Solving Basis Pursuit: Heuristic Optimality Check and Solver Comparison

DIRK A. LORENZ, Technische Universität Braunschweig
MARC E. PFETSCH and ANDREAS M. TILLMANN, Technische Universität Darmstadt

The problem of finding a minimum ℓ_1-norm solution to an underdetermined linear system is an important problem in compressed sensing, where it is also known as *basis pursuit*. We propose a heuristic optimality check as a general tool for ℓ_1-minimization, which often allows for early termination by "guessing" a primal-dual optimal pair based on an approximate support. Moreover, we provide an extensive numerical comparison of various state-of-the-art ℓ_1-solvers that have been proposed during the last decade, on a large test set with a variety of explicitly given matrices and several right hand sides per matrix reflecting different levels of solution difficulty. The results, as well as improvements by the proposed heuristic optimality check, are analyzed in detail to provide an answer to the question which algorithm is the best.

1. INTRODUCTION

The ℓ_1-minimization problem

$$\min \|x\|_1 \quad \text{s.t.} \quad Ax = b, \tag{P$_1$}$$

for $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ with $m < n$, has lately received a tremendous amount of attention in the literature. It is equivalent to a linear program, and a broad variety of solution algorithms has been developed recently. This is mainly due to its connection to *compressed sensing*, see, e.g., [Donoho 2006; Candès et al. 2006]. Most prominently, under certain conditions, a solution to (P$_1$) coincides with the solution of the NP-hard combinatorial optimization problem

$$\min \|x\|_0 \quad \text{s.t.} \quad Ax = b, \tag{P$_0$}$$

where $\|x\|_0$ denotes the number of nonzeros of x. As a consequence, efficient algorithms for solving (P$_1$) can—under desirable circumstances—be used to find the sparsest solution to an underdetermined linear equation system. In this paper we concentrate on the (practical) solution of (P$_1$) and mostly neglect its connection to (P$_0$).

The main contribution of this article is an extensive computational comparison of solvers for (P$_1$). This includes the proposal of a publicly available test set containing 100 sparse and dense matrices A and 548 right hand sides b. In general, such a comparison is complicated by several facts. The most important one is that the termination requirements for each solver may be different, resulting in final solution points that vary highly in their distance to feasibility and optimality. Of course, there is a trade-off between feasibility/optimality requirements and performance. We cope with this problem by designing the test set in such a way that the optimal solutions are unique. The results of the solvers are then compared both with respect to the running time as well as the distance to the optimal point, which allows to estimate both the distance to feasibility and optimality at the same time. Earlier comparisons of solvers for ℓ_1-related minimization problems can be found, e.g., in [van den Berg and Friedlander 2008; Becker et al. 2011; Yang et al. 2010; Lorenz 2013].

Updated November 18, 2013. This work has been funded by the Deutsche Forschungsgemeinschaft (DFG) within the project "Sparse Exact and Approximate Recovery" under grants LO 1436/3-1 and PF 709/1-1. Moreover, D. A. Lorenz acknowledges support from the DFG project "Sparsity and Compressed Sensing in Inverse Problems" under grant LO 1436/2-1.
Authors' addresses: D. A. Lorenz, Technische Universität Braunschweig, Institut für Analysis und Algebra, 38092 Braunschweig, Germany, email: d.lorenz@tu-braunschweig.de; M. E. Pfetsch and A. M. Tillmann, Technische Universität Darmstadt, AG Optimierung, Dolivostraße 15, 64293 Darmstadt, Germany, email: {pfetsch,tillmann}@mathematik.tu-darmstadt.de.

Another complication is that there is an abundance of possible data and many solvers to choose from. As a consequence, we had to limit the number of tested solvers and the size of the test set. Concerning the solvers, we decided to only consider algorithms that *provably* solve (P_1) and for which implementations are publicly available. We are aware of the following six solvers that meet these requirements: ℓ_1-Homotopy, SPGL1, ℓ_1-Magic, SolveBP/PDCO, YALL1, and ISAL1; additionally, we include the commercial LP solver Cplex and the non-commercial LP solver SoPlex in our experiments. All these solvers are reviewed in Section 3. Concerning the test set, we deliberately decided to only consider matrices with an explicit representation. Thus, we do not use the ability of several solvers to work with implicit matrices by callback functions that perform matrix-vector products. The test set should also be seen independently of the above mentioned ℓ_0-ℓ_1-equivalence, although we also use tools from compressed sensing to guarantee that the optimal solutions are unique, see Section 4. Note that we do *not* consider denoising instances, e.g., $\min\{\, \|x\|_1 : \|Ax - b\|_2 \leqslant \delta \,\}$, although many of the available ℓ_1-solvers are capable of solving this variant as well.

When it comes to solving (P_1), an important practical concern is the typical slow convergence of iterative solvers. In Section 2, we develop a heuristic method, which, if applied repeatedly during an algorithm, often allows for "jumping" to an optimal point of (P_1) and hence for early termination. This heuristic optimality check (HOC) is based on the idea that the iterates allow to approximate the support of an optimal solution; using the support, we try to construct a feasible primal-dual pair. We show in Section 5 that under certain conditions the constructed solutions are optimal. Another set of numerical results show that HOC can help to improve both the solution time by early termination as well as the solution accuracy when built into existing ℓ_1-solvers. It is noteworthy that HOC is related to, though different from, the idea of support estimation, which has been used for a penalized version of (P_1) to improve solution quality [Figueiredo et al. 2007] or to speed up the algorithm [Wen et al. 2010] and in a modification of ℓ_1-minimization to enhance the recovery properties w.r.t. (P_0) [Wang and Yin 2010].

Additionally, we briefly discuss our implementation of a new algorithm to solve problem (P_1), which is a non-differentiable convex optimization problem. One of the classical algorithms for this class of problems is the projected subgradient method (see, e.g., [Shor 1985]), but as far as we know no (practical) application of it to solve (P_1) has appeared in the literature. Therefore, we adapted the infeasible-point subgradient algorithm from [Lorenz et al. 2011a] to ℓ_1-minimization; we call our implementation ISAL1. It allows for approximate projections onto the set $\{\, x : Ax = b \,\}$, which are realized by a truncated conjugate gradient (CG) scheme. In Section 3.1, we discuss our implementation in some more detail. Despite the well-known difficulties of subgradient methods, namely slow convergence and a high effort needed to tune algorithmic parameters, it turns out that ISAL1 performs quite well on small- or medium-scale instances.

The computational comparison is presented in Section 5. We investigate the performance and quality of solutions generated by the eight ℓ_1-solvers mentioned above. We also investigate the impact of HOC on six of these solvers. It turns out that Cplex and SoPlex (using the dual simplex algorithm) perform remarkably well—this refutes the rumor that standard methods for LP solving perform badly for ℓ_1-minimization. SPGL1, ℓ_1-Homotopy, SolveBP/PDCO, and ISAL1 usually produce acceptable solutions within reasonable time. Using HOC for early termination significantly improves the solution quality and running time of SPGL1, ℓ_1-Homotopy, ISAL1, SolveBP/PDCO, and ℓ_1-Magic. YALL1 does not seem to benefit from HOC.

We see the computational comparison presented in this paper as a step towards a common empirical evaluation of the many different algorithms proposed during the last years. This can help to select the right algorithm for different practical purposes, to further improve the available implementations, and possibly to create new algorithmic ideas.

Algorithm 1: Exact Optimality Check (EOC)

```
Input: matrix $A$, right hand side vector $b \neq 0$, vector $x$
1:   deduce candidate (approx.) support $S$ from $x$
2:   if $\hat{x}$ exists with $A_S \hat{x}_S = b$ and $\hat{x}_i = 0 \; \forall \, i \notin S$ then
3:       compute solution $\hat{w}$ to $A_S^\top w = \operatorname{sign}(\hat{x}_S)$
4:       if $\|A^\top \hat{w}\|_\infty = 1$ then
5:           return "success"
```

The test set, implementations of ISAL1 and HOC, as well as detailed computational results can be downloaded from wwwopt.mathematik.tu-darmstadt.de/spear.

2. HEURISTIC OPTIMALITY CHECK (HOC)

Many algorithms for (P_1) produce an (infinite) sequence of points converging to an optimal point. Typically, (practical) convergence becomes slow towards the end, i.e., there is a trade-off between solution speed and quality. In this section, we propose a general method, the *Heuristic Optimality Check* (HOC), to heuristically compute an optimal point from a given iterate. Applied repeatedly within an ℓ_1-solver, this routine often allows for early termination.

We use the following notation: For $S \subseteq \{1, \dots, n\}$, we write A_S for the submatrix consisting of the columns of A and x_S for the subvector consisting of the components of x indexed by S, respectively.

2.1. Theoretical Foundation

The approach is based on the following characterization of optimal solutions of (P_1):

Lemma 2.1. *A vector $x^* \in X := \{x : Ax = b\}$ is optimal for the ℓ_1-minimization problem* (P_1) *if and only if there exists $w \in \mathbb{R}^m$ such that $A^\top w \in \partial\|x^*\|_1$.*

Proof. The normal cone of X is

$$N_X = \{z \in \mathbb{R}^n : \exists\, w \in \mathbb{R}^m : \; z = A^\top w\} = \operatorname{Im}(A^\top)$$

independently of the point where it is evaluated. Thus, N_X is the range of $A^\top$. Since $N_X = -N_X$, the condition given in the statement of the lemma can be rewritten as $-\partial\|x^*\|_1 \cap N_X \neq \varnothing$. This corresponds to a well-known optimality criterion for constrained nonsmooth convex programs; see, e.g., [Ruszczyński 2006, Theorem 3.33]. □

A consequence of Lemma 2.1 is the following. For a given optimal solution x^* of (P_1), every other vector $x \in X$ with the same sign pattern as x^* must also be optimal, since then $\partial\|x^*\|_1 = \partial\|x\|_1$. Thus, if the iterates x^k of a solution algorithm converge to an optimal solution x^* with support S^*, we expect that $\operatorname{sign}(x^k_{S^*}) = \operatorname{sign}(x^*_{S^*})$ for sufficiently large k. However, x^k may still have many more nonzeros than x^* and can also be infeasible. Now, the idea is to try to identify S^* from x^k, construct a *feasible* solution $\hat{x}$ with support S^*, and try to prove its optimality (via Lemma 2.1) by constructing $\hat{w} \in \mathbb{R}^m$ with $A^\top \hat{w} \in \partial\|\hat{x}\|_1$ or, equivalently, $A^\top_{S^*}\hat{w} = \operatorname{sign}(\hat{x}_{S^*})$ and $-\mathbb{1} \leqslant A^\top \hat{w} \leqslant \mathbb{1}$. To simplify the computations, we only solve the equation system and then verify whether $\hat{w}$ obeys the box constraint as well.

