solution $x^* = (q^*, r^*)$ of the original formulation in standard form, because $r^* \geq 0$ and $A = (D| -I)$ ensure that q^* is a feasible solution in $Dq \geq b$. When applied to the projected formulation, however, $TA = (TD| -TI)$ yields a block matrix $TI = T$ with both

positive and negative entries, since T is sampled from a normal distribution. Thus, the underdetermined $k \times m$ system $Tr = Tb$ has nontrivial solutions with probability 1. In this case, since the objective tends to minimize $c^\top q$, the projected solution $\bar{x} = (\bar{q}, \bar{s})$ will have $\bar{q} = 0$, yielding zero projected objective function value. This, in turn, may yield $D\tilde{q} \not\geq b$. The application of RPs to Diet (and any other LP in canonical form) is not straightforward.

We consider two alternative projected formulations. The first is a scalarization of a bi-objective version of Diet where $\sum_j r_j$ is also minimized:

$$\left.\begin{array}{rcl} \min c^\top q & + & \mu \mathbf{1}^\top r \\ TDq - Tr & = & Tb \\ q & \in & \mathbb{R}^n_+ \\ r & \in & \mathbb{R}^m_+ \end{array}\right\} \quad (T\text{DP}^1),$$

where μ is a given positive scalar. The second aims at minimizing the infeasibilities of the projected system $TDq = Tb$:

$$\left.\begin{array}{rcl} \min c^\top q & + & \mu \mathbf{1}^\top (r^+ + r^-) \\ TDq + I_k(r^+ - r^-) & = & Tb \\ q & \in & \mathbb{R}^n_+ \\ r^+, r^- & \in & \mathbb{R}^k_+. \end{array}\right\} \quad (T\text{DP}^2)$$

For (DP), the upper bounding solution $\hat{q}$ yields slack values $\hat{r}_i = D_i\hat{q} - b_i$ for all $i \leq m$, where D_i is the i-th row of D. So we let $\theta = \sum_j \hat{q}_j + \sum_i \hat{r}_i$ be an upper bound for $\sum_j x^*_j$.

3.6.3 Quantile regression. The Quantile Regression formulation, for a quantile $\tau \in (0, 1)$, is defined over a database table D having density d_D with m records and p fields, and a further column field b. We make a statistical hypothesis $b = \sum_j \beta_j D^j$, and aim at estimating $\beta = (\beta_j \mid j \leq p)$ from the data b, D so that errors from the τ-quantile are minimized. Instances may only have nonzero optimal value if $m > p$, as is clear from the constraints of the formulation below:

$$\left.\begin{array}{ll} \min\limits_{\substack{\beta \in \mathbb{R}^p \\ u^+, u^- \in \mathbb{R}^m_+}} & \tau \mathbf{1}^\top u^+ + (1-\tau)\mathbf{1}^\top u^- \\ & D\beta + Iu^+ - Iu^- = b, \end{array}\right\} \quad (\text{QR})$$

where the constraint system $Ax = b$ has $A = (D|I| - I)$, $x = (\beta, u^+, u^-)$, and τ (the quantile level) is given, and fixed at 0.2 in our experiments. The data matrix (D, b) is sampled uniformly componentwise from $[-1, 1]$, with $d_D = 0.8$. Instances are all feasible because the problem reduces to solving the overconstrained linear system $D\beta = b$ with a skew variant of an ℓ_1 error function.

We note that (QR) is not in standard form, since the components of β are unconstrained; but this is not an issue, insofar as the problem is bounded (since it is feasible and it minimizes a weighted sum of non-negative variables), and this is enough for the results in [63] to hold. On the contrary, the lack of non-negative bounds on β is an advantage, since we need not worry about negativity errors in the β components of the retrieved solution (Prop. 2.1). The density of A is $d_A = (d_D mp + 2m)/(mp + 2m^2) = (d_D p + 2)/(p + 2m)$.

For (QR), given that all data is sampled uniformly from $[-1, 1]$, no optimum can ever have $|\beta_j| > 1$. As for u^+, u^-, we note that any feasible β yields an upper bound to the optimal objective function value, which only depends on u^+, u^-: we can therefore choose $\beta = 0$, and obtain $u^+_i - u^-_i = b_i$ for all $i \leq m$; we then let $u^+_i = b_i \wedge u^-_i = 0$ if $b_i > 0$, and $u^+_i = 0 \wedge u^-_i = -b_i$ otherwise. This yields an upper bound estimate $\theta = p + \sum_i |b_i|$ to $\sum_j x^*_j$.

3.6.4 Basis pursuit. The Basis Pursuit formulation aims at finding the sparsest vector x satisfying the underdetermined linear system $Ax = b$ by resorting to a well-known approximation of the zero-norm by the ℓ_1 norm

[9]:

$$\left.\begin{array}{lll} \min\limits_{x,s\in\mathbb{R}^n} & \mathbf{1}^\top s & \\ & Ax = b & \\ \forall j \le n & -s_j \le x_j & \le s_j. \end{array}\right\} \quad \text{(BP)}$$

According to sparse coding theory [16], we work with a fully dense $m \times n$ matrix A sampled componentwise from $\mathsf{N}(0,1)$ (with density $d_A = 1$), a random message obtained as z/Z from a sparse $z \in (\mathbb{Z} \cap [-Z, Z])^n$ (with density 0.2) and $Z = 10$, and compute the encoded message $b = Az$. We then solve (BP) in order to recover the sparsest solution of the underconstrained system $Ax = b$, which should provide an approximation of z. Basis pursuit problems undergo a phase transition as m decreases from n down to zero [3], so it shouldn't really make sense to decrease m by using RPs, and yet some mileage can unexpectedly be extracted from this operation [37].

Similarly to (MF), in (BP) we can partition the constraints into equations $Ax = b$ and inequalities $-s \le x \le s$. Again by [63, §4.2], we devise a block-structured RP matrix which only projects the equations.

As in Sect. 3.6.3, (BP) is not in standard form, since none of the variables are non-negative. In this case, moreover, it is not easy to establish a bound θ on $\sum_j (x^*_j + s^*_j)$, since A is sampled from a normal distribution. On the other hand, for $A_{ij} \sim \mathsf{N}(0,1)$ we have $\mathbf{Prob}\big(A_{ij} \in [-3,3]\big) = 0.997$. By construction, we have $b \in [-3n, 3n]^m$, which implies a defining interval $[-n, n]$ on the components of optimal solutions, yielding $\theta = 2n^2$ with probability 0.997.

3.7 Our pipeline: hardware and software

The solution pipeline is based on Python 3.11.2 [56] and the libraries `scipy` [60] and `amplpy` [21] (besides other standard python libraries). For each problem type, we loop over instance (row) sizes in a certain set $\mathcal{S}$, over ϵ values in a certain set $\mathcal{E}$, and over 5 different runs for each pair (row size, ϵ) in order to amortize the randomness given by sampling T. We report average performance measures over the 5 instances.

We solve all of the original instances using CPLEX 22.1 [26]. The CPLEX settings are default for the original instances (so the LP algorithm within CPLEX is chosen automatically). For the projected instances, we use the HiGHS [24] LP solver: more specifically, we use the revised simplex algorithm for the Maximum Flow and Quantile Regression applications, and the barrier algorithm without crossover for the Diet application. For the projected instances of the Basis Pursuit application we use the CPLEX solver (barrier algorithm).

We are aware that adapting the solution process to the application removes generality from the algorithm in exchange for performance. This can be easily fixed by simply choosing CPLEX or HiGHS with the automatic choice of algorithm. With HiGHS, the performance degradation is very small for all but Basis Pursuit, in which case only some of the combination of (instance, ϵ) will be impacted severely. Using CPLEX for all projected instances will slow down most runs by a factor between 1 and 10, which is only noticeable in Maximum Flow. This factor could well be reduced further by employing a direct interface to CPLEX rather than using AMPLpy.

All our tests have been carried out on a 128-core Intel Xeon Platinum 8362 2.8GHz CPU running on a CentOS Rocky Linux 8.8 distribution with 2TB RAM.

Our code can be downloaded from `github.com/leoliberti/rp4lp`: we warn readers that it is a research code, and therefore prone to changes, sometimes difficult to decypher, and with plenty of parameters we used to test algorithmic choices. Some manual adaptation of the parameter setting is carried out before deploying the code on a new application.

3.8 Performance measures

At the end of each solver call we record: the optimal objective function f^* of the original problem, the optimal objective function $\bar{f}$ of the projected problem, the objective function value $\tilde{f}$ of the retrieved solution $\tilde{x}$, the average feasibility error w.r.t. range inequalities $L \le x \le U$ (`avgin`), equation constraints $Ax = b$ (`avgeq`), the

CPU time t^* taken to solve the original formulation, and the CPU time $\tilde{t}$ taken to construct and solve the projected formulation and retrieve the solution.

