Practical Performance of Random Projections in Linear Programming

Leo Liberti
LIX CNRS, Ecole Polytechnique, Institut Polytechnique de Paris, 91128 Palaiseau, France

Benedetto Manca
Department of Mathematics and Informatics, University of Cagliari, Italy

Pierre-Louis Poirion
RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

Abstract

The use of random projections in mathematical programming allows standard solution algorithms to solve instances of much larger sizes, at least approximately. Approximation results have been derived in the relevant literature for many specific problems, as well as for several mathematical programming subclasses. Despite the theoretical developments, it is not always clear that random projections are actually useful in solving mathematical programs in practice. In this paper we provide a computational assessment of the application of random projections to linear programming.

2012 ACM Subject Classification Mathematics of computing → Mathematical optimization; Theory of computation → Random projections and metric embeddings

Keywords and phrases Linear Programming, Johnson-Lindenstrauss Lemma, Computational testing

Digital Object Identifier 10.4230/LIPIcs.SEA.2022.21

Supplementary Material *Software (Source Code)*:
`https://mega.nz/file/p8MQhbpT#0TJBUVgaBf4KPVk2fu_5k05cMy2VozJk-0fQ1PZdJ0U`

Funding *Benedetto Manca*: Partly supported by grant STAGE, Fondazione Sardegna 2018.

1 Introduction

This paper is about applying Random Projections (RP) to Linear Programming (LP) formulations. RPs are dimensional reduction operators that usually apply to data. The point of applying RPs to LPs is to obtain an approximate solution of the high-dimensional formulation by solving a related lower-dimensional one. The main goal of this paper is to discuss the pros and cons of this technique from a computational (practical) point of view.

1.1 Random Projections

In general, RPs are functions, sampled randomly from certain distributions, that map a vector in $\mathbb{R}^m$ to one in $\mathbb{R}^k$, where $k \ll m$. In this paper we restrict our attention to linear RPs, which are $k \times m$ random matrices T. The most famous result about RPs is the Johnson-Lindenstrauss Lemma [13], which we recall here in its probabilistic form. Given a finite set $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^m$ and an $\epsilon \in (0,1)$, there exists a $\delta = O(e^{-\mathcal{C}\phi(k)})$ (with ϕ usually linear and $\mathcal{C}$ a universal constant not depending on input data) and an RP T with $k = O(\epsilon^{-2} \ln n)$ such that

$$\mathbf{Prob}\big(\forall i < j \le n \ (1-\epsilon)\|x_i - x_j\|_2 \le \|Tx_i - Tx_j\|_2 \le (1+\epsilon)\|x_i - x_j\|_2\big) \ge 1 - \delta. \quad (1)$$

If T is sampled componentwise from the normal distribution $\mathsf{N}(0, 1/\sqrt{k})$, Eq. (1) holds (note that other distributions also work). The JLL is not the only result worth mentioning in RP [22, 11, 19], but it is the object of interest in this paper.

20th International Symposium on Experimental Algorithms (SEA 2022).
Editors: Christian Schulz and Bora Uçar; Article No. 21; pp. 21:1–21:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The JLL directly applies to all problems involving the Euclidean distance between points in a Euclidean space of high dimension, e.g. the design of an efficient nearest-neighbor data structure (i.e. given $X \subset \mathbb{R}^m$ and $q \in \mathbb{R}^m$ quickly return $x \in X$ closest to q) [12].

More in general, the JLL shows that RPs can transform the point set X to a lower dimensional set TX such that X and TX are "approximately congruent": the pairwise distances in X are approximately the same (multiplicatively) as the corresponding pairwise distances in TX, even if X has m dimensions and TX only k (proportional to $\epsilon^{-2} \ln |X|$). Since "approximately congruence" means "almost the same, aside from translations, rotations, and reflections", it is reasonable to hope that RPs might apply to other constructs than just sets of points, and still deliver a theoretically quantifiable approximation. In this paper we consider LP.

1.2 Applying RPs to Linear Programming

In this paper we are interested in the application of the JLL to LP in standard form:

$$\left.\begin{array}{rrcl} \min\limits_{x} & c^\top x & & \\ & Ax & = & b \\ & x & \geq & 0, \end{array}\right\} \quad \text{(LP)}$$

where $x = (x_1, \ldots, x_n)$, A is an $m \times n$ matrix, and $b \in \mathbb{R}^m$.

There are several issues in applying RPs to Mathematical Programs (MP) in general. The three foremost are:

1. RPs project vectors rather than decision variables and constraint functions;
2. RPs ensure approximate congruence of the input vectors in the lower-dimensional output: but approximation arguments in LP must instead be based on optimality and feasibility (unrelated to the ℓ_2 norm);
3. RPs only apply to finite point sets, whereas LP decision variables represent infinite point sets.

These issues pose nontrivial theoretical challenges, and the proof techniques vary considerably depending on the MP subclass being considered. The first issue mentioned above is addressed by applying RPs to the problem parameters (the input data); in the LP case, we project the linear system $Ax = b$. We speak of the *original* formulation P and the *projected* formulation TP. This yields a fourth issue: the solution of TP may be infeasible in P: in such cases, a *solution retrieval* phase is necessary in order to construct a feasible solution of P from that of TP.

The second and third issues are addressed in [25], leading to statements similar to the JLL, but concerning approximate LP feasibility and optimality. If $E(P,T)$ is a statement about the feasibility or optimality error between the LP formulations P and TP, the general structure of these results is similar to the probabilistic version of the JLL:

$$\mathbf{Prob}\big(\, E(P,T)\,\big) \geq 1 - \delta, \tag{2}$$

where δ usually depends on ϵ, k and possibly even the solution of P. We shall recall the statements of these results more precisely in Sect. 2.