Note that if x^* is the unique optimum, we have $|S^*| \leqslant m$ and A_{S^*} has full rank. Thus, it suffices to correctly anticipate S^* and solve $A_{S^*}\hat{x}_{S^*} = b$, $\hat{x}_i = 0$ for all $i \notin S^*$. It then follows that $\hat{x} = x^*$ and, in particular, $\operatorname{sign}(\hat{x}_{S^*}) = \operatorname{sign}(x^*_{S^*})$. If x^* is not unique or the above reasoning is applied to some arbitrary $S \supset S^*$, inducing a full-rank submatrix A_S, instead of S^*, the hope is that $\operatorname{sign}(\hat{x}_S) = \operatorname{sign}(x^*_S)$ will nevertheless hold and optimality can still be proved. This yields the *exact optimality check* (EOC), see Algorithm 1.

```
ALGORITHM 2: Heuristic Optimality Check (HOC)
Input: matrix $A$, right hand side vector $b \neq 0$, vector $x$
1:  deduce candidate (approx.) support $S$ from $x$
2:  compute approximate solution $\hat{w}$ to $A_S^\top w = \operatorname{sign}(x_S)$
3:  if $\|A^\top \hat{w}\|_\infty \approx 1$ then
4:      if $\hat{x}$ exists with $A_S \hat{x}_S = b$ and $\hat{x}_i = 0 \; \forall\, i \notin S$ then
5:          if $(\|\hat{x}\|_1 + b^\top(-\hat{w}))/\|\hat{x}\|_1 \approx 0$ then
6:              return "success" ($\hat{x}$ is an optimal solution)
```

Note that Lemma 2.1 is by no means new; for instance, the result is also derived in the proof of [Fuchs 2004, Theorem 4] via linear programming duality. Moreover, conditions to certify *unique* optima have been studied in more detail in, e.g., [Fuchs 2004; Grasmair et al. 2011].

THEOREM 2.2. *If Algorithm 1 is successful, $\hat{x}$ is an optimal solution for* (P$_1$).

PROOF. If the point $\hat{x}$ exists, it is obviously feasible. If $\hat{w}$ satisfies $A_S^\top \hat{w} = \operatorname{sign}(\hat{x}_S)$ and $(A^\top \hat{w})_i \in [-1, 1]$ for all $i \notin S$ (i.e., i such that $\operatorname{sign}(\hat{x})_i = 0$), we have $A^\top \hat{w} \in \partial\|\hat{x}\|_1$. For Step 4, note that since $b \neq 0$, $\|h\|_\infty = 1$ for every $h \in \partial\|x\|_1$ of any feasible point x. The result hence follows from Lemma 2.1. $\square$

There are different ways to select the candidate support S in Step 1 of Algorithm 1, the easiest being

$$S_\delta^k := \{\, i : |x_i^k| > \delta \,\} \tag{1}$$

for some $\delta \geqslant 0$. An alternative is based on ℓ_1-norm concentration: For a fixed number $c \in (0, 1)$, choose S as

$$S_c^k := \arg\min \Big\{\, |I| : I \subseteq \{1, \dots, n\},\; \sum_{i \in I} |x_i^k| \geqslant c\, \|x^k\|_1 \,\Big\}. \tag{2}$$

Of course, (1) and (2) are in a sense equivalent: For each δ there exists a c such that the two sets S_δ^k and S_c^k coincide, and vice versa. However, a straightforward implementation of (2) requires sorting the $|x_i^k|$'s, yielding an $\mathcal{O}(n \log n)$ running time, whereas (1) can be implemented in $\mathcal{O}(n)$. On the other hand, (1) will only become effective when the entries outside of the optimal support are small enough already and if δ is not too large (to avoid cutting off entries in the optimal support). In contrast, (2) is less dependent on the magnitudes of optimal nonzero entries.

2.2. Practical Considerations

EOC possibly involves a high computational cost when used as a frequently evaluated stopping criterion in an ℓ_1-solver. We now describe how it can be turned into an efficient and practically useful device, the *heuristic optimality check* (HOC). The result is Algorithm 2, which differs from Algorithm 1 in the following aspects.

In Step 2 of Algorithm 2, we wish to obtain a solution to $A_S^\top w = \operatorname{sign}(x_S)$ (if possible). To that end, we use the pseudo-inverse $(A_S^\top)^\dagger = A_S(A_S^\top A_S)^{-1}$ to calculate $\hat{w} = (A_S^\top)^\dagger \operatorname{sign}(x_S)$. In fact, we can avoid the calculation of $(A_S^\top)^\dagger$ by obtaining a solution $\hat{v}$ to $A_S^\top A_S v = \operatorname{sign}(x_S)$ via the CG method and then taking $\hat{w} = A_S \hat{v}$; for a convergence argument (in particular, for the case when A_S is rank-deficient), see [Nemirovskiy and Polyak 1984].

We observed that if S equals the optimal support S^* (or a superset of S^* which induces a full-rank submatrix A_S), then approximating $\hat{v}$ by computing only a few CG iterations is enough to identify $\hat{w}$ with $\|A^\top \hat{w}\|_\infty \approx 1$ (here, as well as in all other comparisons, we used a tolerance value of 10^{-6}). Hence, the computational overhead for HOC in case the support approximations are incorrect can be reduced by limiting the number of CG iterations (we use at most 20). For this reason we first compute $\hat{w}$ and then $\hat{x}$.

The next step is to check whether indeed $\|A^\top \hat{w}\|_\infty \approx 1$. If this is the case, we try to compute $\hat{x}$ with $\hat{x}_i = 0$ for $i \notin S$ and $A_S \hat{x}_S = b$. If $\hat{x}$ exists, $\operatorname{sign}(\hat{x}_S)$ might differ from $\operatorname{sign}(x_S)$ (in particular, $\hat{x}_S$ may contain zeros whereas x_S does not), and we cannot directly apply Lemma 2.1. Instead, we use strong duality

$$\min_{Ax=b} \|x\|_1 = \max_{\|A^\top y\|_\infty \leqslant 1} -b^\top y$$

and conclude that $(\hat{x}, -\hat{w})$ form an optimal primal-dual pair, if the duality gap $\|\hat{x}\|_1 - b^\top \hat{w}$ is (approximately) 0. The empirical results suggest that this optimality check is robust: in our computational experiments, if HOC was successful, it always obtained the (unique) optimal point.

Let us now give a few remarks on the integration of HOC into ℓ_1-solvers. First, note that HOC (and, in particular, the CG scheme within it) does not require A or A_S to be given explicitly: Instead of A_S, we can work with $A \cdot \mathcal{P}_S$ where $\mathcal{P}_S$ is the projection onto the components in S (i.e., $(\mathcal{P}_S x)_j = 0$ for all $j \notin S$ and $(\mathcal{P}_S x)_S = x_S$); similarly $A_S^\top y = \mathcal{P}_S^\top A^\top y$. Thus, HOC can be used with fast transforms without the need to extract columns from A to explicitly form submatrices A_S, which may be nontrivial when A is given as an (implicit) operator.

Moreover, the solver type should be taken into account when choosing a support approximation scheme: In methods usually needing many (cheap) iterations, (2) offers a more dynamical scheme, which—albeit being more expensive—may recognize the optimal support far earlier than the hard thresholding scheme (1). Examples where (2) proved very effective are ISAL1 and SPGL1, see Sections 3.1 and 3.5, respectively. However, SPGL1 maintains a different type of approximate support anyway (called "active set"), so suitable support approximations are immediately available and were found to work just as well.

On the other hand, for algorithms with typically only relatively few (but expensive) iterations, (1) may be more adequate. An example where HOC using this variant improved performance is the interior-point method ℓ_1-Magic, see Section 3.3. In this case, we applied (1) with a different δ in each iteration, namely, $\delta^k := \|x^k\|_1/10^6$. This scheme was also used (at constant iteration intervals) in YALL1. Finally, in ℓ_1-Homotopy, we used the inherent support build by the algorithm and additional hard thresholding (1) with $\delta = 10^{-9}$.

To limit the overall computational effort in either case, we only execute HOC if the (approximate) support S has changed. Since there always exists an optimal solution with at most m nonzeros, we do not run HOC as long as $|S| > m$. Moreover, we run HOC only every R iterations. The choice of this HOC frequency R is non-trivial: lower values for R increase the overhead induced by HOC, but may also lead to early termination if HOC is successful. Thus, one can clearly expect a certain trade-off between the overhead by running HOC and the speed-up it (hopefully) yields. The goal should thus be to try to maximize HOC efficiency while at the same time keeping the overhead low for at least those instances where HOC fails.

Since in second-order methods, iterations are relatively expensive and change the iterate point quite significantly, it makes sense to try HOC in every iteration, i.e., set $R = 1$. Thus, we did so for ℓ_1-Magic and SolveBP/PDCO. For the first-order solvers, we benchmarked R by choosing the closest integer $\lceil \alpha\, m \rfloor$ to the value $\alpha\, m$, with $\alpha \in \{\frac{1}{1000}, \frac{1}{500}, \frac{1}{200}, \frac{1}{100}, \frac{1}{50}, \frac{1}{20}, \frac{1}{10}\}$, which led to the best balance between speed-up and accuracy improvement (over our whole testset), respectively. (We used fractions of m for these tests since this is the leading dimension of the systems solved within HOC.) Thus, we set R to $\lceil m/500 \rfloor$ in ℓ_1-Homotopy, to $\lceil m/100 \rfloor$ in SPGL1 and ISAL1, and to $\lceil m/10 \rfloor$ in YALL1.