To be precise, the CPU time t^* takes into account: reading the instance, constructing the original formulation, and solving it. The CPU time $\tilde{t}$ takes into account: reading the instance, sampling the RP, projecting the instance data, constructing the projected formulation, solving it, and performing solution retrieval.

In summary, this benchmark is based on the following measures: the average objective function ratios $\bar{f}/f^*$, $\tilde{f}/f^*$, the average errors avgeq, avgin for $Ax = b$ and $x \geq 0$, the ratio k/m, the average CPU ratio $\bar{t}/t^*$: all averages are computed over 5 solution runs over a given instance and ϵ value.

4 COMPUTATIONAL RESULTS

We present computational results in tabular and graphical forms.

For each problem in our set (Max Flow, Diet, Quantile Regression, Basis Pursuit) we randomly sampled five reasonably large-sized instances (generation details in Sect. 3.6, size details in specific sections below), solved them with the process described in Sect. 3.7 for a set $\mathcal{E} = \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ of ϵ values, and collected the performance measures described in Sect. 3.8.

In all tested instances, the proposed RP-based solution methodology took less time than running CPLEX with default settings: in other words, $\tilde{t} < t^*$ for all instances and runs. In Table 1 we give a succinct summary of the performance comparison: for each application problem we give averages (over all instances, ϵ values, and runs) of: original objective value f^*, projected objective value $\bar{f}$, retrieved objective value $\tilde{f}$, maximum range error over variables, maximum equation error over linear equality constraints, CPU time t^* to solve the original problem, CPU time $\tilde{t}$ to construct and solve the projected problem and retrieve the solution. Although we do not present instance sizes in this table, the CPLEX average CPU times t^* bear witness to the fact that these are not easy LPs to solve.

Problem	f^*	$\bar{f}$	$\tilde{f}$	maxin	maxeq	t^*	$\tilde{t}$
Max Flow	147.562	148.233	74.296	0.000	0.000	47.70	33.33
Diet TDP^1	0.243	0.830	1.817	0.033	0.187	3.65	1.40
Diet TDP^2	0.243	0.831	0.973	0.039	0.683	3.68	0.22
Quantile Regression	1361.976	0.000	1400.650	0.001	0.000	10.44	1.43
Basis Pursuit	19.820	7.987	89.509	0.000	0.000	649.58	7.57

Table 1. Average results over all instances, ϵ values, and runs.

Since we are trying to ascertain how a new solution methodology works, we believe that the relative behaviour change of objective function values and CPU times in function of instance size and ϵ will give the reader a clearer picture than their absolute counterpart. In the following detailed tables 2, 3, 4, 5, 7 we shall therefore show:

- the value of ϵ,
- the ratio $\tilde{f}/f^*$ of projected to original objective function value (the closer to 1 the better),
- the ratio $\tilde{f}/f^*$ of retrieved to original objective function value (the closer to 1 the better),
- the average error avgin on the inequality constraints $x \geq 0$ (the closer to zero the better),
- the average error avgeq on the equality constraints $Ax = b$ (the closer to zero the better),
- the ratio k/m of the number of projected to original constraint (tends to zero as ϵ increases),
- the ratio $\tilde{t}/t^*$ of the CPU time taken to construct and solve the projected problem and then perform the solution retrieval, and of the CPU time taken to solve the original problem (the lower the better: a successful run when < 1).

In each of Figures 1, 2, 3, 4, 5 we give 15 plots for each test set arranged in a 5×3 array, corresponding to three plots per instance on each row, with instance size increasing downwards. Each plot maps two related performance measures (y axis) in function of increasing ϵ (x axis). The leftmost column of the array plots $\bar{f}/f^*$ and $\tilde{f}/f^*$. The center column plots avgin and avgeq. The rightmost column plots k/m and $\tilde{t}/t^*$.

4.1 Maximum flow

We generated Max Flow instances with constraint matrices having row sizes in

$$\{5000, 5500, 6000, 6500, 7000\}$$

(number of nodes aside from source and destination), and column sizes (number of arcs) approximately as the row size $\times 1.05$. the arc capacities were sampled from $\textsc{Uniform}(0, 1)$.

The application of RPs to Max Flow is very successful according to most measures. The projected optimal value is always very close to the original. The retrieved solution has objective value that is always close to half of the original one, which is not ideal, but not bad either (we do not think this regularity is due to chance, but we could not understand its reason). The errors are always very close to zero, so the retrieved solution is practically feasible. The CPU time $\tilde{t}$ is always smaller than t^*.

Having said that, the results in Table 2 do not wholly justify choosing to use RPs rather than simply solve the problem with a good LP solver: even though RPs take less time than CPLEX, t^* and $\tilde{t}$ are in the same order of magnitude. The point is what to do when the original problem is so large it cannot be solved, so that RPs become the only alternative.

We sampled an instance having row size 20000 for which our pipeline fails to solve due to size, but were able to solve it in 400.27s of CPU time, with a retrieved solution that is practically feasible in the original LP. We note that the reason why this instance could not be solved is not the CPLEX solver, but a rather old version of AMPL [21] (called by the `amplpy` library). So our test is realistic rather than real, but in the setting of an academic paper the argument should remain convincing: increasing sizes further would eventually challenge the solver. To be fair to AMPL, we also note that replacing the AMPL binary with a recent version allowed our pipeline to solve the (original) instance in 531.76s, yielding $\bar{f}/f^* \approx 1$ and $\tilde{f}/f^* \approx 0.5$ as in the rest of the instances.

4.2 Diet

We generated Diet instances with constraint matrices D having row sizes in

$$\{3000, 4000, 5000, 6000, 7000\}$$

(number of nutrients) and column size fixed to 400 (number of foods). The objective function coefficients were sampled from Uniform$(0, 1)$. The given constant μ in Eq. (TDP^1)-(TDP^2) was set to $2n$, to attempt to drive slack variables to zero.

The application of RPs to Diet presents an abnormal behaviour of the relaxation property of the projected problem w.r.t. the original one: namely, it is a minimization problem, so we should expect $\bar{f} \leq f^*$, and we find instead that $\bar{f}$ is often much larger than f^*. This is due to the fact that, according to Sect. 3.6.2, we use modified projected formulations for which the theory no longer holds. We therefore ignore the projected value, and look at the other measures instead. It turns out that TDP^1 yields smaller errors but worse retrieved solution values in more CPU time than TDP^2.

It is worth mentioning that a theory of RPs applied to LPs that are natively cast in canonical form (i.e. with inequality constraints) is presented in [49]: χ^2 distributions are used instead of subgaussian distributions in order to sample the RP T. The theory in [49] is partial because it does not propose a solution retrieval method accompanied by a concentration of measure result. But it does show that the error between projected value and

ϵ	$\tilde{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
			maxflow-5000			
0.20	1.0000	0.5055	0.0000	0.0000	0.0702	0.78
0.30	1.0000	0.5027	0.0000	0.0000	0.0308	0.71
0.40	1.0000	0.5025	0.0000	0.0000	0.0168	0.68
0.50	1.0000	0.5023	0.0000	0.0000	0.0112	0.64
0.60	1.0000	0.5012	0.0000	0.0000	0.0084	0.67
0.70	1.0000	0.5016	0.0000	0.0000	0.0056	0.66
0.80	1.0000	0.5024	0.0000	0.0000	0.0056	0.67
0.90	1.0000	0.5011	0.0000	0.0000	0.0028	0.60
			maxflow-5500			
0.20	1.0000	0.5058	0.0000	0.0000	0.0645	0.74
0.30	1.0000	0.5019	0.0000	0.0000	0.0284	0.67
0.40	1.0000	0.4978	0.0000	0.0000	0.0155	0.63
0.50	1.0000	0.4962	0.0000	0.0000	0.0102	0.61
0.60	1.0000	0.4953	0.0000	0.0000	0.0076	0.61
0.70	1.0000	0.4953	0.0000	0.0000	0.0051	0.61
0.80	1.0000	0.4957	0.0000	0.0000	0.0051	0.60
0.90	1.0000	0.4955	0.0000	0.0000	0.0025	0.52
			maxflow-6000			
0.20	1.0000	0.5096	0.0000	0.0000	0.0600	0.84
0.30	1.0000	0.5051	0.0000	0.0000	0.0263	0.74
0.40	1.0000	0.5040	0.0000	0.0000	0.0143	0.77
0.50	1.0000	0.5046	0.0000	0.0000	0.0095	0.68
0.60	1.0000	0.5035	0.0000	0.0000	0.0072	0.69
0.70	1.0000	0.5036	0.0000	0.0000	0.0047	0.69
0.80	1.0000	0.5041	0.0000	0.0000	0.0047	0.68
0.90	1.0000	0.5031	0.0000	0.0000	0.0023	0.63
			maxflow-6500			
0.20	1.0000	0.5035	0.0000	0.0000	0.0560	0.81
0.30	1.0000	0.4999	0.0000	0.0000	0.0246	0.75
0.40	1.0000	0.4991	0.0000	0.0000	0.0134	0.71
0.50	1.0000	0.4991	0.0000	0.0000	0.0089	0.69
0.60	1.0000	0.4998	0.0000	0.0000	0.0066	0.66
0.70	1.0000	0.4995	0.0000	0.0000	0.0045	0.66
0.80	1.0000	0.4986	0.0000	0.0000	0.0045	0.68
0.90	1.0000	0.4978	0.0000	0.0000	0.0022	0.63
			maxflow-7000			
0.20	1.0198	0.5149	0.0000	0.0000	0.0524	0.87
0.30	1.0198	0.5126	0.0000	0.0000	0.0230	0.79
0.40	1.0198	0.5113	0.0000	0.0000	0.0126	0.79
0.50	1.0198	0.5107	0.0000	0.0000	0.0083	0.73
0.60	1.0198	0.5098	0.0000	0.0000	0.0063	0.73
0.70	1.0198	0.5105	0.0000	0.0000	0.0041	0.73
0.80	1.0198	0.5101	0.0000	0.0000	0.0041	0.73
0.90	1.0198	0.5103	0.0000	0.0000	0.0020	0.66

Table 2. Max Flow results.

original value cannot be excessively different with arbitrarily high probability. Convincing computational results are presented in this sense.