1.3 Relevant literature

The main reference for RPs and LP in standard form is [25], which presents the theory addressing the above issues, and a computational study focussing on dense random LP instances. RPs were also applied to some specific LP problems: PAC learning [20] and quantile regression [26], with dimensional reduction techniques tailored to the corresponding

LP structure. Other works in applying RPs to different types of MP subclasses are [24] (quadratic programs with a ball constraint), [4] (general quadratic programs), [18] (conic programs including second-order cone and semidefinite programs).

1.4 Contributions of this paper

Although some of the relevant literature carries computational results, we think that, computationally, the application of RPs to LPs is still experimental: in practice the output on a given instance can range from accurate all the way to catastrophic.

One of the difficulties is that, in writing $k = O(\epsilon^{-2} \ln n)$, we are neglecting a constant multiplicative coefficient C 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, no-one is able to justifiably foresee whether RPs will be useful or not on a given LP instance. The only existing work about practical RP usage is [23], which only focusses on computational testing of different RP matrices.

This paper will provide a computational analysis of LP cases where RPs work reasonably well, and others where they do not, and attempt to derive some guidelines for choosing appropriate values for the most critical unknown parameters. On the theoretical side, we tighten two results of [18] when applied to the LP case.

The rest of this paper is organized as follows. In Sect. 2 we recall the main theoretical results relative to the application of RPs to LP, and state the two new tightened results. In Sect. 3 we illustrate the benchmark goal, the LP structures we test, and the methodology. In Sect. 4 we discuss the benchmark results.

2 Summary of theoretical results

We apply RPs to the original formulation (LP) by reducing the number m of constraints. Let T be a $k \times m$ RP matrix. The projected formulation is:

$$\min\{c^\top x \mid TAx = Tb \wedge x \geq 0\} \quad (T\mathsf{LP}).$$

We first discuss feasibility. We note that the geometric interpretation of the feasible set $F = \{x \mid Ax = b \wedge x \geq 0\}$ of (LP) is that F is the set of conic combinations of the columns A^j of A, i.e. $F = \mathsf{cone}(A)$. We also let $\mathsf{conv}(A)$ the convex hull of the columns of A, and $\|x\|_A = \min\{\sum_j \lambda_j \mid x = \sum_j \lambda_j A^j\}$ be the A-*norm* of $x \in \mathsf{cone}(A)$. Is 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

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

is proved in [25, Thm. 3] for all $\epsilon \in (0, \Delta^2/(\mu_A + 2\mu_A\sqrt{1-\Delta^2}+1))$, where $\mathcal{C}$ is the universal constant of the JLL, $\mu_A = \max\{\|x\|_A \mid x \in \mathsf{cone}(A) \wedge \|x\|_2 \leq 1\}$, and Δ is a lower bound to $\min_{x \in \mathsf{conv}(A)} \|b - x\|_2$.

Let $\mathsf{val}(\cdot)$ indicate the optimal objective function value of a MP formulation. The approximate optimality statement for (LP) derived in [25, Thm. 4] is conditional to the LP formulation being feasible and bounded, so that, if x^* is an optimal solution, there is θ (assumed w.l.o.g. ≥ 1) such that $\sum_j x^*_j < \theta$. Given $\gamma \in (0, \mathsf{val}(\mathsf{LP}))$,

$$\mathbf{Prob}\big(\mathsf{val}(\mathsf{LP}) - \gamma \leq \mathsf{val}(T\mathsf{LP}) \leq \mathsf{val}(\mathsf{LP})\big) \geq 1 - \delta, \tag{4}$$

where $\delta = 4ne^{-\mathcal{C}(\epsilon^2-\epsilon^3)k}$, $\epsilon = O(\gamma/(\theta^2\|y^*\|_2))$, and y^* is an optimal dual solution of (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 [25, Prop. 3], it is proved that $\bar{x}$ is feasible in the original formulation with zero probability. We therefore need to provide a solution retrieval method. A couple were proposed in [25], but the one found in [18, Eq. (6)] comes with an approximation guarantee and a good practical performance. The retrieved solution $\tilde{x}$ is defined as the projection of $\bar{x}$ on the affine subspace $Ax = b$, and computed using the pseudoinverse:

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

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 in [25]. Let $\kappa(A)$ be the condition number of A; applying [25, Thm. 4.4] to LP, we obtain the following result, which bound the (negativity of) the smallest component of $\tilde{x}$ in terms of that of $\bar{x}$.

▶ **Proposition 1.** *There is a universal constant $\mathcal{C}_2$ such that, for any $u \geq 0$, we have:*

$$\mathbf{Prob}\big(\min_{j\le n}\tilde{x}_j \geq \min_{j\le n}\bar{x}_j - \epsilon\theta\kappa(A)(\mathcal{C}_2 + u\sqrt{2/\ln(n)})\big) \geq 1 - 2e^{-u^2}.$$

The proof is based on an improvement of [18, Eq. (7)] based on computing the Gaussian width and diameter of $\{x \geq 0 \mid \langle \mathbf{1}, x\rangle \leq 1\}$. As a corollary, we also have the following result about the difference between objective function values of the retrieved and projected solutions.

▶ **Corollary 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 $\mathcal{C}_2$ such that, for any $u \geq 0$, we have:*

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

3 What we establish and how

Upon receving an LP instance to be solved using RPs, one has to at least know how to decide k (the projected dimension) so that the solution of the projected formulation is reasonably close to that of the original one.