Note that for YALL1, we chose the largest interval between executions of HOC because the success rate is lower; the results were about the same as for the R-value that seemed best for the other first-order methods with a typically high number of iterations (i.e., SPGL1 and ISAL1), but naturally the induced overhead is smaller. Moreover, the choices were not entirely unambiguous: For SPGL1, with $R = \lceil m/500 \rceil$, HOC success occured slightly more often, but the mean running times were unsatisfactory. Similarly, ISAL1 with $R = \lceil m/200 \rceil$ solved more instances but the running time improvement was notably less.

Numerical results supporting the claimed usefulness of the HOC will be presented later, when we turn to the computational comparison of ℓ_1-solvers.

3. ALGORITHMS FOR EXACT ℓ_1-MINIMIZATION

In this section we briefly introduce the state-of-the-art solvers used in the computational comparison. These solvers were chosen according to the following guidelines.

— Only *exact* solvers for ℓ_1-minimization (P_1) are considered, i.e., algorithms that are theoretically guaranteed to be capable of solving (P_1) to optimality. This choice excludes probabilistic or heuristical methods. It also excludes, for example, orthogonal matching pursuit (OMP), since it only solves (P_1) if the solution is sufficiently sparse (see, e.g., [Davies et al. 1994; Pati et al. 1993; Tropp and Gilbert 2007]) and hence is not a *general-purpose* ℓ_1-solver.

— Furthermore, we restrict ourselves to algorithms with readily available implementations (all but two in MATLAB).

As a consequence, we leave out, e.g., NESTA [Nesterov 2005; Becker et al. 2011]. Although in theory it can solve (P_1) exactly, the implementation (provided by the authors of [Becker et al. 2011] at www-stat.stanford.edu/~candes/nesta/) is designed for instances in which the matrix A is a partial isometry (i.e., $AA^T = I$). To be fair, the NESTA implementation can handle general matrices, but then it (currently) requires a full singular value decomposition of A, which adds a considerable amount of computational effort to the method; this renders it non-competitive with the other considered ℓ_1-solvers for the larger instances. Moreover, we did not include the "Bregman iteration" [Yin et al. 2008] (since the available implementation uses an outdated subproblem solver) and the linearized Bregman iteration [Cai et al. 2009], since it solves a slightly perturbed problem, see [Yin 2010].

In the following, we provide an overview of the algorithms and their respective implementations which we included in our computational comparison.

3.1. ISAL1

In the following, we describe our implementation of ISAL1, i.e., the specialization of the infeasible-point subgradient algorithm (ISA) developed in [Lorenz et al. 2011a] to basis pursuit. The theoretical aspects of this adaption have been discussed in Section 5.2 of [Lorenz et al. 2011a] already. Here, we use several ideas to improve practical performance, regardless of the theoretical convergence conditions, which we will discuss below.

The basic iteration scheme underlying ISAL1 is

$$x^{k+1} := \mathcal{P}_X\Big(x^k - \lambda_k \frac{\|x^k\|_1 - \varphi}{\|h^k\|_2^2} h^k\Big),$$

where $\mathcal{P}_X$ is the (Euclidean) projection onto the feasible set $X := \{\, x : Ax = b \,\}$, (λ_k) are vanishing positive step size parameters, φ is a lower bound on the optimal objective function value, and $h^k \in \partial\|x^k\|_1$ is a subgradient of the ℓ_1-norm at the current point x^k. However, as mentioned in the introduction, the ISA framework employs adaptive approximate projections onto the feasible set. The exact projection onto X is replaced by a truncated CG scheme; see Section 5.2 in [Lorenz et al. 2011a] for details. The theoretical convergence result (Theorem 3 in [Lorenz et al. 2011a]) hinges on dynamic bounds on the projection accuracies and thus on the number of CG steps needed to guarantee these accuracies. Surprisingly, in practice this does not seem to be essential. We observed that limiting the number of CG steps to a small constant suffices for practical convergence. The actual number seems to depend on the density of A—the sparser the matrix, the more CG steps are apparently necessary; for dense matrices often two or three steps suffice. As a default in our ISAL1 implementation, we use at most five CG iterations to approximate the projection.

A well-known practical property of subgradient methods is the so-called "zig-zagging" of the iteration points (x^k), which is also a main cause for the slow local convergence often exhibited by such algorithms. To alleviate this effect, we apply the following stabilization scheme, suggested in [Löbel 1997]: replace the current subgradient h^k by the convex combination

$$\tilde{h}^k := 0.6h^k + 0.2h^{k-1} + 0.1(h^{k-2} + h^{k-3}). \tag{3}$$

As a consequence, $\tilde{h}^k \in \partial\|x^k\|_1$ does not generally hold. Other stabilization techniques have been proposed, e.g., in [Camerini et al. 1975; D'Antonio and Frangioni 2009; Lim and Sherali 2005], or in [D'Antonio and Frangioni 2009; Larsson et al. 1996] for conditional subgradients.

Furthermore, as suggested in [Held et al. 1974], instead of using a predetermined step size parameter sequence (λ_k) we reduce λ_k by a factor of $\frac{1}{2}$ after a number p of consecutive iterations without "relevant" improvement in the objective, i.e., $\min\{\|x^0\|_1, \dots, \|x^{k-1}\|_1\} - \|x^k\|_1 \geqslant 10^{-6}$, or after $\overline{p}$ iterations (independently of the improvement). Choosing improvement (as opposed to change) in the objective turned out to work well in practice. We start with an initial value of $\lambda_0 = 0.85$. Moreover, we performed a number of experiments to estimate good rules for choosing p and $\overline{p}$. It turns out that they depend on the density of A. Details can be found in the source code available at wwwopt.mathematik.tu-darmstadt.de/spear. Furthermore, we apply a restart mechanism that resets some parameters to more conservative values (e.g., p is multiplied by 5) in case the iterates fail to converge for the default parameter choices.

The dynamic step sizes use an estimate φ of the optimal value φ^* of (P$_1$). This estimate is obtained as a dual lower bound. More precisely, given the starting point x^0 with (approximate) support S^0, we compute an approximate solution w^0 to $A_{S^0}^\top w = -\operatorname{sign}(x^0)$. The scaled vector $w^0/\|A^\top w^0\|_\infty$ is a feasible solution to the dual problem of (P$_1$),

$$\max\ -b^\top w \quad \text{s.t.} \quad -\mathbb{1} \leqslant A^\top w \leqslant \mathbb{1}, \tag{4}$$

and thus implies the dual lower bound $-b^\top w^0/\|A^\top w^0\|_\infty =: \varphi \leqslant \varphi^*$, which we use in our implementation. Note that other lower bounds are also easily available, e.g., 0 is always valid, and $b^\top b/\|A^\top b\|_\infty$ always yields a positive bound (since $b \neq 0$). However, we found the duality-based approach described above often gives a better bound and worked well in our experiments. (Note also that HOC can be used to obtain such lower bounds even if it is unsuccessful.)

Moreover, in the projection scheme (cf. (52)–(54) in [Lorenz et al. 2011a]), we usually compute $z = AA^\top q$ separately as $v = A^\top q$, $z = Av$, instead of directly computing $AA^\top$. For sparse A, this allows for faster evaluation of matrix-vector products (since $AA^\top$ may be dense), while for large dense matrices, computing $AA^\top$ explicitly can yield memory problems, so that in either case, the separate computation is preferable. For small dense matrices, $AA^\top$ is precomputed and used directly. (The experiments in [Lorenz et al. 2011a] show that, in this case, it still makes sense to approximate the projection instead of computing it exactly.) Note also that ISA with dynamic step sizes, i.e., Algorithm 2 in [Lorenz et al. 2011a], has an inner projection accuracy refinement loop, to handle the case $\|x^k\|_1 \leqslant \varphi$ and $x^k \notin X$. However, this case never occurred in our experiments with ISAL1; the implementation therefore does not presently contain such a refinement phase.

Experiments showed that generally, $x^0 := A^\top b$ seems to be a good starting point. If $AA^\top = I$, x^0 is feasible, since then $Ax^0 = AA^\top b = b$. This property of A allows for fast (exact) projections onto X and is (approximately) fulfilled by several types of matrices in our test set, see Section 4.

Additionally, two other aspects greatly improve the performance of ISAL1: Scaling the right hand side to unit Euclidean length, i.e., working with $b/\|b\|_2$ instead of b, helps in terms of numerical stability and convergence speed—for instance, scaling results in ISAL1 achieving more accurate solutions in less time for about 95% of the HDR test instances below (the geometric mean running time reduced by about 81%, the average accuracy improved by 5 orders of magnitude).

We terminate ISAL1 (or initiate a restart) as soon as the step sizes become too small (close to numerical precision) or when a stagnation of algorithmic progress is detected (either no relevant improvement of the objective or no change in the (approximate) support over a span of iterations), respectively. Although we also look at ISAL1 without HOC for the numerical comparison to follow, our implementation uses HOC as the (main) stopping criterion by default, since this often allows us to "jump" to the exact optimal solution before the algorithm would terminate with respect to other conditions. When stopping for reasons other than successful HOC, we test whether the current approximate support induces a feasible solution (having this support) with better objective than the best solution obtained so far; should this be the case, the support-induced solution is returned instead.

Our MATLAB implementation of ISAL1 is available at wwwopt.mathematik.tu-darmstadt.de/spear (we used version 0.91).

3.2. Homotopy Method

The homotopy method [Osbourne et al. 2000; Malioutov et al. 2005] (see also [Yang et al. 2010]) works with the following auxiliary problem with a non-negative parameter λ:

$$\min_{x \in \mathbb{R}^n} \tfrac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1. \tag{5}$$

More precisely, denoting by x^*_λ a solution of (5) for some $\lambda \geqslant 0$, it was shown that the solution path $\chi : \lambda \mapsto x^*_\lambda$ following the change in λ is piecewise linear [Osbourne et al. 2000]. Moreover, for $\lambda \geqslant \|A^\top b\|_\infty$, we have $x^*_\lambda = 0$, and for $\lambda \to 0$, x^*_λ converges to a solution of (P_1). The homotopy method starts with a large λ (and $x^*_\lambda = 0$) and then identifies the "breakpoints" of χ (which correspond to changes in the support of the x^*_λ's). This yields a decreasing sequence of λ's so that x^*_λ approaches the optimum as $\lambda \to 0$.