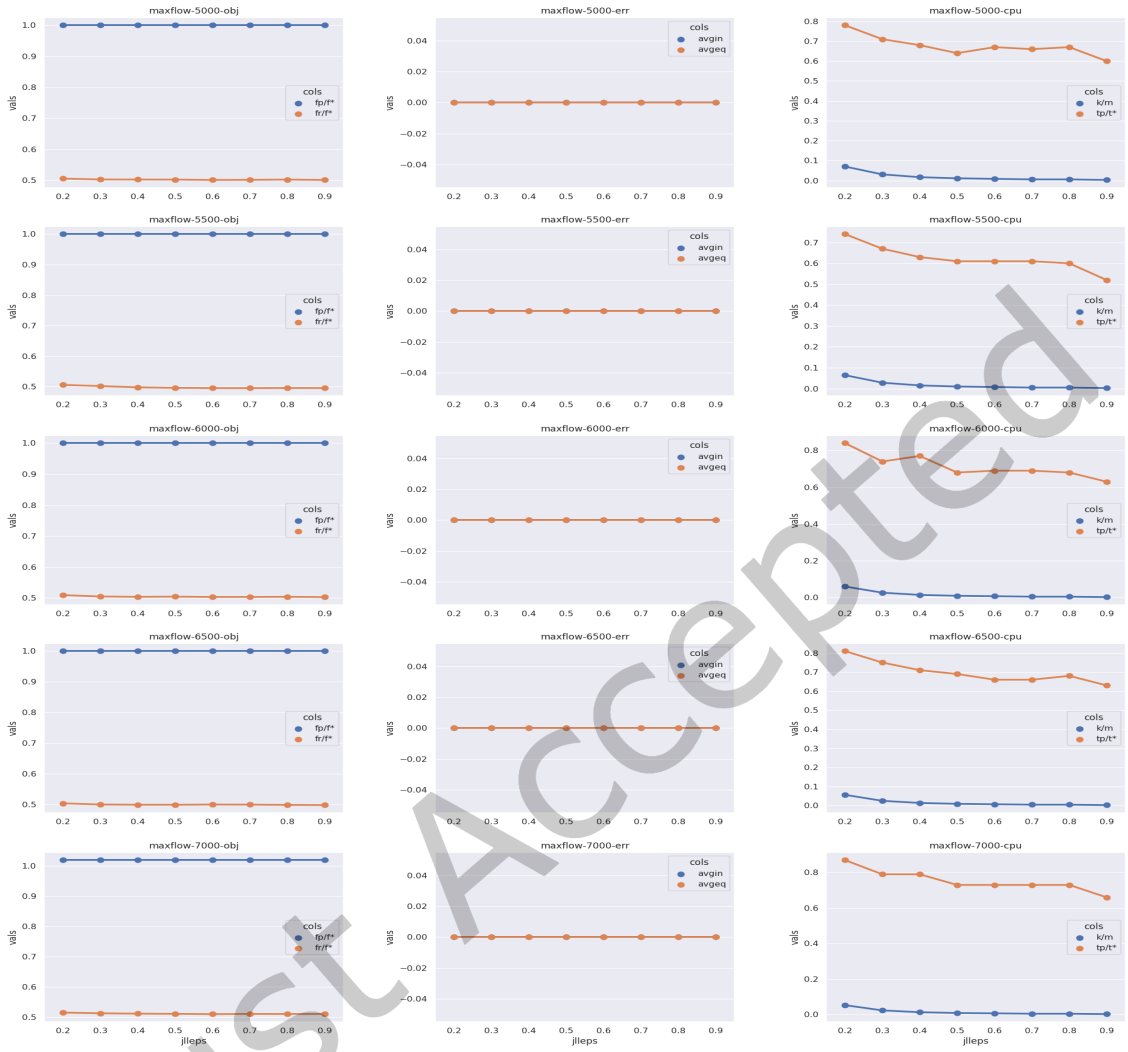

Fig. 1. Max Flow plots.

4.3 Quantile regression

We generated Quantile Regression instances with data matrices D with components from Uniform$(-1, 1)$ having row sizes in

$$\{5000, 6000, 7000, 8000, 9000\}$$

(number of records) and column size fixed at 400 (number of features). The constraint matrices have $p = 399$ columns corresponding to the independent features; the right-hand side is the 400-th column of D (dependent feature).

The application of RPs to Quantile Regression yielded excellent results everywhere aside from the projected solution value $\bar{f}$, which was always very close to zero. This denotes a poor relaxation quality, since the problem

ϵ	$\bar{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
diet-3000						
0.20	4.2167	6.7057	0.0042	0.0035	0.0677	0.58
0.30	4.6096	6.7329	0.0042	0.0036	0.0297	0.46
0.40	4.8227	6.7726	0.0042	0.0037	0.0160	0.44
0.50	4.5505	6.7598	0.0042	0.0037	0.0107	0.43
0.60	3.7557	6.7526	0.0042	0.0037	0.0080	0.43
0.70	1.5655	6.7539	0.0042	0.0037	0.0053	0.43
0.80	1.6198	6.7534	0.0042	0.0037	0.0053	0.43
0.90	0.3558	6.7541	0.0042	0.0037	0.0027	0.42
diet-4000						
0.20	5.3459	8.6092	0.0030	0.0040	0.0522	0.51
0.30	5.1756	8.5729	0.0030	0.0041	0.0230	0.44
0.40	4.9819	8.5718	0.0030	0.0041	0.0125	0.42
0.50	6.1089	8.5732	0.0030	0.0041	0.0083	0.41
0.60	4.6691	8.5740	0.0030	0.0041	0.0063	0.41
0.70	1.5736	8.5705	0.0030	0.0041	0.0040	0.41
0.80	1.4134	8.5701	0.0030	0.0041	0.0040	0.41
0.90	0.8319	8.5695	0.0030	0.0041	0.0020	0.41
diet-5000						
0.20	5.0390	7.8446	0.0017	0.0042	0.0428	0.47
0.30	4.7305	7.8167	0.0017	0.0043	0.0188	0.40
0.40	4.4483	7.8166	0.0017	0.0043	0.0102	0.38
0.50	5.4889	7.8220	0.0017	0.0043	0.0068	0.38
0.60	3.5377	7.8292	0.0017	0.0044	0.0050	0.38
0.70	3.3575	7.8246	0.0017	0.0044	0.0034	0.37
0.80	2.8848	7.8266	0.0017	0.0044	0.0034	0.37
0.90	0.4368	7.8265	0.0017	0.0044	0.0016	0.37
diet-6000						
0.20	4.4477	7.9308	0.0014	0.0043	0.0365	0.43
0.30	5.0808	7.9087	0.0014	0.0044	0.0160	0.37
0.40	5.2797	7.9091	0.0014	0.0044	0.0087	0.36
0.50	5.2320	7.9066	0.0014	0.0044	0.0058	0.36
0.60	3.6482	7.9054	0.0014	0.0044	0.0043	0.35
0.70	2.3367	7.9060	0.0014	0.0044	0.0028	0.35
0.80	1.5380	7.9063	0.0014	0.0044	0.0028	0.35
0.90	0.2818	7.9059	0.0014	0.0044	0.0013	0.35
diet-7000						
0.20	4.1153	6.7361	0.0011	0.0045	0.0317	0.40
0.30	4.1567	6.7149	0.0011	0.0045	0.0140	0.34
0.40	3.6206	6.7017	0.0011	0.0046	0.0076	0.33
0.50	4.3503	6.7149	0.0011	0.0046	0.0050	0.33
0.60	5.0517	6.7136	0.0011	0.0046	0.0037	0.33
0.70	1.4903	6.7127	0.0011	0.0046	0.0024	0.33
0.80	1.1576	6.7125	0.0011	0.0046	0.0024	0.33
0.90	0.5228	6.7132	0.0011	0.0046	0.0011	0.33

Table 3. Diet results (with projected formulation $T\text{DP}^1$).

is in minimization form. Looking at the results in [39], the reason is likely to be the fact that $\epsilon = 0.2$ is excessive for the considered instance sizes. Further tests with $\epsilon \in \{0.10, 0.15\}$ increased $\bar{f}$ (but not significantly), at the expense of longer computation times (Table 6). In particular, in the size range given in Table 6, the "sweet spot" for ϵ for Quantile Regression is around 0.15.