Ideally, one would like to estimate all unknown parameters: $k, \epsilon, C, \mathcal{C}$ in function of γ and δ. This is theoretically hopeless because the theoretical bounds derived for "all LPs" are far from tight. We shall see below that it is also computationally hopeless. In practice, moreover, one might be much more interested in finding a good retrieved solution (i.e. almost feasible in the original problem), rather than finding a good approximation to the optimal objective function value, since a feasible solution can be improved by local methods, while an approximate optimal value may at best be useful as an objective cut.

Our approach will accordingly be based on solving sets of uniformly sampled LP instances (from different applications) using a standard solver, and analyse the output in terms of how the feasibility and optimality errors of the retrieved solution vary with problem size and ϵ.

3.1 The RP matrix

All componentwise sampled sub-Gaussian distributions [7] can be used to ensure the results cited in this paper. Some sparse variants also exist, along the lines of [1, 15]. We use the sparse RPs described in [4, §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 $\sigma = d_A/2$, where d_A is the density of the constraint matrix A.

3.2 LP structures

We consider randomly generated LPs of the following four classes: Max Flow problems [8], Diet problems [6], Quantile Regression problems [16], and Basis Pursuit problems from sparse coding [3]. This choice yields a set of LP problems going from extremely sparse (Max Flow) to completely dense (Basis Pursuit), with the Diet and Quantile Regression providing cases of various intermediate densities. These four test cases arise from a diverse range of application settings: combinatorial optimization, continuous optimization, statistics, data science.

3.2.1 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:

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

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 [25, §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 $(i,j) \in \mathcal{A}$.

3.2.2 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:

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

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 $\hat{q} = \big(\max\limits_{i \leq m}(b_i/(\|D_i\|_0 D_{ij})) \mid j \leq n\big)$ is a feasible solution.

Again, (DP) is not in standard form, but the transformation is immediate using slack variables $r_i \geq 0$ for $i \leq m$. 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 $d_A = (d_D mn + m)/(m(n+m)) = (d_D n + 1)/(n+m)$.

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 $\sum_j x^*_j$.

3.2.3 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:

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

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 "skewed" version 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 to have the results in [25] 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. 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 $\sum_j x^*_j$.

3.2.4 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 [3]:

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

According to sparse coding theory [5], 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 [2], so it shouldn't really make sense to decrease m by using RPs, and yet some mileage can unexpectedly be extracted from this operation [17].

Similarly to (MF), in (BP) we can partition the constraints into equations $Ax = b$ and inequalities $-s \le x \le s$. Again by [25, §4.2], we devise a block-structured RP matrix which only projects the equations.

As in Sect. 3.2.3, (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.

3.3 Methodology

The goal of this paper is to provide a computational assessment of RPs applied to LP.

As discussed at the beginning of Sect. 3, the actual determination of all relevant parameters is theoretically hopeless. We can certainly simplify the task a little by noting that the coefficient C can be removed since it suffices to decide a value for ϵ in order to decide k. Ideally we would like to decide γ first (see Eq. (4)), then compute ϵ as $O(\gamma/(\theta^2\|y^*\|_2))$, and sample an appropriate RP. Unfortunately, estimating θ and $\|y^*\|_2$ prior to solving the original LP leads to tiny values for ϵ (e.g. 10^{-i} for $i \in \{2,\ldots,11\}$ in some preliminary tests), which would require the rows of A to be at least $O(10^{i^2})$ in order to yield a useful projection. Since we are interested in applying RPs to LPs with $O(10^2)$ and $O(10^3)$ rows, this "ideal" approach is inapplicable.

Instead, we repeatedly solve sets of instances of each LP structure. Each projected instance is solved with different values of $\epsilon \in \mathcal{E} = \{0.15, 0.2, 0.25, 0.3, 0.35, 0.4\}$ (these values have been found to be the most relevant in preliminary computational experiments performed over several years). Moreover, to mitigate the effect of randomness, we solve each instance with each ϵ multiple times. For each instance and ϵ we collect performance measures on objective function values, infeasibility errors, and CPU time. This allows us to illustrate the co-variability of ϵ and instance size with the performance measures.

4 The benchmark

The solution pipeline is based on Python 3 [21] and the libraries `scipy` [14] and `amplpy` [9] (besides other standard python libraries). For each problem type, we loop over instances (based on row size of the equality constraint system, varying in $\mathcal{S}$, see below), over $\epsilon \in \mathcal{E}$, and over 5 different runs for each instance and ϵ in order to amortize the result randomness depending on the choice of T. We solve all of the original and projected instances using CPLEX 20.1 [10]. We use the barrier solver, because we found this to be more efficient with large dense LPs than the simplex-based solvers in CPLEX. Our code can be downloaded here.[1] All tests have been carried out on a MacBook 2017 wih a 1.4GHz dual-core Intel Core i7 with 16GB RAM.

4.1 Choice of instances

In the case of Diet, Quantile Regression, and Basis Pursuit, we generated instances so that the number of rows of the equality constraint system $Ax = b$ is in the set $\mathcal{S} = \{500p \mid 1 \leq p \leq 5 \wedge p \in \mathbb{N}\}$. For Max Flow we used $\mathcal{S}' = \mathcal{S} \smallsetminus \{2500\}$ because the larger size triggered a RAM-related error in a part of the solution pipeline involving the AMPL [9] interpreter.

4.1.1 The variable space

The space of original, projected, and retrieved variable values is identical for Max Flow, Quantile Regression, and Basis Pursuit, since these three structures are originally cast in an equality constraint form $Ax = b$. This desirable property fails to hold for Diet, which deserves a separate discussion.