The homotopy method provably solves (P_1), whereas its well-known variant LARS [Efron et al. 2004] is only a heuristic for (P_1), since it leaves out a critical algorithmic step (namely, that indices may leave the support of x^*_λ again after they were added in some previous iteration). However, if the solution is sufficiently sparse, i.e., in a favorable situation from the compressed sensing perspective, both methods are equivalent, see [Donoho and Tsaig 2008], and only need as many iterations as the (then unique) optimal solution of (P_1) has nonzero entries[1]. Note that the best known worst-case bound on the number of iterations of the homotopy method is exponential (as for the simplex method from linear programming with virtually all deterministic pivot rules, see, e.g., [Klee and Minty 1972]), reflecting the possible traversal of all sign-patterns [Mairal and Yu 2012].

We used the implementation ℓ_1-*Homotopy* from users.ece.gatech.edu/~sasif/homotopy (version 1.0); see also [Asif 2008].

3.3. ℓ_1-Magic

The ℓ_1-Magic algorithm is a specialization of the primal-dual interior-point method (IPM) from [Boyd and Vandenberghe 2004] to an LP reformulation of (P_1). The linear program

[1]This is called the *k-step solution property*; other sufficient conditions besides a certain solution sparsity can also ensure this property, see [Duan et al. 2011].

considered here is

$$\min\ \mathbb{1}^\top u \quad \text{s.t.} \quad -u \leqslant x \leqslant u,\ Ax = b. \tag{6}$$

Implicitly, $u \geqslant 0$ holds, and the objective effectively minimizes the absolute values of x. Notably, and unlike typical IPMs known from linear (or convex) programming, ℓ_1-Magic solves intermediate equation systems with a conjugate gradient method, thereby avoiding some computationally expensive matrix inversions. A more detailed description is provided within the user's guide of the ℓ_1-Magic implementation. We used the version 1.1 which can be found at www.l1-magic.org.

3.4. SolveBP/PDCO

The SolveBP function comes as a part of SparseLab (sparselab.stanford.edu) and solves (P_1) by applying PDCO (www.stanford.edu/groups/SOL/software/pdco.html), a primal-dual log-barrier method for convex objectives, to the equivalent linear program

$$\min\ \mathbb{1}^\top x^+ + \mathbb{1}^\top x^- \quad \text{s.t.} \quad Ax^+ - Ax^- = b,\ x^+ \geqslant 0,\ x^- \geqslant 0, \tag{7}$$

which arises from the standard split of variables $x = x^+ - x^-$ with $x^+ := \max\{0, x\}$ and $x^- := \max\{0, -x\}$. Like ℓ_1-Magic, SolveBP only needs callback functions for multiplication with A and $A^\top$ to perform all necessary computations involving matrices. We used version 2.1 of SparseLab.

3.5. SPGL1

The SPGL1 algorithm, introduced in [van den Berg and Friedlander 2008], solves (P_1) via a sequence of LASSO problems with suitably chosen parameters τ,

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 \quad \text{s.t.} \quad \|x\|_1 \leqslant \tau, \tag{LS$_\tau$}$$

which are solved efficiently with a specialization of the spectral projected gradient method (see [Birgin et al. 2000]).

We used version 1.7 of SPGL1, available at www.cs.ubc.ca/labs/scl/spgl1, which has proven to quickly provide (approximate) solutions to (P_1) in various experiments already, see, e.g., [Becker et al. 2011; van den Berg and Friedlander 2008; Yang and Zhang 2011].

3.6. YALL1

The YALL1 software package (yall1.blogs.rice.edu) contains an implementation of an alternating direction method (ADM) tailored to (P_1). The paper [Yang and Zhang 2011] describes both a primal-based ADM, which applies a soft thresholding/shrinkage operator and thus corresponds to an inexact augmented Lagrangean algorithm for (P_1), and a dual-based ADM. Two variants of the latter algorithm are implemented in YALL1, with slightly different iterate updates for the cases $AA^T = I$ and $AA^T \neq I$. (Note that the code, while designed for the first situation, handles $AA^\top \neq I$ without requiring additional expensive computations such as the full SVD constructed in the NESTA program in this case.) We used version 1.3 of YALL1.

3.7. CPLEX

It is often mentioned that (P_1) is essentially a linear program, see (6) and (7). Thus, as a reference, we also solve (P_1) with the dual simplex method of the commercial LP solver CPLEX (IBM ILOG CPLEX Optimization Studio 12.4.0.1), applied to (7).

It should be noted that while CPLEX is accessed via a compiled C-library, all other implementations considered in this paper are MATLAB programs (although the projections onto the ℓ_1-ball in SPGL1 are performed by an external C-program, embedded in the MATLAB code via MEX-functions). Since (at user level) MATLAB is an interpreted language,

Table I. Matrix constructions and corresponding abbreviations.

name	dim.	matrix construction/type
BAND	$m \times m$	band matrix with bandwidth 5, entries drawn uniformly at random from $(0, 1)$; the upper right and lower left corner of the matrix are filled such that we obtain a circulant zero pattern
BIN	$m \times n$	binary matrix, entries drawn uniformly from $\{0, 1\}$
BINB	$m \times m$	binary full-rank square matrix, see BIN
BLROW	$m \times m$	block diagonal matrix with a full row-block at the bottom; the square diagonal blocks have dimension randomly chosen from $\{5, \dots, 10\}$, the row-block spans 5 rows; the last diagonal block may thus be smaller than 5×5; entries are drawn uniformly at random from $(0, 1)$
CONV	$m \times m$	convolution matrix for a convolution with a Gaussian kernel with variance 1/2 (truncated such that we obtain a banded matrix with bandwidth 7)
GAUSS	$m \times m$	entries are i.i.d. Gaussian $\mathcal{N}(0, 1)$
HAAR	$m \times m$	Haar matrix (requires $m = 2^r$, $r \in \mathbb{N}$)
HAD	$m \times m$	Hadamard matrix (requires $m = 2^r$, $r \in \mathbb{N}$)
ID	$m \times m$	identity matrix
INT	$m \times n$	integer matrix, entries drawn uniformly from $\{-10, -9, \dots, 9, 10\}$
PHAD	$m \times n$	partial Hadamard matrix, consisting of m randomly chosen rows of the $n \times n$ HAD matrix
PRST	$m \times n$	partial real sinusoid transform matrix, consisting of m randomly chosen rows of the $n \times n$ RST matrix
ROB	$m \times m$	random orthonormal basis matrix
RSE	$m \times n$	random sign ensemble, entries are drawn from a Bernoulli ±1 distribution
RST	$m \times m$	real sinusoid (Fourier) transform matrix
TER	$m \times n$	ternary matrix, entries are drawn uniformly from $\{-1, 0, +1\}$
URP	$m \times n$	uniform random projection matrix, generated by taking m random rows of an $n \times n$ random orthogonal matrix
USE	$m \times n$	uniform spherical ensemble, columns are m-vectors uniformly distributed on the sphere $\mathbb{S}^{m-1}$

there will be a certain bias to the advantage of Cplex in terms of running times. On the other hand, we use a C-program that sets up the linear programs from the raw data files via the SCIP LP-interface (version 3.0.0, available within the SCIP optimization suite at scip.zib.de) and then calls the dual simplex algorithm of Cplex. This LP construction step is included in the time measurements for Cplex.

3.8. SoPlex

Since Cplex is commercial, it is also of interest to include a non-commercial LP solver. Since for Cplex, we used the interface from SCIP, we consider the solver SoPlex (version 1.7.0), cf. [Wunderling 1996], that comes with the SCIP optimization suite. The C-program setting up the linear programs from data files is the same one we used for Cplex; the LP construction is again included in the time measurements, and, as for Cplex, we use the dual simplex method.

4. THE TEST SET

We performed computational experiments on a test set consisting of 100 matrices A of different types (random or structured matrices and concatenations thereof) and four or six right hand side vectors b per matrix. The dimensions and specific constructions of the matrices are summarized in Tables I and II. After construction, the columns of each matrix were normalized to unit Euclidean length. Whenever a column appeared twice in a matrix, we randomly added single entries in one such column and renormalized, until all columns were unique.

For each matrix A, we build two sets of right hand side vectors, from solution vectors with high dynamic range and low dynamic range, respectively. For either dynamic range, the first two vectors b were each constructed using a vector x whose support satisfies the

Table II. The $m \times n$ matrices in our test set: matrices with $m \geqslant 2048$ are sparse, those with $m \leqslant 1024$ are dense (but may contain sparse subdictionaries); concatenations of matrix types are listed in square brackets in order of appearance.

m	n	matrix constructions
512	1024	BIN, INT, PHAD, PRST, RSE, TER, URP, USE, [HAAR,ID], [HAAR,RST], [HAD,ID], [ROB,RST]
	1536	BIN, INT, PHAD, PRST, RSE, TER, URP, USE, [BAND,GAUSS,RST], [BINB,ID,HAD], [HAAR,ID,RST], [HAAR,ROB,RST]
	2048	BIN, INT, PHAD, PRST, RSE, TER, URP, USE, [BAND,BINB,HAD,ID], [BAND,BLROW,HAD,ID], [BINB,CONV,HAAR,ROB], [HAAR,ID,ROB,RST]
	4096	[ID,HAAR,ROB,RST,BAND,BINB,BLROW,HAD]
1024	2048	as for 512×1024
	3072	as for 512×1536
	4096	as for 512×2048
	8192	as for 512×4096
2048	4096	[BINB,BLROW], [BINB,CONV], [CONV,HAAR], [HAAR,ID]
	6144	[BAND,BINB,HAAR], [BINB,HAAR,ID], [BLROW,CONV,ID], [CONV,HAAR,ID]
	8192	[BAND,BINB,CONV,ID], [BAND,BLROW,CONV,ID], [BINB,BLROW,HAAR,ID], [BINB,CONV,HAAR,ID]
	12288	[ID,HAAR,BAND,BINB,BLROW,CONV]
8192	16384	as for 2048×4096
	24576	as for 2048×6144
	32768	as for 2048×8192
	49152	[BAND,BINB,BLROW,CONV,HAAR,ID]

Exact Recovery Condition (ERC) [Tropp 2004] and thus is the unique optimal solution to the instance of (P_1) specified by matrix A and vector $b := Ax$. More precisely, the supports of the two vectors per matrix are constructed as follows:

(1) For each support size k (starting with $k = 1$), take a random k-element subset $S \subseteq \{1, \dots, n\}$, and check whether the ERC holds, i.e., if

$$\mathrm{ERC}(A, S) := \max_{j \notin S} \|(A_S^\top A_S)^{-1} A_S^\top A_j\|_1 = \max_{j \notin S} \|A_j^\top (A_S^\top)^\dagger\|_1 < 1. \tag{8}$$

If for the current k a support S satisfying (8) was found, k is increased by 1 and the process is repeated. We stop as soon as 25 trials for some k failed.