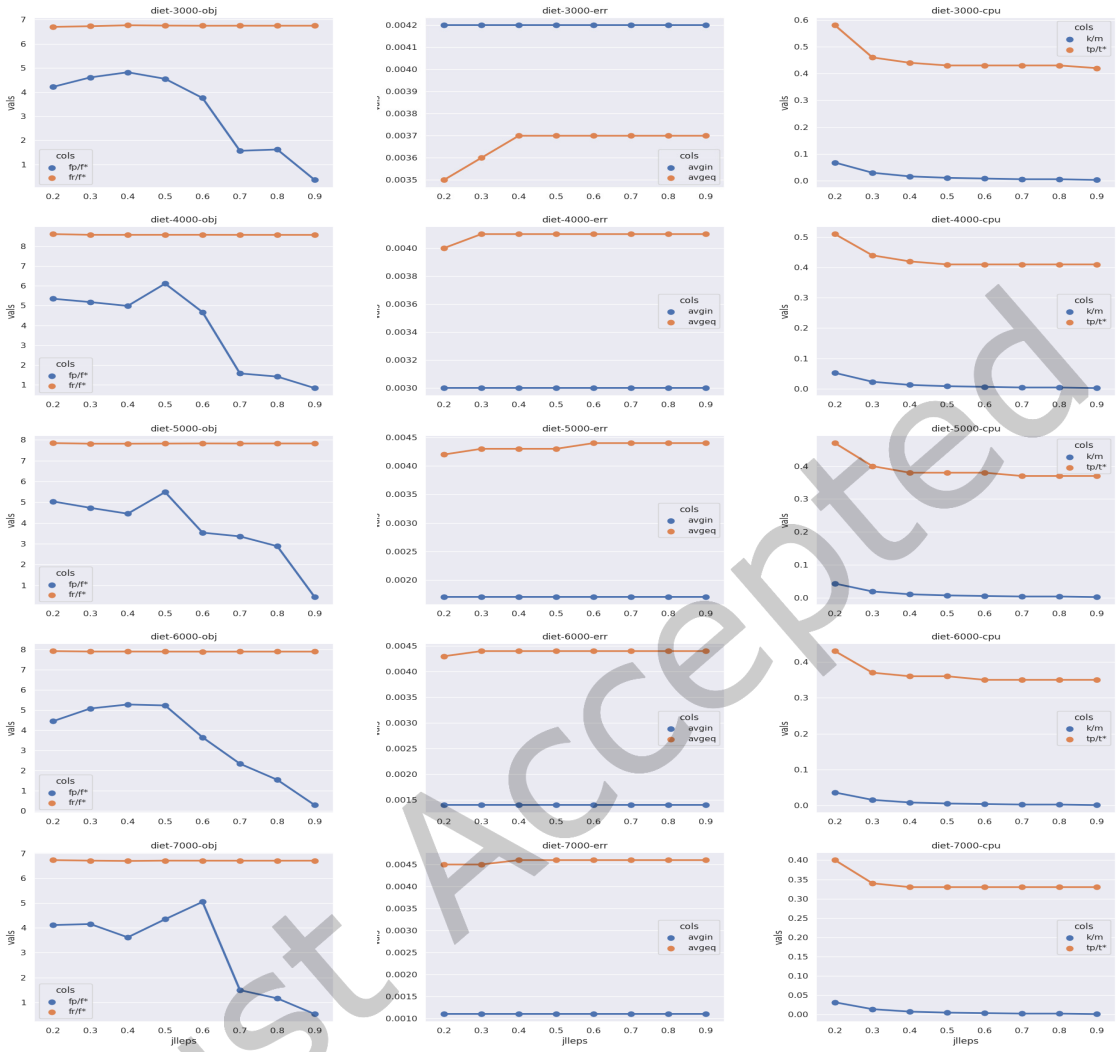

Fig. 2. DIET plots from projected formulation TDP^1.

4.4 Basis pursuit

We generated BASIS PURSUIT instances with encoding matrices A having row sizes in

$$\{1000, 2000, 3000, 4000, 5000\},$$

column sizes equal to row sizes +1000, and density 0.1. The (right-hand size) compressed vector b is obtained by sampling a vector x and setting $b = Ax$.

The application of RPs to BASIS PURSUIT yielded excellent results overall, specially at $\epsilon = 0.2$ for sizes up to 3000: the projected problem has almost the same value as the original one, and the retrieved solution is perfectly

ϵ	$\tilde{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
diet-3000						
0.20	4.3407	3.8700	0.0047	0.1141	0.0677	0.23
0.30	4.1194	3.8696	0.0047	0.1141	0.0297	0.08
0.40	4.6687	3.8698	0.0047	0.1141	0.0160	0.06
0.50	4.5584	3.8697	0.0047	0.1141	0.0107	0.06
0.60	3.8724	3.8700	0.0047	0.1141	0.0080	0.05
0.70	1.3965	3.8698	0.0047	0.1141	0.0053	0.05
0.80	1.3862	3.8700	0.0047	0.1141	0.0053	0.05
0.90	0.2955	3.8695	0.0047	0.1141	0.0027	0.05
diet-4000						
0.20	5.3094	4.5163	0.0041	0.1186	0.0522	0.19
0.30	4.8067	4.5163	0.0041	0.1186	0.0230	0.07
0.40	5.7862	4.5163	0.0041	0.1186	0.0125	0.05
0.50	5.9006	4.5163	0.0041	0.1186	0.0083	0.05
0.60	4.4612	4.5163	0.0041	0.1186	0.0063	0.05
0.70	1.2559	4.5163	0.0041	0.1186	0.0040	0.04
0.80	1.5625	4.5163	0.0041	0.1186	0.0040	0.04
0.90	0.5322	4.5163	0.0041	0.1186	0.0020	0.04
diet-5000						
0.20	4.9491	3.9406	0.0033	0.1203	0.0428	0.16
0.30	4.6016	3.9406	0.0033	0.1203	0.0188	0.06
0.40	5.0448	3.9406	0.0033	0.1203	0.0102	0.05
0.50	4.7865	3.9406	0.0033	0.1203	0.0068	0.04
0.60	6.0189	3.9405	0.0033	0.1203	0.0050	0.04
0.70	2.0706	3.9406	0.0033	0.1203	0.0034	0.04
0.80	1.9125	3.9406	0.0033	0.1203	0.0034	0.04
0.90	0.4111	3.9406	0.0033	0.1203	0.0016	0.04
diet-6000						
0.20	4.3454	4.0887	0.0026	0.1200	0.0365	0.14
0.30	5.0528	4.0887	0.0026	0.1200	0.0160	0.06
0.40	5.4369	4.0887	0.0026	0.1200	0.0087	0.04
0.50	5.0093	4.0887	0.0026	0.1200	0.0058	0.04
0.60	4.7694	4.0887	0.0026	0.1200	0.0043	0.04
0.70	1.7530	4.0887	0.0026	0.1200	0.0028	0.04
0.80	1.8450	4.0887	0.0026	0.1200	0.0028	0.04
0.90	0.2088	4.0887	0.0026	0.1200	0.0013	0.04
diet-7000						
0.20	4.0452	3.7501	0.0023	0.1221	0.0317	0.13
0.30	4.2951	3.7501	0.0023	0.1221	0.0140	0.05
0.40	4.6308	3.7501	0.0023	0.1221	0.0076	0.04
0.50	4.5838	3.7501	0.0023	0.1221	0.0050	0.04
0.60	3.8588	3.7501	0.0023	0.1221	0.0037	0.04
0.70	2.0121	3.7501	0.0023	0.1221	0.0024	0.03
0.80	1.5508	3.7501	0.0023	0.1221	0.0024	0.04
0.90	0.4129	3.7501	0.0023	0.1221	0.0011	0.03

Table 4. DIET results from projected formulation $T\text{DP}^2$.

feasible in the original problem. Moreover, $\tilde{t}$ is only 4% to 12% of t^*. Larger sizes fare more poorly as far as objective value is concerned, but the solution time is 1%: these are indications that ϵ should be decreased.