The original formulation (DP) of Diet is in inequality form $Dq \geq b$, but the projected formulation is derived from the constraints $Ax = b$ in standard form, where $A = (D| - I)$.

The theoretical results in Sect. 2 justify a fair comparison only between original and projected solutions in standard form. Since this paper is about a *practical* comparison, however, and since no-one would convert (DP) to standard form before solving it (because the solver would do it as needed), 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 of $\tilde{x}$.

Considering only the q variables is unproblematic if applied to the optimal solution x^* of the original formulation in standard form, because $s^* \geq 0$ 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 with both positive and negative entries (since T is sampled from a normal distribution). Thus, it often happens that the underdetermined $k \times m$ system $TI = Tb$ has solutions. 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 $D\tilde{q} \ngeq b$. The application of RPs to Diet is therefore less successful than for other structures.

[1] The URL is `https://mega.nz/file/p8MQhbpT#0TJBUVgaBf4KPVk2fu_5k05cMy2VozJk-0fQ1PZdJ0U`.

4.2 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 feasibility error w.r.t. equation constraints $Ax = b$ (`eq`) and inequalities $x \geq 0$ (`in`), the CPU time t^* taken to solve the original formulation, and the CPU time $\bar{t}$ taken to solve the projected formulation.

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

The benchmark considers: 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 size and ϵ value.

4.3 RP performance on Max Flow

The application of RPs to the MAX FLOW problem looks like a success story: the ratio of projected to original optimal objective function value is very close to 1.0 and constant w.r.t. ϵ ($\bar{f}/f^* \geq 1$ is normal insofar as MAX FLOW is a maximization problem, and TLP is a relaxation of LP). The feasibility error of the retrieved solution related to the equality constraints $Ax = b$ is very close to zero, and the error w.r.t. $x \geq 0$ decreases as m increases (a healthy behaviour in RPs) and also as ϵ increases (implying that maximum negativity error increases more slowly than the number of variables). The CPU time ratio decreases proportionally to k/m, as expected. The only issue is that the objective function value at the retrieved solution is only around 0.5 of the optimum.

4.4 RP performance on Diet

As mentioned in Sect. 4.1.1, the practical application of RPs to the DIET problem is not successful, as shown by the plots in Fig. 2. The projected cost is almost always zero, because the constraint projection allowed the solver to satisfy $(D|-I)(q,r)^\top = b$ using slack variables only. This causes sizable errors in the retrieved solutions. As expected, the CPU time taken to solve the projected formulation is a tiny fraction of the time to solve the original formulation.

We tried to experiment with a modified projected objective $(c\,|\,\mathbf{1})$ so that we would minimize the sum of the projected slack variables. This yielded quantitatively better results, as shown in Fig. 3; qualitatively, the results still look like a failure.

4.5 RP performance on Quantile Regression

The results quality on QUANTILE REGRESSION is mixed. The ratio $\bar{f}/f^*$ is rather low, but we note that it is higher (better) for low sizes and low ϵ values, which is a sign that ϵ should be further decreased for all (and specially large) sizes. Interestingly, the objective value of the retrieved solution $\tilde{x}$ has better quality. The feasibility errors of $\tilde{x}$ are zero for $Ax = b$, and not negligible (around 0.2, with one outlier) for $x \geq 0$: the trend, unfortunately, is not decreasing, either with ϵ or m increasing. CPU time ratios are good.

To see whether increasing sizes and decreasing ϵ improved performances, we solved an instance with $m = 5000$ and $p = 100$ with $\epsilon = 0.1$, obtaining the following results.