(2) For the second vector, we tried to enlarge supports by adding the index j^* to S that (for the current k) defines the maximum in (8): If the support extension $S \cup \{j^*\}$ also satisfies the ERC, we update S to $S \cup \{j^*\}$, increase k by 1, and repeat this procedure as long as possible. Then we continue our search with random supports of size $k = |S| + 1$. For a given matrix, we stop the search routine as soon as it failed for 25 random supports at some size k or if a time limit of one day was exceeded.

For these two supports per matrix A and dynamic range type, we define x by fixing its nonzeros at the locations specified by the respective supports, with random signs and magnitudes. For the high dynamic range (HDR) test set, the magnitudes are 10^{5y} with the y's chosen uniformly at random from $(0, 1)$. For the low dynamic range (LDR), we use entries drawn uniformly at random from $(0, 1)$. (Note that this does not necessarily guarantee vectors with high or low dynamic range; however, it usually is the case: over 80% of the HDR instance dynamic ranges exceed 10^4 and more than 95% of the LDR instance dynamic ranges are below 10^2.) All respective right hand sides are then obtained via $b := Ax$.

If the support of an optimal solution x^* satisfies the ERC, x^* is the unique optimum of both (P_1) and (P_0) (see [Tropp 2004]). Thus, the above-described 400 instances all represent a favorable situation in compressed sensing, i.e., recoverability of sparse solutions via ℓ_1-minimization. As we wish to assess solver performance outside of this so-called ℓ_0-ℓ_1-equivalence as well, we constructed a third set of right hand sides for the first (smaller) 74 matrices as follows (for the other matrices, the construction took unreasonably long):

Given A, we start with a support size k of $m/10$, rounded to the nearest integer. Then, we take a random k-element subset $S \subseteq \{1, \ldots, n\}$ and create a vector x having support S as described above (with HDR or LDR, respectively). Next, we try to find a vector w such that $A^\top w \in \partial\|x\|_1$, using the alternating projections scheme (see [Lorenz 2013]) included in L1TestPack, available at wwwopt.mathematik.tu-darmstadt.de/spear. If such a w was found, x will be an optimal solution to the (P_1) instance given by A and $b := Ax$; see Lemma 2.1. Moreover, uniqueness can be verified by [Grasmair et al. 2011, Theorem 4.7]: Denoting $S' := \{\, i : |(A^\top w)_i| = 1 \,\}$, x is the unique optimum, if $A_{S'}$ has full rank. If no w was found, we repeat this procedure up to 5 times before decreasing k by 1. We iterate the whole scheme until a unique solution was successfully created. Note that it is still possible that a support created this way obeys the ERC; however, we verified that this is not the case for any of our supports generated by this construction.

In total this results in a test set of 548 instances, 274 each with high (HDR) or low (LDR) dynamic range solutions. Note that the repeated verifications of the ERC require a significant amount of time, as did the calculations within the second approach. Overall, the construction of the test set took more than a week of computation time.

In general, a larger number of rows allows for larger supports, as can be seen in Figure 1(a). Note that for larger instances the number of nonzeros has a larger variance in general.

Moreover, the test set exhibits both incoherent and highly coherent dictionaries: Recall the definition (see, e.g., [Donoho and Huo 2001]) of the mutual coherence

$$\mu(A) := \max_{1 \leqslant j \neq k \leqslant n} \frac{|A_j^\top A_k|}{\|A_j\|_2 \cdot \|A_k\|_2}$$

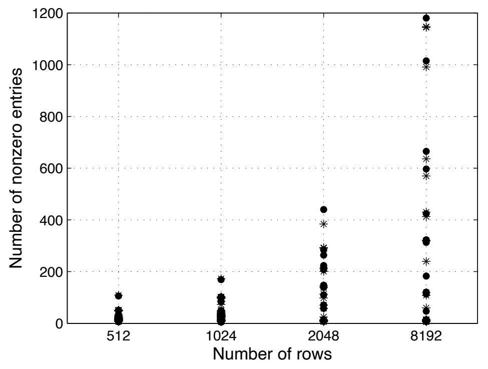
of a matrix A with columns $A_1, \ldots, A_n$. The mutual coherence is an indicator of dependence among the matrix columns and hence, in a sense, for the difficulty of corresponding instances. In general, small $\mu(A)$ yields better recoverability of solutions to (P_0) via ℓ_1-minimization and other approaches. Also, one can show that HOC is successful under certain conditions involving the mutual coherence, see Section 5.2.

The distribution of the mutual coherences of our test matrices is depicted in Figure 1(b). The 25 matrices with coherence larger than 0.95 are used in 114 of the 548 instances (57 each for the HDR and LDR parts); 258 instances (129 HDR, 129 LDR) have a matrix with $\mu < 0.25$. The sparse matrices generally have higher coherence (average 0.88, median 0.98) than the dense ones (average 0.40, median 0.21).

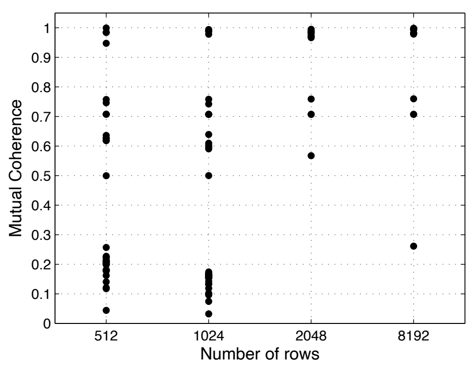
5. COMPUTATIONAL COMPARISON OF ℓ_1-SOLVERS

In the following we present computational results for the ℓ_1-solvers and test set described above. The calculations were performed on a 64bit Ubuntu Linux system with a 3.6 GHz AMD FX-8150 eight-core CPU (8MB cache) and 8GB RAM, using Matlab R2012a (7.14.0).

Since the solvers use a variety of different stopping criteria and tolerance parameters, we decided to assess them in a "black box" manner. Thus, we keep the default values of all algorithmic parameters. Required input parameters were chosen with respect to the goal of an exact solution of (P_1): the (main) tolerance parameter in YALL1 was set to 10^{-6}, and for ℓ_1-Homotopy we set the final regularization parameter to 0 and the iteration limit to $10n$ (never reached in our experiments). Moreover, ℓ_1-Magic does not include a default



(a) Number of nonzero entries in the solutions plotted against the number of matrix rows in the resp. instances (dots: HDR, asterisks: LDR).



(b) Mutual coherences plotted against the resp. number of matrix rows.

Fig. 1. Test set properties: solution sparsity and mutual coherences.

starting point (unlike the other solvers), so we used the projection of the origin onto the constraint set (i.e., $x^0 = A^\top(AA^\top)^{-1}b$); the additional computation times are included in the corresponding solution times stated below.

To eliminate the possible influence of output on the running times, we disabled any such output in all solvers, either by setting a corresponding option or by removing the respective lines of code from the programs. Moreover, for each instance we report the average time (geometric mean) over three runs.

In the following, we define an instance to be *solved* by an algorithm that produces solution $\bar{x}$ if

$$\|\bar{x} - x^*\|_2 \leqslant 10^{-6}, \tag{9}$$

where x^* is the exact optimum (which we know from our test set construction). This ensures $|\,\|\bar{x}\|_1 - \|x^*\|_1\,| \leqslant 10^{-6}$ and $\|A\bar{x} - b\|_\infty \leqslant \|A\|_2\ 10^{-6}$. Note that because of the (perhaps less common) choice of considering the absolute difference $\|\bar{x} - x^*\|_2$ instead of a relative criterion, by the latter inequality, (9) immediately also bounds the feasibility violation. The spectral norms of the matrices in our test set are around 18 on average (median circa 10, maximum about 64); empirically, evaluating the results of all solvers over the whole test set, the quotients $\|A\bar{x} - b\|_\infty / \|\bar{x} - x^*\|_2$ are usually even smaller (very often below 1, all below 5, with a single exception at roughly 9). Thus, for the upcoming results the "solved" status implies that feasibility violation is at most of the order 10^{-6}.

Due to the different natures of the diverse stopping criteria and corresponding parameters employed by the solvers, it cannot be expected that all resulting solutions will satisfy the above condition. (In particular, several algorithms are usually employed in a denoising context, where one is typically content with lesser accuracy levels than may be hoped for in the noisefree setting and the focus is more towards high speed in large-scale situations.) This must of course be taken into account when evaluating the computational results. To that end, we define a solution $\bar{x}$ produced by an algorithm to be *acceptable* if

$$10^{-6} < \|\bar{x} - x^*\|_2 \leqslant 10^{-1}.$$

Note that this definition excludes the solutions of instances that are considered solved. If the upper bound is violated, we consider the obtained point *unacceptable.*

It should be noted that we did not distinguish between the various possible reasons why an algorithm terminated. In particular, the returned point may be the last iterate if an

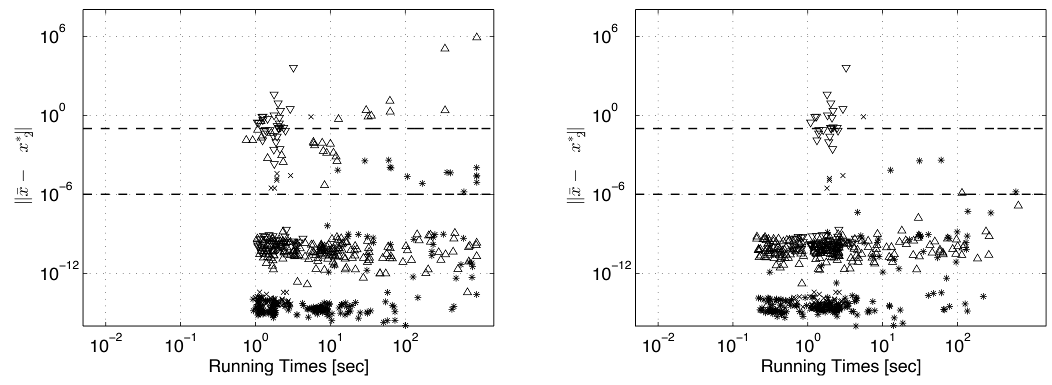


Fig. 2. Results for ISAL1. The plots show running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: △ (HDR), ∗ (LDR). Non-ERC-based instances: ▽ (HDR), × (LDR).