See Table 8 for results where $\mathcal{E} = \{0.05, 0.10, 0.15, 0.20\}$. We can see that the instances up to $m = 3000$ produce wrong results for $\epsilon = 0.05$: this is due to the fact that $k/m > 1$ (i.e. $k > m$): the LP solver, presented with fully dense over-constrained systems $TAx = Tb$, attempts to reduce their ranks, and fails due to floating point inaccuracies

Fig. 3. DIET plots from projected formulation TDP^2.

and extreme memory demands, which in turn yields an infeasible status and meaningless solutions. However, for larger instances, where $k < m$, we see a decreasing CPU time ratio trend, which means that for huge instances, a very small ϵ is appropriate. The same trend can be read from the other values of ϵ at all sizes. We can also see the impact that extreme values of ϵ have on performance: smaller ϵ values yield better solution quality until $k > m$ occurs, while large ϵ values yield poorer $\bar{f}$ and $\tilde{f}$ but with a decrease in CPU time. In particular, in the size range given in Table 8, the "sweet spot" for ϵ in BASIS PURSUIT is around 0.1.

ϵ	$\tilde{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
			quantreg-5000			
0.20	0.0000	1.0402	0.0001	0.0000	0.0462	0.19
0.30	0.0000	1.0402	0.0001	0.0000	0.0202	0.15
0.40	0.0000	1.0402	0.0001	0.0000	0.0110	0.14
0.50	0.0000	1.0402	0.0001	0.0000	0.0072	0.14
0.60	0.0000	1.0402	0.0001	0.0000	0.0054	0.14
0.70	0.0000	1.0402	0.0001	0.0000	0.0036	0.16
0.80	0.0000	1.0402	0.0001	0.0000	0.0036	0.14
0.90	0.0000	1.0402	0.0001	0.0000	0.0018	0.14
			quantreg-6000			
0.20	0.0000	1.0373	0.0001	0.0001	0.0392	0.18
0.30	0.0000	1.0373	0.0001	0.0001	0.0172	0.14
0.40	0.0000	1.0374	0.0001	0.0001	0.0093	0.14
0.50	0.0000	1.0374	0.0001	0.0001	0.0062	0.13
0.60	0.0000	1.0373	0.0001	0.0001	0.0047	0.14
0.70	0.0000	1.0373	0.0001	0.0001	0.0030	0.13
0.80	0.0000	1.0373	0.0001	0.0001	0.0030	0.15
0.90	0.0000	1.0373	0.0001	0.0001	0.0015	0.13
			quantreg-7000			
0.20	0.0000	1.0276	0.0001	0.0001	0.0341	0.18
0.30	0.0000	1.0276	0.0001	0.0001	0.0150	0.14
0.40	0.0000	1.0276	0.0001	0.0001	0.0081	0.13
0.50	0.0000	1.0276	0.0001	0.0001	0.0054	0.13
0.60	0.0000	1.0276	0.0001	0.0001	0.0040	0.13
0.70	0.0000	1.0276	0.0001	0.0001	0.0027	0.14
0.80	0.0000	1.0276	0.0001	0.0001	0.0027	0.14
0.90	0.0000	1.0276	0.0001	0.0001	0.0013	0.14
			quantreg-8000			
0.20	0.0000	1.0236	0.0001	0.0001	0.0302	0.17
0.30	0.0000	1.0235	0.0001	0.0001	0.0132	0.13
0.40	0.0000	1.0236	0.0001	0.0001	0.0073	0.12
0.50	0.0000	1.0236	0.0001	0.0001	0.0047	0.16
0.60	0.0000	1.0236	0.0001	0.0001	0.0036	0.13
0.70	0.0000	1.0236	0.0001	0.0001	0.0024	0.12
0.80	0.0000	1.0236	0.0001	0.0001	0.0024	0.12
0.90	0.0000	1.0236	0.0001	0.0001	0.0011	0.12
			quantreg-9000			
0.20	0.0000	1.0212	0.0001	0.0000	0.0272	0.16
0.30	0.0000	1.0212	0.0001	0.0000	0.0120	0.13
0.40	0.0000	1.0212	0.0001	0.0000	0.0064	0.12
0.50	0.0000	1.0212	0.0001	0.0000	0.0043	0.12
0.60	0.0000	1.0212	0.0001	0.0000	0.0032	0.12
0.70	0.0000	1.0212	0.0001	0.0000	0.0021	0.12
0.80	0.0000	1.0212	0.0001	0.0000	0.0021	0.12
0.90	0.0000	1.0212	0.0001	0.0000	0.0010	0.12

Table 5. Quantile Regression results.

4.5 Comments on the retrieval phase improvement

In this section we give a few insights about the computational improvement of our new retrieval phase. These insights are based on the computational experience we gained by running the experiments many times over: this is due to removing occasional bugs (which entails a re-computation of all results), and to hands-on, sometimes

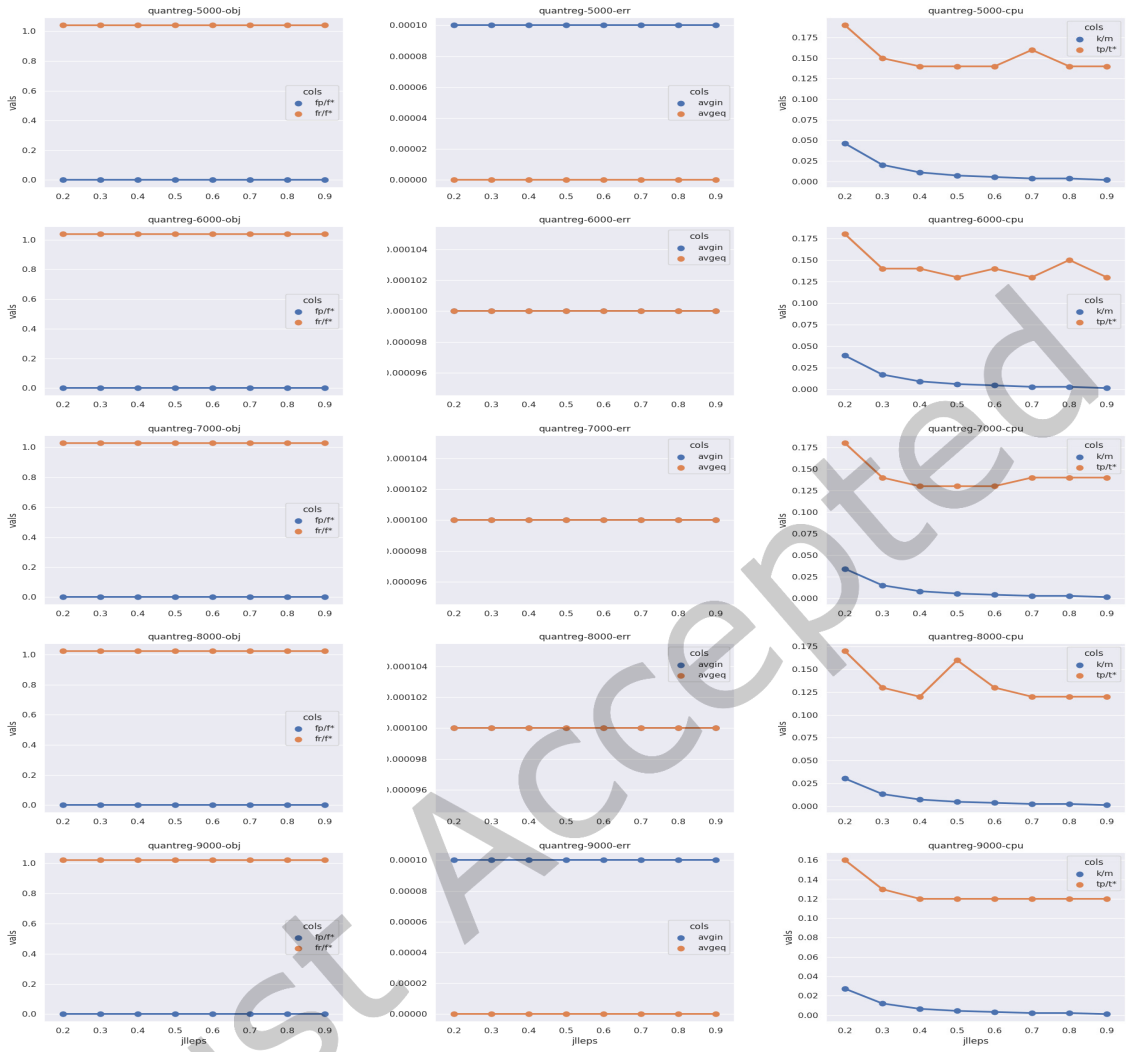

Fig. 4. Quantile Regression plots.

interactive, experiments with parts of the solution retrieval algorithm, i.e. the APM with bisection postprocessing described in Sect. 3.4-3.5.

The final code that gave rise to these experiments stems out of trade-offs and compromises. We do not present complete comparisons on all instances to motivate each of our algorithmic choices. This is because these choices influence each other, and a fully convincing set of computational experiments would need to include all possible combinations of all possible algorithic choices, which would give rise to an excessive amount of data, and an unreadable or even "un-writable" paper. Here, however, we give some details about what we think are our two most important choices.