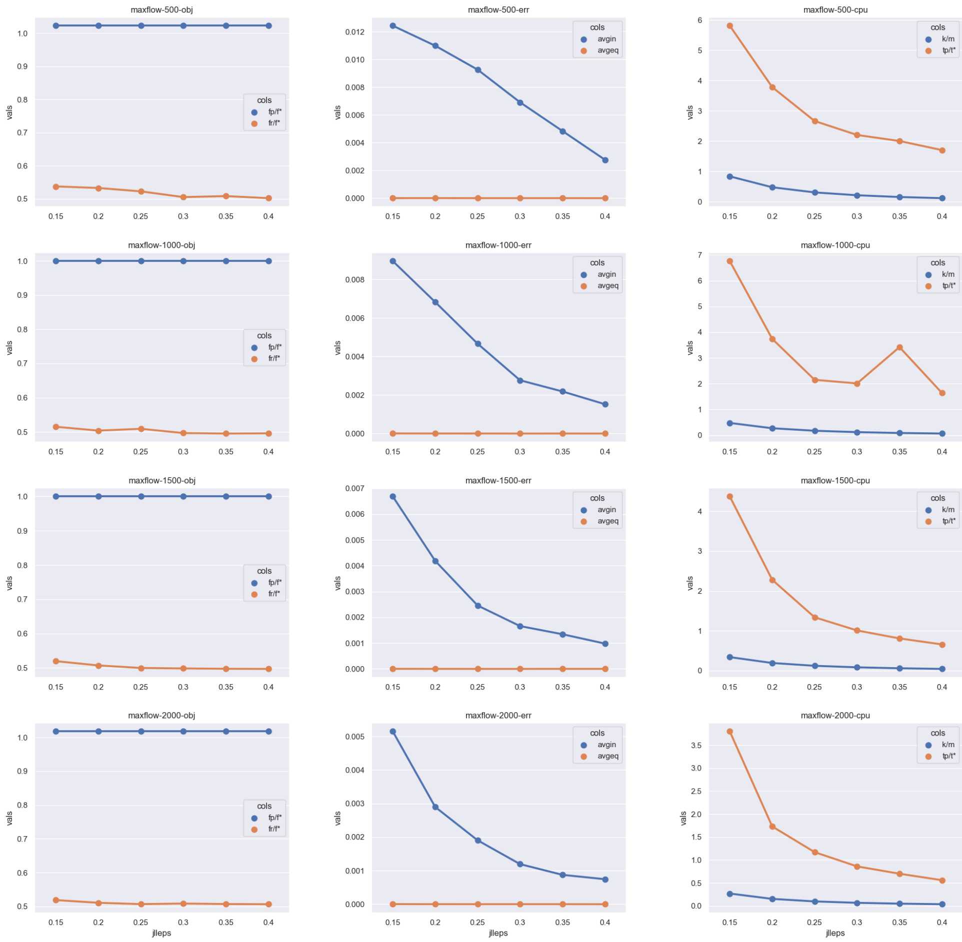


Figure 1 Max Flow plots (increasing ϵ on abscissae): instances of growing size on rows, objective function ratios on the first column, feasibility errors on the second, k/m and CPU time ratio on the third.

ϵ	$\bar{f}/f^*$	$\tilde{f}/f^*$	`avgin`	`avgeq`	k/m	$\bar{t}/t^*$
			quantreg-5000			
0.10	0.1460	0.3839	0.1784	0.0000	0.18	4.43

We can see that the objective function ratios of this instance provide a definite improvement with respect to the three largest instances in Fig. 4 ($m \in \{1500, 2000, 2500\}$). The negativity error is, however, of the same magnitude as before.

4.6 RP performance on Basis Pursuit

In the Basis Pursuit problem we see an encouraging trend of the ratio $\bar{f}/f^*$, which starts off at 0.8 for $m = 500$ and $\epsilon = 0.15$, and indicates that ϵ should be decreased for larger sizes. The retrieved solution was not computed on the "sandwich" variables s (see Eq. (BP)), but as the ℓ_1 norm of $\tilde{x}$. Since there are fewer constraints in the encoding matrix A, it follows from compressed sensing theory that the sparsest solution is found less often, a fact that increases the objective value of the retrieved solution. The feasibility errors are always zero

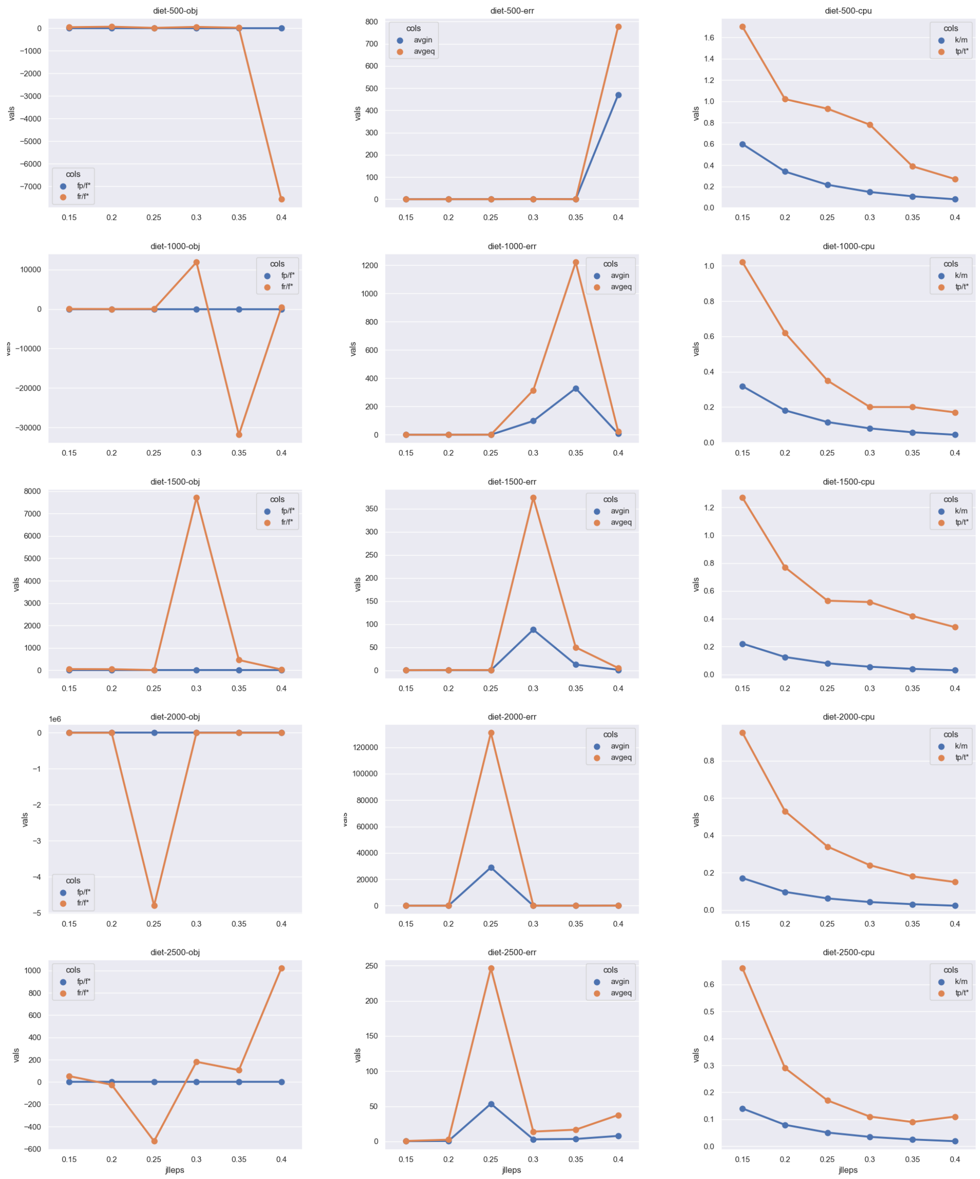


Figure 2 DIET plots (increasing ϵ on abscissae): instances of growing size on rows, objective function ratios on the first column, feasibility errors on the second, k/m and CPU time ratio on the third.