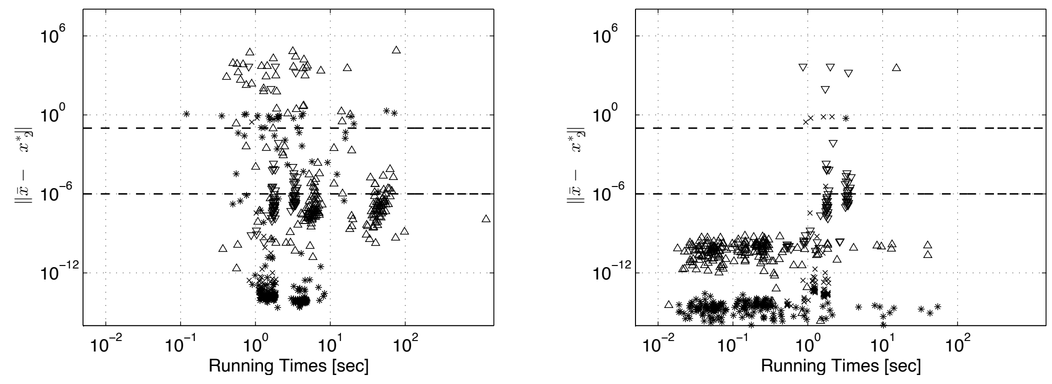


Fig. 3. Results for ℓ_1-Homotopy. The plot shows running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: △ (HDR), ∗ (LDR). Non-ERC-based instances: ▽ (HDR), × (LDR).

algorithm stopped due to numerical problems. Therefore, should an unacceptable solution be obtained, it is not necessarily implied that the algorithm claims this point to be optimal.

5.1. Numerical Results

For all eight solvers we illustrate the performance on the whole test set of 548 instances in Figures 3–9, in which we plot running time (in seconds) versus the distance to the (unique) optimum. For the solvers into which we integrated HOC (i.e., ℓ_1-Homotopy, ISAL1, ℓ_1-Magic, SolveBP/PDCO, SPGL1, and YALL1) we present two figures each, one without and one with HOC. The different test set parts (HDR, LDR) and construction methods (ERC-based, non-ERC-based) are distinguishable by different marker symbols in the plots. Note also that all figures are in double logarithmic scale, and that they share the same axes limits (thus, they are directly comparable).

Let us start by looking at the results without HOC; the changes induced by integrating HOC into the solvers are discussed in Section 5.2.

Firstly, we observe that only Cplex and SoPlex are able to solve all instances w.r.t. the above measure. It is remarkable that the dual simplex methods from Cplex and SoPlex perform that well even for the dense instances, since the codes are optimized for sparse instances. It is noteworthy that the time taken by Cplex or SoPlex to actually solve an

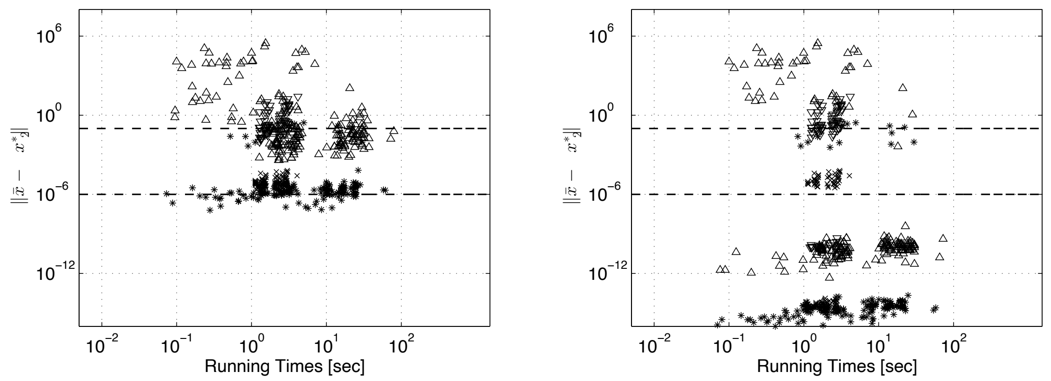


Fig. 4. Results for ℓ_1-Magic. The plots show running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

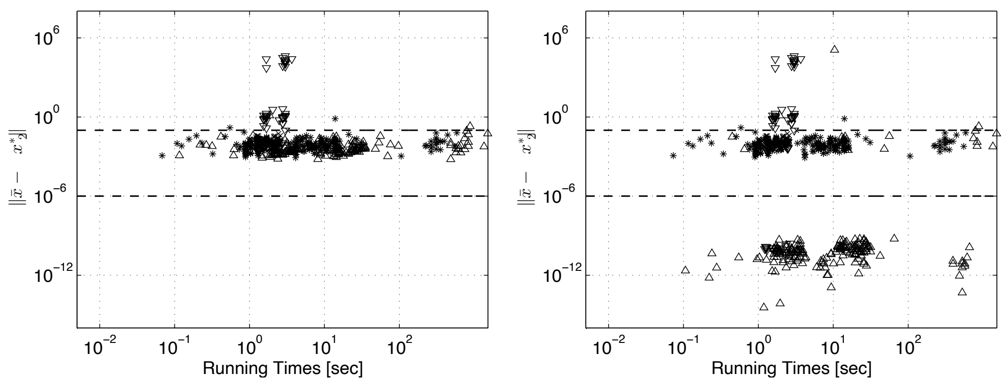


Fig. 5. Results for SolveBP/PDCO. The plot shows running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

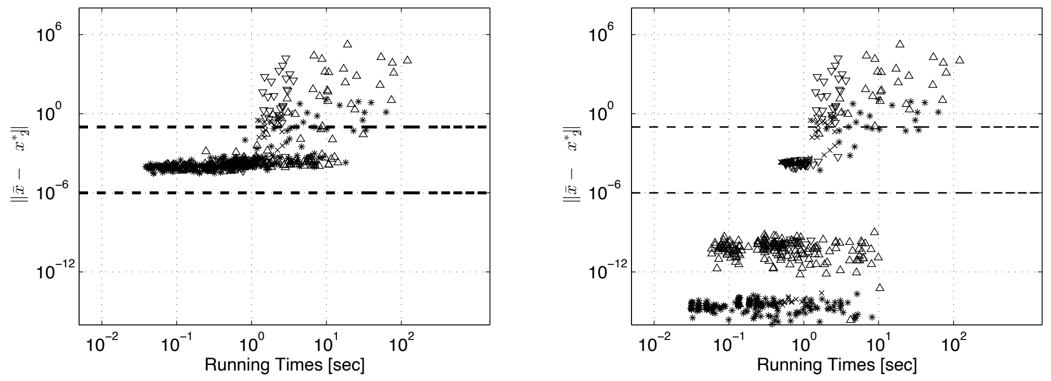
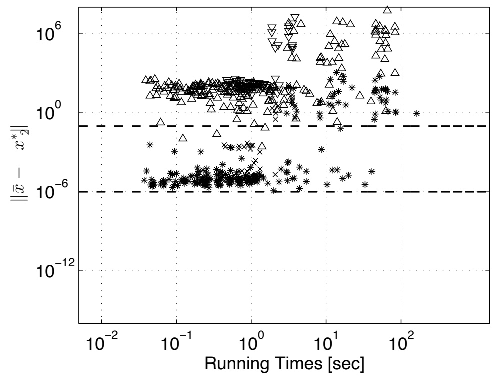


Fig. 6. Results for SPGL1. The plots show running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).



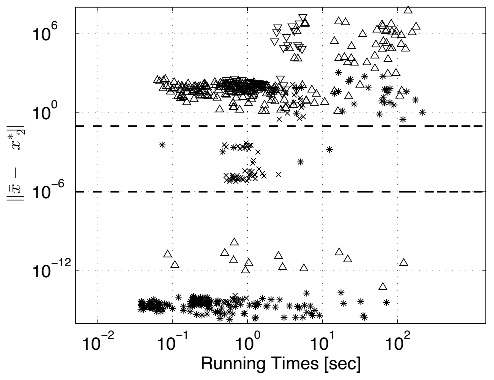


Fig. 7. Results for YALL1. The plots show running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

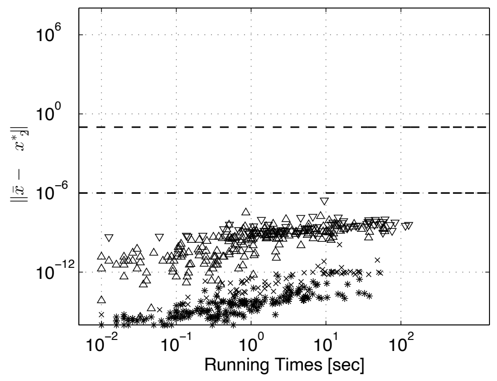


Fig. 8. Results for Cplex. The plot shows running times versus distance to optimum in loglog scale. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

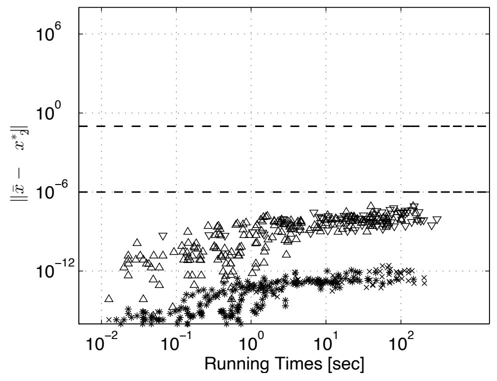


Fig. 9. Results for SoPlex. The plot shows running times versus distance to optimum in loglog scale. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

Table III. Percentage of instances yielding different solutions statuses, per solver. (CPLEX and SoPlex both reached "solved" for *every* instance.)