ϵ	$\bar{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
quantreg-5000						
0.10	0.1026	1.1994	0.0001	0.0000	0.1848	100.86
0.15	0.0050	1.0527	0.0001	0.0000	0.0812	0.30
quantreg-6000						
0.10	0.0898	1.1803	0.0001	0.0001	0.1570	91.13
0.15	0.0070	1.0531	0.0001	0.0001	0.0690	0.31
quantreg-7000						
0.10	0.0804	1.1572	0.0001	0.0001	0.1367	76.19
0.15	0.0091	1.0489	0.0001	0.0001	0.0601	0.31
quantreg-8000						
0.10	0.0722	1.1418	0.0001	0.0001	0.1212	60.79
0.15	0.0080	1.0413	0.0001	0.0001	0.0534	0.26
quantreg-9000						
0.10	0.0682	1.1373	0.0000	0.0000	0.1091	49.42
0.15	0.0081	1.0390	0.0001	0.0000	0.0480	0.24

Table 6. Quantile Regression results with ϵ values in $\{0.1, 0.15\}$.

As mentioned at the end of Sect. 3.4, we chose Dykstra's APM variant in a set of three possibilities: von Neumann's original APM, Dykstra's, and the AARM. This choice was made by testing all three variants with a couple of instances for each of our four LP applications. Other possible choices might have been:

(i) the retrieval strategy based on the LP dual used to estimate a basis of the original primal LP [63];
(ii) the retrieval strategy based on a single projection of the projected solution $\bar{x}$ on $Ax = b$ via the pseudoinverse [42];
(iii) running a few iterations of a local solver with $\bar{x}$ as the starting point.

Theoretically speaking, (i) is very elegant: most LP solvers, deployed on TLP, will provide a primal-dual pair $(\bar{x}, \bar{y})$, where $\bar{x}$ solves the primal projected LP in standard form $\min\{cx \mid TAx = Tb \wedge x \geq 0\}$ and $\bar{y}$ solves the corresponding dual LP in canonical form $\max\{yTb \mid yTA \leq c\}$. Specifically, since $\bar{y}(TA) \leq c$, by associativity we have $(\bar{y}T)A \leq c$, which means that $\bar{y}T$ is a solution of the original dual LP $\max\{yb \mid yA \leq c\}$. In order to reconstruct the corresponding primal, however, we need a basic feasible solution (bfs), which $\bar{y}T$ unfortunately fails to be with probability 1. In practice, we tried estimating a basis by choosing the smallest components of $\bar{y}T$, but this yielded the unsatisfactory results of [63, col. "neg1" in Table 3] in terms of errors w.r.t. $x \geq 0$.

As for (ii), this is simply the first half-iteration of any APM used for retrieval purposes. The main issue with this choice, which was already apparent in [63, col. "neg2" in Table 3], is that the retrieved solution is perfectly feasible w.r.t. $Ax = b$, but infeasible w.r.t. $x \geq 0$. In fact, this is the reason that motivated us to add a further projection on $x \geq 0$, which unfortunately offset the feasibility w.r.t. $Ax = b$, whence the idea of the APM. So, more than a possible choice, this is actually our motivation to use an APM as a solution retrieval method.

Finally, we comment on (iii). Our tests with the generic local solver `scipy.optimize.minimize` were catastrophic in terms of CPU time. While we hoped to be able to test the more appropriate `scipy.optimize.linprog`, this was not possible: as explained in `docs.scipy.org`, the argument "`x0=None`" is "currently used only by the 'revised simplex' method, and can only be used if `x0` represents a basic feasible solution". This meant that we could not use the HiGHS [24] IPM, since its `scipy` interface does not accept a starting point, and we could not use the HiGHS simplex method because our starting point is not a bfs. We looked for another good quality IPM for LP with a reasonably easy interface to Python, but we could not find one.

ϵ	$\tilde{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\tilde{t}/t^*$
		basispursuit-1000				
0.20	1.0000	1.0000	0.0000	0.0000	0.1900	0.12
0.30	0.8254	4.2269	0.0000	0.0000	0.0830	0.05
0.40	0.6111	5.1504	0.0000	0.0000	0.0450	0.04
0.50	0.4694	5.0960	0.0000	0.0000	0.0300	0.03
0.60	0.3361	5.1374	0.0000	0.0000	0.0220	0.03
0.70	0.3237	5.3041	0.0000	0.0000	0.0150	0.03
0.80	0.3448	5.1774	0.0000	0.0000	0.0150	0.03
0.90	0.2002	5.1665	0.0000	0.0000	0.0070	0.03
		basispursuit-2000				
0.20	1.0000	1.0000	0.0000	0.0000	0.1000	0.04
0.30	0.7329	4.8814	0.0000	0.0000	0.0440	0.02
0.40	0.5218	5.2099	0.0000	0.0000	0.0240	0.02
0.50	0.3972	5.2238	0.0000	0.0000	0.0160	0.01
0.60	0.3065	5.2564	0.0000	0.0000	0.0120	0.01
0.70	0.2849	5.3341	0.0000	0.0000	0.0080	0.01
0.80	0.2734	5.3472	0.0000	0.0000	0.0080	0.01
0.90	0.1820	5.1992	0.0000	0.0000	0.0040	0.01
		basispursuit-3000				
0.20	1.0000	1.0000	0.0000	0.0000	0.0690	0.04
0.30	0.7012	5.0074	0.0000	0.0000	0.0303	0.02
0.40	0.5216	5.3400	0.0000	0.0000	0.0163	0.02
0.50	0.3314	5.3262	0.0000	0.0000	0.0110	0.01
0.60	0.3110	5.3398	0.0000	0.0000	0.0080	0.01
0.70	0.2700	5.4954	0.0000	0.0000	0.0053	0.01
0.80	0.2293	5.3591	0.0000	0.0000	0.0053	0.01
0.90	0.1483	5.2518	0.0000	0.0000	0.0027	0.01
		basispursuit-4000				
0.20	0.9009	3.6023	0.0000	0.0000	0.0530	0.02
0.30	0.6015	4.6420	0.0000	0.0000	0.0232	0.01
0.40	0.4004	4.6775	0.0000	0.0000	0.0127	0.01
0.50	0.3239	4.7214	0.0000	0.0000	0.0085	0.01
0.60	0.2969	4.7464	0.0000	0.0000	0.0063	0.01
0.70	0.2222	4.7108	0.0000	0.0000	0.0043	0.01
0.80	0.1811	4.6105	0.0000	0.0000	0.0043	0.01
0.90	0.1186	4.5642	0.0000	0.0000	0.0020	0.01
		basispursuit-5000				
0.20	0.8428	3.6947	0.0000	0.0000	0.0434	0.02
0.30	0.5196	4.4183	0.0000	0.0000	0.0190	0.01
0.40	0.3724	4.4592	0.0000	0.0000	0.0104	0.01
0.50	0.2866	4.4232	0.0000	0.0000	0.0068	0.01
0.60	0.2166	4.3016	0.0000	0.0000	0.0052	0.01
0.70	0.1592	4.3098	0.0000	0.0000	0.0034	0.01
0.80	0.1807	4.4115	0.0000	0.0000	0.0034	0.01
0.90	0.0883	4.2447	0.0000	0.0000	0.0016	0.01

Table 7. Basis Pursuit results.

One further point worth discussing is the CPU time bottleneck when exploiting RPs to solve LPs. This consists of three distinct phases: sampling the RP, constructing and solving the projected LP, and performing solution retrieval. In order to avoid cluttering our results table with too many data, we preferred to just showcase the ratio of $\tilde{t}/t^*$, where $\tilde{t}$ is the sum of the CPU times of these three phases. On the other hand, this approach hides the breakdown of $\tilde{t}$ into sampling, solving, and retrieving.