(for $Ax = b$ and $x \geq 0$), which happens because the variables x are unbounded. The CPU time ratio is not as regular as for the other structures, but still denotes a remarkable time saving when solving projected formulations.

To see whether increasing sizes and decreasing ϵ improved performances, we solved an instance with $m = 5000$ and $n = 6000$ with $\epsilon = 0.1$, obtaining the following results.

ϵ	$\bar{f}/f^*$	$\tilde{f}/f^*$	`avgin`	`avgeq`	k/m	$\bar{t}/t^*$
			basispursuit-5000			
0.10	0.4925	1.5395	0.0000	0.0000	0.17	0.09

An improvement with respect to the three largest instances in Fig. 5 ($m \in \{1500, 2000, 2500\}$) is present, which points to the correct trend, albeit not substantial.

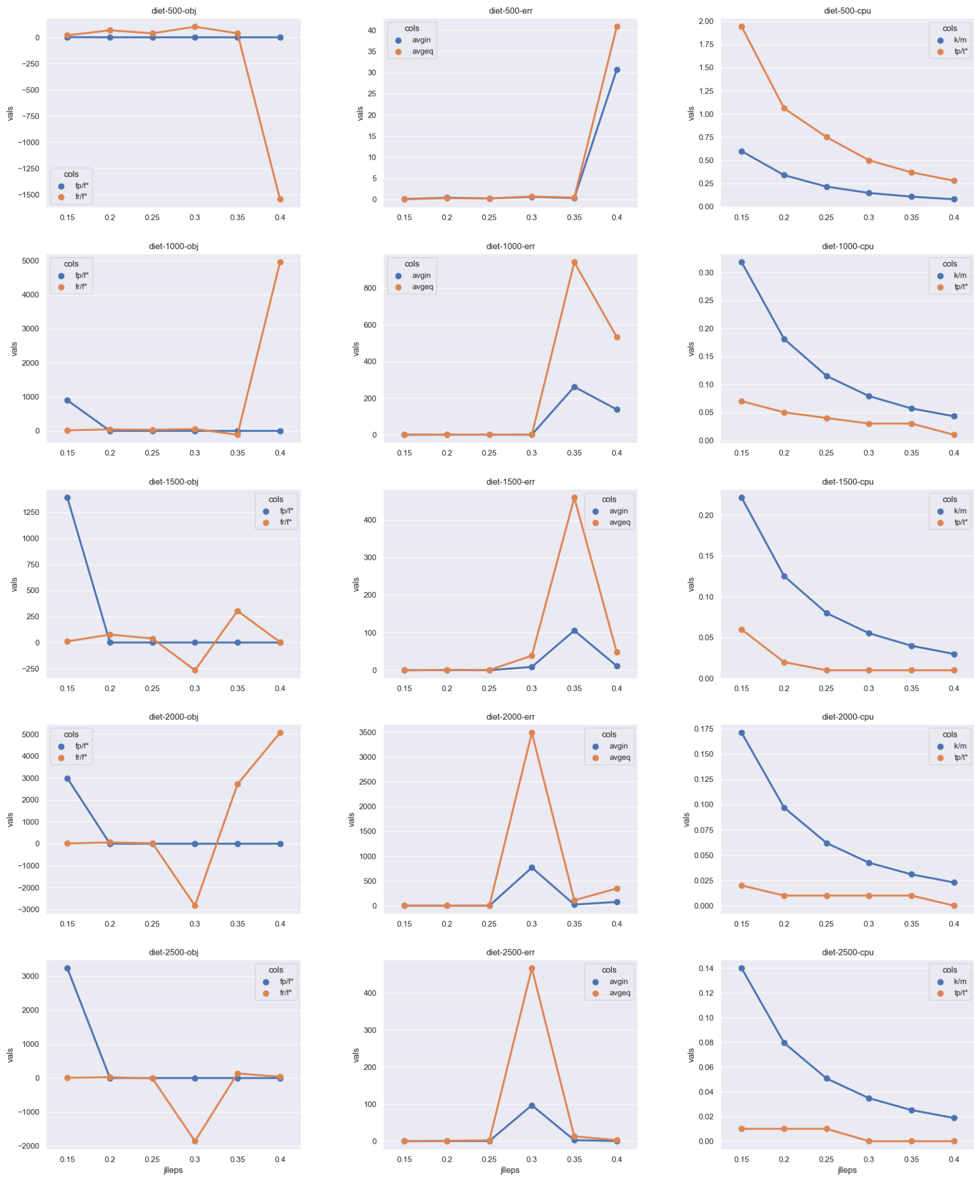
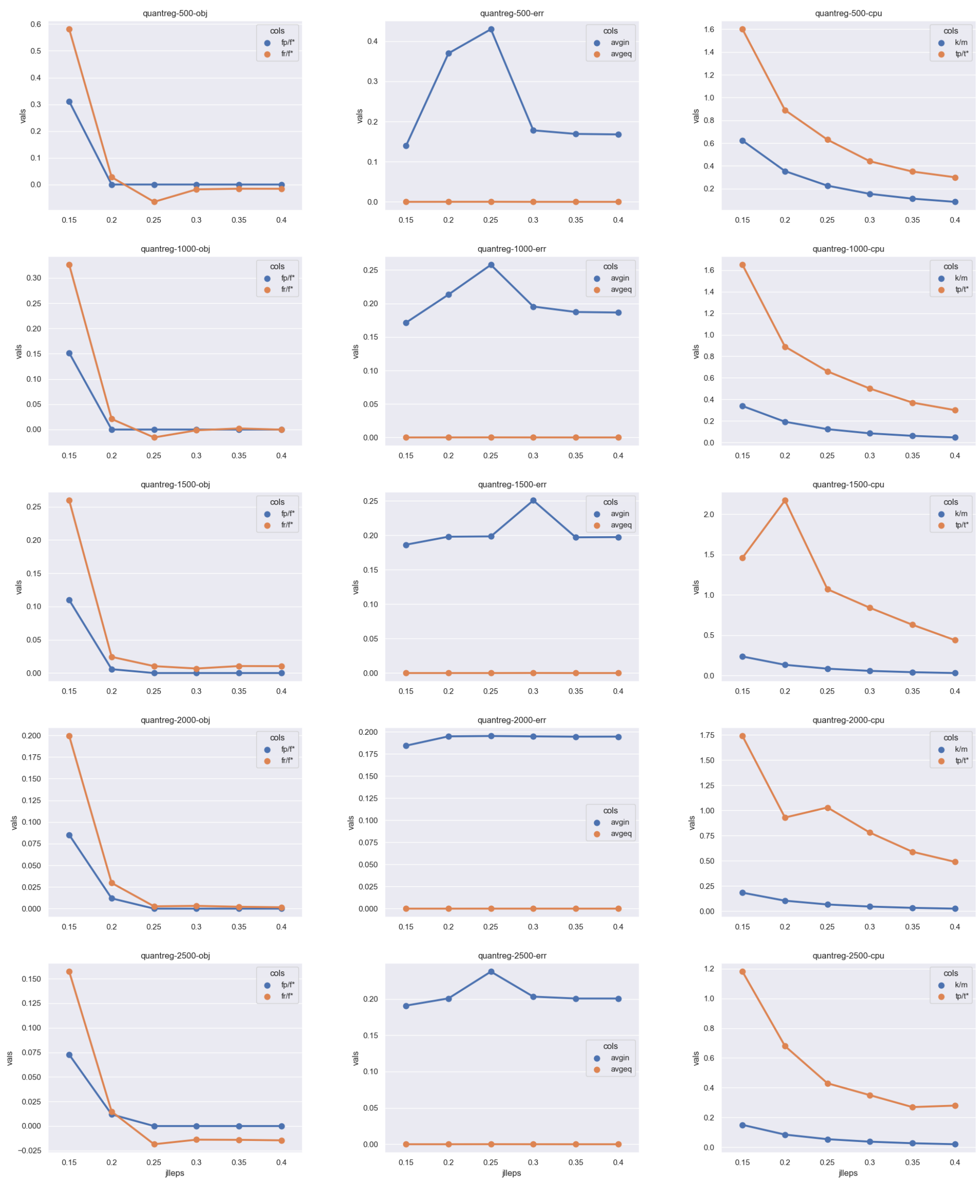