	% solved			% acceptable			% unacceptable		
Solver	HDR	LDR	all	HDR	LDR	all	HDR	LDR	all
ISAL1	78.1	92.3	85.2	11.3	7.3	9.3	10.6	0.4	5.5
ISAL1 (HOC)	91.2	96.7	94.0	3.6	2.9	3.3	5.1	0.4	2.7
SPGL1	0.0	0.0	0.0	82.5	90.1	86.3	17.5	9.9	13.7
SPGL1 (HOC)	69.0	72.6	70.8	13.9	17.9	15.9	17.2	9.5	13.3
YALL1	0.0	0.0	0.0	1.5	78.1	39.8	98.5	21.9	60.2
YALL1 (HOC)	5.1	64.2	34.7	0.0	18.6	9.3	94.9	17.2	56.0
ℓ_1-Hom.	65.3	83.6	74.5	19.0	6.6	12.8	15.0	9.5	12.2
ℓ_1-Hom. (HOC)	92.3	97.8	95.1	5.8	0.4	3.1	1.8	1.8	1.8
SolveBP	0.0	0.0	0.0	85.0	99.3	92.2	15.0	0.7	7.8
SolveBP (HOC)	70.1	0.0	35.0	14.6	99.3	56.9	15.3	0.7	8.0
ℓ_1-Magic	0.0	15.3	7.7	53.3	82.8	68.1	46.7	1.8	24.3
ℓ_1-Magic (HOC)	62.0	76.6	69.3	5.1	21.5	13.3	32.8	1.8	17.3

instance accounts for a significantly smaller portion of the total time for instances with sparse matrices (roughly 74% on average, median 74%, for both CPLEX and SoPlex) as opposed to dense ones (circa 90% (CPLEX) or 89% (SoPlex) on average, median 96% for both). The overhead arises in the interface to the respective solver and can possibly be improved; however, note that all instances with sparse matrices were solved in well below 1 second by both solvers. We also tried the barrier/interior point method and the primal simplex algorithm of CPLEX for selected instances, but the results were not promising.

A large part of the instances was also solved by ISAL1 and, to a bit lesser extend, by ℓ_1-Homotopy. SPGL1 and SolveBP/PDCO produce acceptable solutions for a majority of instances (but solve none), in accordance with their respective default accuracy settings. The two interior point solvers, ℓ_1-Magic and SolveBP/PDCO, behave quite differently: unlike the latter, ℓ_1-Magic reaches acceptable solutions only for about half the HDR instances, but solves several LDR instances. Similarly, YALL1 hardly produces any acceptable solution for the HDR test set part and only for about three quarters of the LDR instances; we also ran YALL1 with input tolerance parameter 10^{-12} on our test set, but the results were only very slightly better. (A possible explanation for the suprisingly unreliable behavior of YALL1 could lie in the modified iterate updates it performs in case $AA^\top \neq I$ (which holds for all our test matrices), as opposed to the situation $AA^\top = I$ which YALL1 is designed for. On the other hand, the large difference between results it achieves on HDR versus LDR test instances suggest a rather strong dependence of its practical convergence behavior on the solution dynamic range.)

Moreover, note that all solvers can handle the LDR instances better than the HDR instances: The number of solved or acceptable instances is notably higher on the LDR test set part, see Table III.

Looking also at the running times (for now, still without HOC), we immediately observe from the figures that SPGL1 and YALL1 are relatively fast, but all other solvers are somewhat comparable in speed, with SolveBP/PDCO and ISAL1 being on the slower end of the spectrum. However, such a cursory analysis is hardly fair, given the different solution accuracy demands the solvers try to satisfy. By default, ISAL1, ℓ_1-Homotopy, CPLEX, and SoPlex aim at high accuracy (i.e., "solved" status), while the other solvers are content with "acceptable" solutions. Thus, we look at these two solver groups separately and compare their running times on the subsets of instances which all four solvers in the respective group solved, or for which they reached at least acceptable solutions, respectively, see Tables IV and V. Since it performed extremely poorly on the HDR instances (see Table III), we excluded YALL1 from Table V to avoid a strong bias towards LDR instances in the stated running times.

From these tables, we can observe that for the smaller instances (having up to 4096 columns), SPGL1 is clearly the fastest algorithm to obtain acceptable solutions; for the large-scale instances (almost all of which have sparse matrices), ℓ_1-Magic performs better than SPGL1, but the LP codes Cplex and SoPlex are much faster still *and* actually solve all instances. For solving the smaller instances to high accuracy, however, one can do better than these two: While Cplex still has the lowest average running times (cf. the rows corresponding to $n \leqslant 4096$ in Table IV), it often yields high maximum times for the listed instance groups. SoPlex and ISAL1 seem good non-commerical alternatives: SoPlex—like Cplex—solving all instances; ISAL1 solving many, and at typically lower running time than SoPlex and ℓ_1-Homotopy (and smaller maximum times even than Cplex). Moreover, ℓ_1-Homotopy is much faster than ISAL1 on large-scale problems, but solves considerably less instances. Similarly, SolveBP/PDCO produces the most acceptable solutions but is much slower than the other algorithms in Table V. Finally, it should be mentioned that YALL1 is comparable to SPGL1 on the LDR test set part, although it achieved acceptable solutions fewer times and is slightly slower.

We note in particular that ℓ_1-Homotopy exhibits moderate to large running times and produces over 12% unacceptable solutions. This is perhaps surprising, since many instances have very sparse solutions, so one could expect the k-step solution property (cf. [Donoho and Tsaig 2008]) to hold. In this case, ℓ_1-Homotopy should obtain the true optimum very quickly. We will therefore discuss the behavior of ℓ_1-Homotopy, and possible improvements of the implementation at hand, in more detail in Section 5.3 below.

Additionally, the Tables IV and V show that the running times generally increase with the number of columns, i.e., with the dimension of the solution vector. Thus, the points on the right in the point clouds in Figures 3–9 usually belong to larger instances.

It should also be noted that all solvers except Cplex and SoPlex would benefit from fast matrix-vector multiplications (e.g., for the real sinusoid transform RST). This is especially true for SPGL1, YALL1, and ISAL1 which use only a small number of matrix-vector multiplications per iteration.

As we will discuss in the next part, the picture we get for the running time and solution accuracy behavior changes very significantly when we consider the algorithms *with* HOC integrated into them.

5.2. Impact of the Heuristic Optimality Check

The benefit of HOC is clearly visible for ℓ_1-Homotopy, SPGL1, ℓ_1-Magic, SolveBP/PDCO, and ISAL1: In all cases the point clouds "moved down left", although the speed-up is less distinct for the interior-point solvers ℓ_1-Magic and SolveBP/PDCO than for the other three (first-order) solvers. Moreover, it becomes apparent from Table III and from comparing the two plots in Figures 3, 2, 4, 6, and 5, respectively, that by using HOC, one can often drastically improve the accuracy of the obtained solutions, especially on instances for which only acceptable solutions were reached without HOC (i.e., solutions corresponding to markers lying between the horizontal dashed lines). A specific example is shown in Figure 10, where one clearly sees how HOC allows for jumping to the optimum (up to numerical precision). Note that the slightly better accuracy of the solution produced by HOC in ISAL1 (left picture) is due to numerical issues.

An exception to the overall success of integrating HOC is YALL1. Here, the performance only improved for very few HDR instances but for about 64% of the LDR instances, see Table III. In YALL1, the (approximate) supports typically vary often and significantly, suggesting that early extraction of good estimates of the optimal support is often impossible. Moreover, YALL1 only reaches acceptable solutions many times on the LDR test set part; as we already observed, HOC seems particularly successful in improving solutions from that region (although it does in fact also work successfully on some of the "inacceptable"

Table IV. Geometric means of the running times (in seconds) of the algorithms (*without* HOC) aiming at "solved" solution status, for instances that all four could solve, grouped by number n of columns.

n	used/of		ℓ_1-Hom.	ISAL1	CPLEX	SoPlex
1024	56/72	solved	63	64	72	72
		avg. t	1.70	1.35	0.37	0.89
		max t	6.18	2.93	2.56	13.02
1536	60/72	solved	66	66	72	72
		avg. t	2.20	1.63	0.75	1.94
		max t	7.06	5.09	5.56	41.76
2048	101/144	solved	120	121	144	144
		avg. t	3.42	3.26	1.59	4.53
		max t	50.41	14.40	19.17	89.95
3072	53/72	solved	60	61	72	72
		avg. t	6.75	6.72	5.26	13.87
		max t	55.17	25.04	44.43	150.00
4096	64/94	solved	73	82	94	94
		avg. t	6.32	9.98	3.22	7.96
		max t	74.30	70.82	126.96	303.95
6144	5/16	solved	6	12	16	16
		avg. t	2.02	31.61	0.03	0.04
		max t	19.95	37.27	0.03	0.05
8192	6/22	solved	6	16	22	22
		avg. t	3.08	63.32	0.27	0.39
		max t	19.22	73.11	2.13	2.87
12288	2/4	solved	2	4	4	4
		avg. t	1.48	123.14	0.05	0.09
		max t	2.03	147.76	0.05	0.10
16384	6/16	solved	6	14	16	16
		avg. t	18.73	194.33	0.21	0.32
		max t	1204.31	237.66	0.29	0.42
24576	1/16	solved	1	13	16	16
		avg. t	5.86	460.46	0.30	0.44
		max t	5.86	460.46	0.30	0.44
32768	3/16	solved	4	14	16	16
		avg. t	5.96	857.45	0.36	0.57
		max t	7.45	900.03	0.36	0.57
49152	0/4	solved	1	0	4	4
		avg. t	–	–	–	–
		max t	–	–	–	–

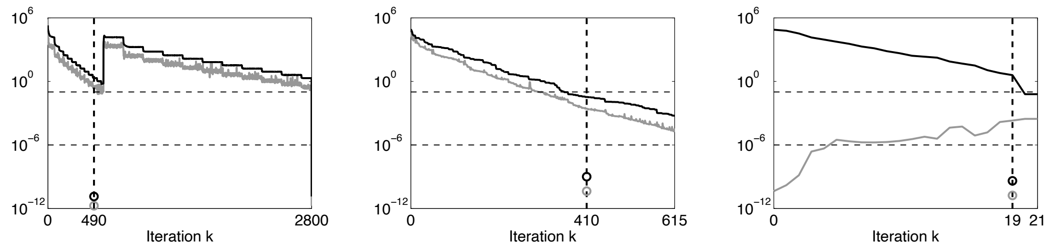


Fig. 10. Impact of HOC in the solvers ISAL1 (left picture), SPGL1 (middle), and ℓ_1-Magic (right) for the HDR instance consisting of the 1024×8192 matrix A and b constructed with the second ERC-based variant. The upper curves trace the distance of the current iterates to the known optimum ($\|x^k - x^*\|_2$), and the curves below show the current feasibility violation ($\|Ax^k - b\|_\infty$) per iteration. Circles indicate the corresponding measures after HOC was successful, at the corresponding iteration.