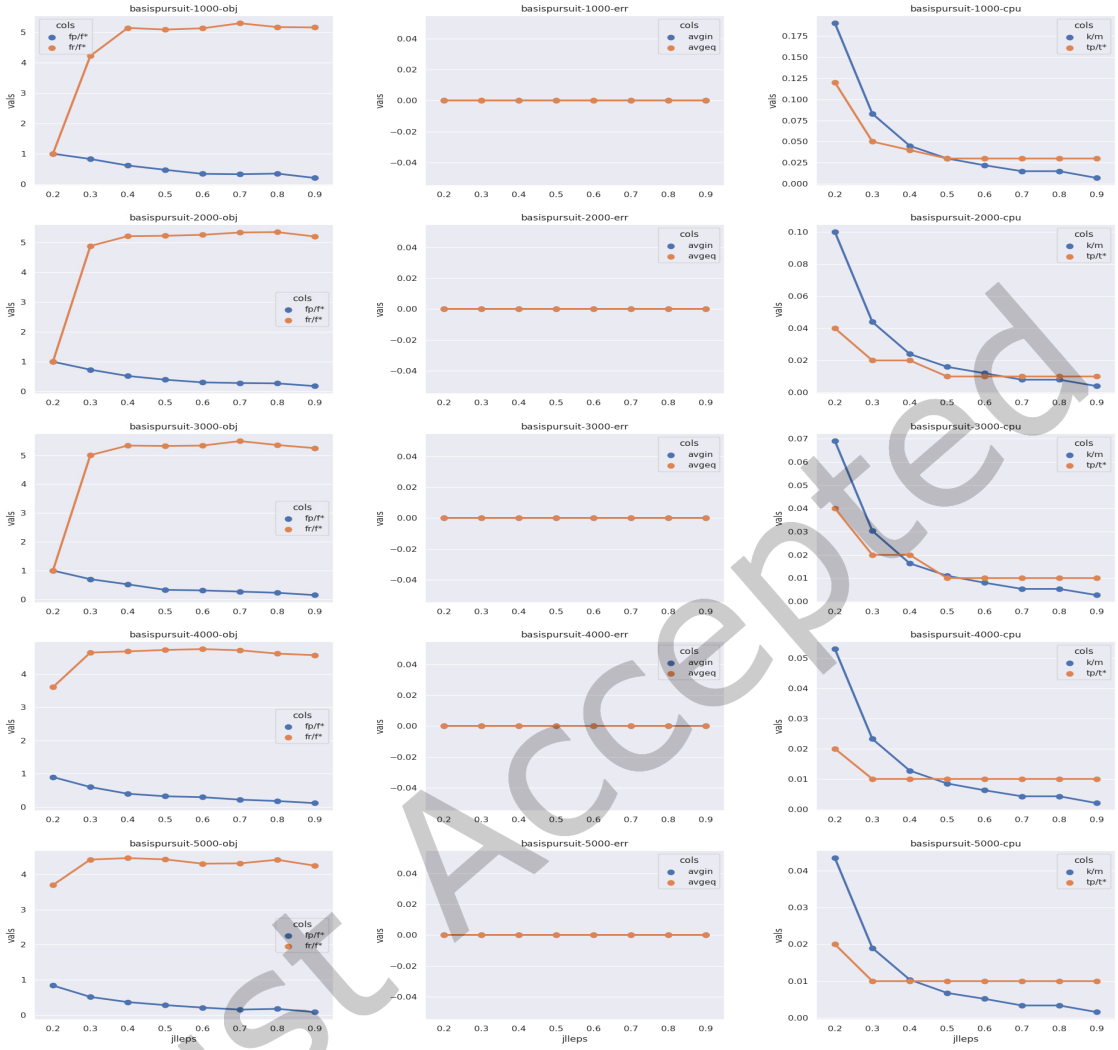

Fig. 5. Basis Pursuit plots.

In Table 9 we look at the total averages over all instances and runs of the CPU times of each phase. It turns out that sampling the RP was fastest, solution retrieval was ranked second, and solving the projected LP took longest. The sampling phase is by far the simplest algorithm out of the three: it is therefore natural that it should be fastest.

5 CONCLUSION

This paper studies the practical applicability to the theory of random projections to linear programming problems with certain given structures: maximum flow, diet, quantile regression, basis pursuit: the last three naturally rather dense, the first naturally sparse.

ϵ	$\bar{f}/f^*$	$\tilde{f}/f^*$	avgin	avgeq	k/m	$\bar{t}/t^*$
			basispursuit-1000			
0.05	40.8465	25528.2439	0.0000	0.0000	3.0400	5.01
0.10	1.0000	1.0000	0.0000	0.0000	0.7600	0.67
0.15	1.0000	1.0000	0.0000	0.0000	0.3340	0.18
0.20	1.0000	1.0000	0.0000	0.0000	0.1900	0.12
			basispursuit-2000			
0.05	3110.1331	3110.1331	0.0000	0.0000	1.6010	2.52
0.10	1.0000	1.0000	0.0000	0.0000	0.4000	0.19
0.15	1.0000	1.0000	0.0000	0.0000	0.1760	0.06
0.20	1.0000	1.0000	0.0000	0.0000	0.1000	0.04
			basispursuit-3000			
0.05	7.5265	7.5265	0.0000	0.0000	1.1057	3.13
0.10	1.0000	1.0000	0.0000	0.0000	0.2763	0.14
0.15	1.0000	1.0000	0.0000	0.0000	0.1213	0.05
0.20	1.0000	1.0000	0.0000	0.0000	0.0690	0.03
			basispursuit-4000			
0.05	1.0000	1.0000	0.0000	0.0000	0.8515	1.88
0.10	1.0000	1.0000	0.0000	0.0000	0.2127	0.09
0.15	1.0000	1.0000	0.0000	0.0000	0.0935	0.04
0.20	0.9179	3.6021	0.0000	0.0000	0.0530	0.02
			basispursuit-5000			
0.05	1.0000	1.0000	0.0000	0.0000	0.6958	1.15
0.10	1.0000	1.0000	0.0000	0.0000	0.1738	0.08
0.15	0.9979	1.1646	0.0000	0.0000	0.0764	0.03
0.20	0.8054	3.9630	0.0000	0.0000	0.0434	0.02

Table 8. Basis Pursuit results with ϵ values in $\{0.05, 0.1, 0.15, 0.2\}$.

	sampling	solving	retrieval
mean	1.13	42.97	7.00
standard deviation	3.84	214.43	15.89

Table 9. Breakdown of $\tilde{t}$ into sampling, projected problem solving, and solution retrieval.

A randomly projected linear program is obtained from the standard form by replacing the original equality constraints with a version where each of the few constraints is a random aggregation of all the many original constraints, with weights randomly sampled from a subgaussian distribution. This smaller linear program is solved using any solver, which yields a projected optimal solution having a value which is a relaxation of of the original problem. The projected solution is normally infeasible w.r.t. the original problem, so a feasible solution is obtained by a solution retrieval methodology.

The investigation conducted in this paper tightened error bounds on the feasibility error of the retrieved solution, and improved the existing retrieval methodology using an alternating projection method. We performed a computational test on reasonably large-scale instances of the four linear programming structures mentioned above. The CPU time taken to obtain an almost feasible solution of the LP using random projections was always less than the time taken by CPLEX to solve the LP to optimality, while the solution quality varied, yielding best results on the quantile regression problem.

REFERENCES

[1] D. Achlioptas. 2003. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. System Sci.* 66 (2003), 671–687.

[2] F. Alizadeh. 2012. An Introduction to Formally Real Jordan Algebras and Their Applications in Optimization. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, M. Anjos and J. Lasserre (Eds.). Operations Research & Management Science, Vol. 166. Springer, Boston, MA.

[3] D. Amelunxen, M. Lotz, M. McCoy, and J. Tropp. 2014. Living on the edge: phase transitions in convex programs with random data. *Information and Inference: A Journal of the IMA* 3 (2014), 224–294.

[4] F. Aragón Artacho and R. Campoy. 2018. A new projection method for finding the closest point in the intersection of convex sets. *Computational Optimization and Applications* 69 (2018), 99–132.

[5] A. Barvinok. 1997. Measure concentration in optimization. *Mathematical Programming* 79 (1997), 33–53.

[6] R. Bixby. 1992. Implementing the simplex method: The initial basis. *ORSA Journal on Computing* 4, 3 (1992), 267–284.

[7] Andreas Bluhm and Daniel Stilck França. 2019. Dimensionality reduction of SDPs through sketching. *Linear Algebra Appl.* 563 (2019), 461–475.

[8] C. Boutsidis, A. Zouzias, and P. Drineas. 2010. Random projections for k-means clustering. In *Advances in Neural Information Processing Systems (NIPS)*. NIPS Foundation, La Jolla, 298–306.

[9] E. Candès and T. Tao. 2005. Decoding by Linear Programming. *IEEE Transactions on Information Theory* 51, 12 (2005), 4203–4215.

[10] C. Cartis, E. Massart, and A. Otemissov. 2022. Global optimization using random embeddings. *Mathematical Programming B* online (2022).

[11] A. Chowdhury, G. Dexter, P. London, H. Avron, and P. Drineas. 2022. Faster randomized interior point methods for tall/wide linear programs. *Journal of Machine Learning Research* 23 (2022), 1–48.

[12] L. Clarskon, P. Drineas, M. Magdon-Ismail, M. Mahoney, X. Meng, and D. Woodruff. 2013. The fast Cauchy transform and faster robust linear regression. In *Proceedings of the 24-th Symposium on Discrete Algorithms*. ACM, New York.

[13] M.B. Cohen, J. Nelson, and D.P. Woodruff. 2016. Optimal approximate matrix product in terms of stable rank. In *43rd International Colloquium on Automata, Languages, and Programming (LIPIcs, Vol. 55)*, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi (Eds.). Dagstuhl Publishing, Saarbrücken, 11:1–11:14.