Figure 3 DIET plots with modified objective attempting to drive the slack variables to zero.

5 Conclusion

In this paper we have pursued a computational study of the application of random projections to linear program data, based on solving original and projected formulations linear program instances of various structures and sizes. We found that original formulations only involving inequalities are particularly challenging, but those that natively involve equations behave better. The sparsity of the constraint matrix does not appear to pose issues, as long as sparse RPs are used. Lastly, the sizes we considered here are possibly at the lower end of the range allowed by RPs: better results should be obtained with larger sizes and smaller values of ϵ, which in turn imply larger CPU times.



■ **Figure 4** QUANTILE REGRESSION plots (increasing ϵ on abscissae): instances of growing size on rows, objective function ratios on the first column, feasibility errors on the second, k/m and CPU time ratio on the third.

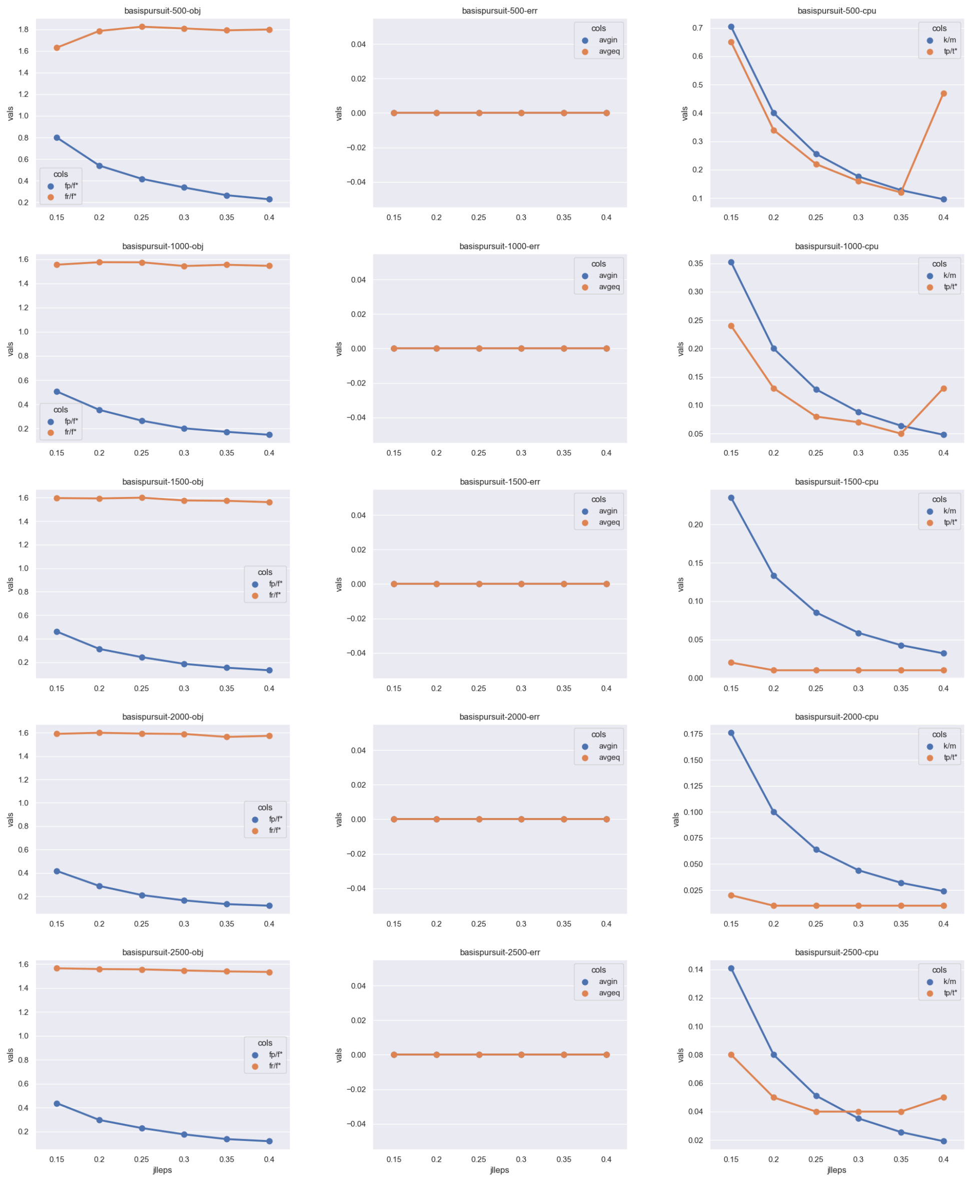


Figure 5 Basis Pursuit plots (increasing ϵ on abscissae): instances of growing size on rows, objective function ratios on the first column, feasibility errors on the second, k/m and CPU time ratio on the third.

References

1 D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66:671–687, 2003.

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

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

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

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

6 G. Dantzig. The Diet Problem. *Interfaces*, 20(4):43–47, 1990.

7 S. Dirksen. Dimensionality reduction with subgaussian matrices: A unified theory. *Foundations of Computational Mathematics*, 16:1367–1396, 2016.

8 L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.

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

10 IBM. *ILOG CPLEX 20.1 User's Manual.* IBM, 2020.

11 P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Foundations of Computer Science*, volume 42 of *FOCS*, pages 10–33, Washington, DC, 2001. IEEE.

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

13 W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In G. Hedlund, editor, *Conference in Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, Providence, RI, 1984. AMS.

14 E. Jones, T. Oliphant, and P. Peterson. `SciPy`: Open source scientific tools for Python, 2001. [Online; accessed 2016-03-01]. URL: `http://www.scipy.org/`.

15 D. Kane and J. Nelson. Sparser Johnson-Lindenstrauss transforms. *Journal of the ACM*, 61(1):4, 2014.

16 R. Koenker. *Quantile regression.* CUP, Cambridge, 2005.

17 L. Liberti. Decoding noisy messages: a method that just shouldn't work. In A. Deza, S. Gupta, and S. Pokutta, editors, *Data Science and Optimization.* Fields Institute, Toronto, pending minor revisions.

18 L. Liberti, P.-L. Poirion, and K. Vu. Random projections for conic programs. *Linear Algebra and its Applications*, 626:204–220, 2021.

19 L. Liberti and K. Vu. Barvinok's naive algorithm in distance geometry. *Operations Research Letters*, 46:476–481, 2018.

20 D. Pucci de Farias and B. Van Roy. On constraint sampling in the Linear Programming approach to approximate Dynamic Programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

21 G. van Rossum and *et al. Python Language Reference, version 3.* Python Software Foundation, 2019.

22 S. Vempala. *The Random Projection Method.* Number 65 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS, Providence, RI, 2004.

23 S. Venkatasubramanian and Q. Wang. The Johnson-Lindenstrauss transform: An empirical study. In *Algorithm Engineering and Experiments*, volume 13 of *ALENEX*, pages 164–173, Providence, RI, 2011. SIAM.

24 K. Vu, P.-L. Poirion, C. D'Ambrosio, and L. Liberti. Random projections for quadratic programs over a Euclidean ball. In A. Lodi and *et al.*, editors, *Integer Programming and Combinatorial Optimization (IPCO)*, volume 11480 of *LNCS*, pages 442–452, New York, 2019. Springer.

25 K. Vu, P.-L. Poirion, and L. Liberti. Random projections for linear programming. *Mathematics of Operations Research*, 43(4):1051–1071, 2018.

26 J. Yang, X. Meng, and M. Mahoney. Quantile regression for large-scale applications. *SIAM Journal of Scientific Computing*, 36(5):S78–S110, 2014.