[14] P. Combettes and J-C. Pesquet. 2011. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, H. Bauschke, R. Burachik, P. Combettes, V. Elser, D. Russel Luke, and H. Wolkowicz (Eds.). Optimization and Its Applications, Vol. 49. Springer, New York, 185–212.

[15] C. D'Ambrosio, L. Liberti, P.-L. Poirion, and K. Vu. 2020. Random projections for quadratic programs. *Mathematical Programming B* 183 (2020), 619–647.

[16] S. Damelin and W. Miller. 2012. *The mathematics of signal processing*. CUP, Cambridge.

[17] G.B. Dantzig. 1963. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ.

[18] F. Deutsch and H. Hundal. 1997. The rate of convergence for the method of alternating projections, II. *J. Math. Anal. Appl.* 205 (1997), 381–405.

[19] S. Dirksen. 2016. Dimensionality reduction with subgaussian matrices: A unified theory. *Foundations of Computational Mathematics* 16 (2016), 1367–1396.

[20] R. Dykstra. 1983. An algorithm for restricted least squares regression. *Journal of the American Statistical Asociation* 78, 384 (1983), 837–842.

[21] R. Fourer and D. Gay. 2002. *The AMPL Book*. Duxbury Press, Pacific Grove.

[22] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[23] I. Halperin. 1962. THe product of projection operators. *Acta Scientiarum Mathematicarum (Szeged)* 23 (1962), 96–99.

[24] Q. Huangfu and J.A.J. Hall. 2018. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (2018), 119–142.

[25] IBM. 2020. *ILOG CPLEX 20.1 User's Manual*. IBM.

[26] IBM. 2022. *ILOG CPLEX 22.1 User's Manual*. IBM.

[27] P. Indyk. 2001. Algorithmic applications of low-distortion geometric embeddings. In *Foundations of Computer Science (FOCS, Vol. 42)*. IEEE, Washington, DC, 10–33.

[28] P. Indyk and J. Matoušek. 2004. Low-distortion embeddings of finite metric spaces. In *Handbook of Discrete and Computational Geometry*, J. Goodman and J. O'Rourke (Eds.). Chapman and Hall, Boca Raton.

[29] P. Indyk and R. Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Symposium on the Theory Of Computing (STOC, Vol. 30)*. ACM, New York, 604–613.

[30] P. Indyk and A. Naor. 2007. Nearest neighbor preserving embeddings. *ACM Transactions on Algorithms* 3, 3 (2007), Art. 31.

[31] W. Johnson and J. Lindenstrauss. 1984. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability (Contemporary Mathematics, Vol. 26)*, G. Hedlund (Ed.). AMS, Providence, RI, 189–206.
[32] D. Kane and J. Nelson. 2014. Sparser Johnson-Lindenstrauss transforms. *J. ACM* 61, 1 (2014), 4.
[33] N. Karmarkar. 1984. A new polynomial time algorithm for linear programming. *Combinatorica* 4, 4 (1984), 373–395.
[34] J. Lee. 1989. Turán's triangle theorem and binary matroids. *European Journal of Combinatorics* 10 (1989), 85–90.
[35] L. Liberti. 2009. Reformulations in Mathematical Programming: Definitions and Systematics. *RAIRO-RO* 43, 1 (2009), 55–86.
[36] L. Liberti. 220. Distance Geometry and Data Science. *TOP* 28 (220), 271–339.
[37] L. Liberti. pending minor revisions. Decoding noisy messages: a method that just shouldn't work. In *Data Science and Optimization*, A. Deza, S. Gupta, and S. Pokutta (Eds.). Fields Institute, Toronto.
[38] L. Liberti and B. Manca. 2022. Side-constrained minimum sum-of-squares clustering: Mathematical programming and random projections. *Journal of Global Optimization* 83 (2022), 83–118.
[39] L. Liberti, B. Manca, and P.-L. Poirion. 2022. Practical performance of Random Projections in Linear Programming. In *Proceedings of the 20th International Symposium on Experimental Algorithms (SEA22) (LIPIcs, Vol. 233)*, C. Schulz and B. Uçar (Eds.). Dagstuhl Publishing, Saarbrücken.
[40] L. Liberti, B. Manca, and P.-L. Poirion. 2022. Random projections for semidefinite programming. In *Proceedings of the AIRO-ODS Conference (AIRO Series)*, P. Cappanera et al. (Ed.). Springer, Cham.
[41] L. Liberti, B. Manca, and P.-L. Poirion. 2022. Random projections for the distance geometry problem. In *Proceedings of the workshop Discrete Mathematics Days*, M. Noy et al. (Ed.). Universidad de Cantabria, Santander.
[42] L. Liberti, P.-L. Poirion, and K. Vu. 2021. Random projections for conic programs. *Linear Algebra Appl.* 626 (2021), 204–220.
[43] P.-G. Martinsson and J.A. Tropp. 2020. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica* 29 (2020), 403–572.
[44] J. Matousek. 1996. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics* 93 (1996), 333–344.
[45] J. Matoušek. 2008. On variants of the Johnson-Lindenstrauss lemma. *Random Structures and Algorithms* 33 (2008), 142–156.
[46] J. Matoušek. 2013. *Lecture notes on metric embeddings.* Technical Report. ETH Zürich.
[47] M. Pilanci and M. Wainwright. 2015. Randomized sketches of convex programs with sharp guarantees. *IEEE Transactions on Information Theory* 61, 9 (2015), 5096–5115.
[48] M. Pilanci and M. Wainwright. 2017. Newton sketch: A linear time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization* 27, 1 (2017), 205–245.
[49] P.-L. Poirion, B.F. Lourenço, and A. Takeda. 2020. *Random projections of linear and semidefinite problems with linear inequalities.* Technical Report 2007.00242. arXiv.
[50] D. Pucci de Farias and B. Van Roy. 2004. On constraint sampling in the Linear Programming approach to approximate Dynamic Programming. *Mathematics of Operations Research* 29, 3 (2004), 462–478.
[51] J. Renegar. 1988. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming* 40 (1988), 59–93.
[52] F. Roosta-Khorasani and M.W. Mahoney. 2019. Sub-sampled Newton methods. *Mathematical Programming* 174, 1 (2019), 293–326.
[53] Z. Song and Z. Yu. 2020. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM, New York, 775–788.
[54] Z. Song and Z. Yu. 2021. Oblivious sketching-based central path method for linear programming. In *Proceedings of the 38th International Conference on Machine Learning (PMLR, Vol. 139)*. 9835–9847.
[55] J. Tropp, A. Yurtsever, M. Udell, and V. Cevher. 2017. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal of Matrix Analysis and Applications* 38, 4 (2017), 1454–1485.
[56] G. van Rossum and *et al.* 2019. *Python Language Reference, version 3.* Python Software Foundation.
[57] S. Vempala. 2004. *The Random Projection Method.* Number 65 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS, Providence, RI.
[58] S. Venkatasubramanian and Q. Wang. 2011. The Johnson-Lindenstrauss Transform: An Empirical Study. In *Algorithm Engineering and Experiments (ALENEX, Vol. 13)*. SIAM, Providence, RI, 164–173.
[59] R. Vershynin. 2018. *High-dimensional probability.* CUP, Cambridge.
[60] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2
[61] J. von Neumann. 1950. *Functional Operators. Volume II: The geometry of orthogonal spaces.* Number 22 in Annals of Mathematics Studies. Princeton University Press, Princeton NJ.

[62] K. Vu, P.-L. Poirion, C. D'Ambrosio, and L. Liberti. 2019. Random projections for quadratic programs over a Euclidean ball. In *Integer Programming and Combinatorial Optimization (IPCO) (LNCS, Vol. 11480)*, A. Lodi and *et al.* (Eds.). Springer, New York, 442–452.
[63] K. Vu, P.-L. Poirion, and L. Liberti. 2018. Random projections for linear programming. *Mathematics of Operations Research* 43, 4 (2018), 1051–1071.
[64] K. Vu, P.-L. Poirion, and L. Liberti. 2019. Gaussian random projections for Euclidean membership problems. *Discrete Applied Mathematics* 253 (2019), 93–102.
[65] D. Woodruff. 2014. Sketching as a tool for linear algebra. *Foundations and Trends in Theoretical Computer Science* 10, 1-2 (2014), 1–157.
[66] J. Yang, X. Meng, and M. Mahoney. 2014. Quantile regression for large-scale applications. *SIAM Journal of Scientific Computing* 36, 5 (2014), S78–S110.
[67] L. Zhang, M. Mahdavi, R. Jin, T. Yang, and S. Zhu. 2013. Recovering the Optimal Solution by Dual Random Projection.. In *Conference on Learning Theory (COLT) (Proceedings of Machine Learning Research, Vol. 30)*, S. Shalev-Shwartz and I. Steinwart (Eds.). ⟨jmlr.org⟩, 135–157.