Table V. Geometric means of the running times (in seconds) of the algorithms (*without* HOC) aiming at "acceptable" solution status, for instances that all three could reach at least acceptable solutions, grouped by number n of columns.

n	used/of		ℓ_1-Magic	SPGL1	SolveBP
1024	63/72	$\leqslant$ acceptable	65	68	72
		avg. t	1.52	0.16	1.28
		max t	2.42	1.63	2.32
1536	58/72	$\leqslant$ acceptable	62	68	67
		avg. t	2.27	0.23	1.76
		max t	3.94	1.49	3.80
2048	112/144	$\leqslant$ acceptable	122	131	134
		avg. t	5.07	0.38	4.07
		max t	17.64	6.03	17.44
3072	54/72	$\leqslant$ acceptable	57	66	61
		avg. t	11.96	0.70	9.40
		max t	29.43	4.68	28.21
4096	57/94	$\leqslant$ acceptable	64	81	81
		avg. t	9.42	0.96	9.03
		max t	37.00	10.82	35.12
6144	7/16	$\leqslant$ acceptable	8	12	16
		avg. t	0.37	0.75	1.64
		max t	1.12	2.94	5.77
8192	9/22	$\leqslant$ acceptable	13	12	21
		avg. t	5.16	3.11	8.68
		max t	79.64	12.86	76.06
12288	1/4	$\leqslant$ acceptable	2	2	4
		avg. t	0.84	1.99	5.13
		max t	0.84	1.99	5.13
16384	8/16	$\leqslant$ acceptable	8	14	16
		avg. t	2.82	2.40	64.57
		max t	9.20	23.01	552.15
24576	4/16	$\leqslant$ acceptable	7	8	15
		avg. t	2.49	3.76	297.78
		max t	5.72	5.00	485.01
32768	6/16	$\leqslant$ acceptable	6	9	15
		avg. t	6.63	7.26	113.33
		max t	29.88	33.77	397.44
49152	1/4	$\leqslant$ acceptable	1	2	3
		avg. t	7.39	8.89	310.30
		max t	7.39	8.89	310.30

instances). In particular, when YALL1 does not practically converge—as often happens on the HDR test set part—HOC hardly has a chance to be successful, since apparently no suitable approximate supports can be obtained in the first place. Thus, including HOC in YALL1 is relatively often unsuccessful, but of course increases the running time.

Further details on the impact of HOC can be found in Table VI: The first row group shows how many of the instances (w.r.t. the whole test set, HDR only, or LDR only, respectively) that were solved by the original algorithm were solved faster when employing HOC. The second group of rows row gives the numbers of instances that were only solved when HOC was used. It should be noted that in most cases listed in this second row, the variant employing HOC was not only able to solve the problem, but also required less time than the unmodified algorithm needed to reach an inferior solution. For instance, of the 380 instances that ℓ_1-Magic with HOC could solve, 379 times the early termination due to

Table VI. Impact of HOC in the solvers.

	ℓ_1-Hom.	ISAL1	ℓ_1-Magic	SPGL1	SolveBP	YALL1
solved faster	329/408	386/467	41/42	0/0	0/0	0/0
(HDR)	(148/179)	(186/214)	(0/0)	(0/0)	(0/0)	(0/0)
(LDR)	(181/229)	(200/253)	(41/42)	(0/0)	(0/0)	(0/0)
solved extra	113	49	338	388	192	190
(HDR)	(74)	(37)	(170)	(189)	(192)	(14)
(LDR)	(39)	(12)	(168)	(199)	(0)	(176)
% imp. ERC	99.5	98.2	85.2	87.8	45.0	44.8
(HDR)	(99.5)	(99.5)	(78.0)	(87.0)	(90.0)	(7.0)
(LDR)	(99.5)	(97.0)	(92.5)	(88.5)	(0.0)	(82.5)
% imp. non-ERC	29.7	28.4	25.7	25.0	8.1	7.4
(HDR)	(31.1)	(32.4)	(18.9)	(20.3)	(16.2)	(0.0)
(LDR)	(28.4)	(24.3)	(32.4)	(29.7)	(0.0)	(14.9)
% avg. rel. speed-up	67.9	57.1	10.4	15.4	4.8	−25.8
(HDR)	(68.9)	(58.8)	(8.6)	(8.7)	(11.0)	(−50.9)
(LDR)	(67.0)	(55.3)	(12.1)	(22.0)	(−1.5)	(−0.8)
# faster	444	438	413	365	240	141
% speed-up if faster	84.6	71.9	14.0	24.1	12.8	50.3
# slower	104	110	135	183	308	407
% overhead if slower	2.9	1.8	0.8	1.9	1.4	26.3

HOC success led to a smaller running time compared to ℓ_1-Magic without HOC (which only reached solved status on 42 instances).

The next two row groups of Table VI give the percentages of improvements (in the sense defined by the first two row groups) achievable by including the HOC, with respect to the two methods used for constructing the corresponding instances, cf. Section 4. Note that the HOC success rate is considerably higher for the ERC-based test set than for the non-ERC part. A theoretical explanation for this is discussed below, at the end of this subsection.

The fifth group of three rows shows the average relative speed-up that HOC yields for each solver, over the whole test set or the HDR and LDR parts, respectively. The average relative speed-up is defined as

$$\frac{1}{\#\text{ instances}}\sum_i \frac{t_i^{\text{w/o HOC}} - t_i^{\text{w/ HOC}}}{t_i^{\text{w/o HOC}}};$$

it therefore incorporates the running time changes induced by HOC in both directions, i.e., not only speed-ups are considered but also overheads. The last four rows in Table VI yield the number of instances (out of 548 in the whole test set) for which the algorithm variant with HOC was faster than the one without and the relative speed-up obtained on these instances, followed by the number of instances where HOC slowed down the respective solver and the corresponding relative overhead on those instances.

From Table VI, we see that large parts of instances that a solver could also solve without HOC are solved *faster* when HOC is employed; the actual speed-ups on instances where the variant with HOC was faster are most impressive for ℓ_1-Homotopy (about 85% faster) and ISAL1 (about 72%). As already overserved earlier from Figures 4 and 5, the speed-ups are not so high for the interior-point methods ℓ_1-Magic and SolveBP/PDCO, but still quite significant. SPGL1 and YALL1 are very fast without HOC already, so it does not come as a surprise that the speed-up rate is not as high as for the other first-order methods. In fact, for YALL1, HOC is mostly ineffective on the HDR test set part and apparently also too often slows it down on the LDR part, which results in a *negative* "speed-up", i.e., an overhead. This can also be observed for SolveBP/PDCO on the LDR part: on average, HOC introduces a relative overhead of 1.5% here, while on the HDR instances there is a relative speed-up of 11%, which combines to a total average relative speed-up of almost 5%. Except for YALL1, the last two rows of the table show that the actual overheads occuring in case HOC slows down the algorithm are rather small (between 0.8 and 2.9%). Combined with the much higher speed-up rates on instances where HOC decreased the running time, we end up with average relative speed-ups for each solver (except YALL1) that are quite substantial.

It is a bit surprising that in SolveBP/PDCO, HOC is not once successful on the LDR test set part. Possibly, a different way to obtain the approximate supports used for HOC could resolve this issue, which did not seem to be a problem in any other solvers. Moreover, it is worth mentioning that SPGL1 with HOC frequency $R = \lceil m/500 \rceil$ could solve 16 more instances (especially from the non-ERC-based test set parts) than with $R = \lceil m/100 \rceil$; however, then, we end up with a relative overhead of almost 30% as opposed to the average relative speed-up of 15.4% otherwise. Since spending more time can naturally lead to more accurate solutions without employing HOC, the choice $R = \lceil m/100 \rceil$ seemed more sensible to us.

Since now, with HOC, many algorithms reach “solved” solution status quite often, we should reconsider which algorithm is “the best”. To this end, we again compare running times on instances which all solvers could now solve, see Table VII. Here, we leave out SolveBP/PDCO and YALL1 due to their problems with LDR and HDR instances, respectively. From Table III we know that with HOC, SPGL1 and ℓ_1-Magic solve about 70% of all instances, ℓ_1-Homotopy about 95%, and ISAL1 94%. (Recall that SoPlex and CPLEX both solve all instances anyway.)

The results summarized in Table VII show that by including HOC, both ℓ_1-Homotopy and SPGL1 are now clearly faster than ISAL1, which—although it also benefits from a large speed-up due to HOC—is the slowest solver on the large-scale (sparse-matrix) instances with 6144 columns or more. On the smaller instances (up to column size 4096), ℓ_1-Homotopy, SPGL1, and ISAL1 all beat the LP solvers—even the commercial CPLEX—in terms of the average running time. The interior-point code ℓ_1-Magic is not competitive: On the small instances, all other codes are much faster, and while the code clearly works better on sparse instances (i.e., the large-scale ones here), it still is the second-worst freely available algorithm here both in terms of running time and number of solved instances. Similarly, although still significantly faster than ℓ_1-Magic and SoPlex, and in some cases also CPLEX, on the smaller instances, ISAL1 is always slower than ℓ_1-Homotopy and SPGL1. From the way the running times vary with the number of columns, Table VII also shows that the running time of ℓ_1-Homotopy (with HOC) mostly seems to depend on the solution sparsity; the increase with larger column number is not nearly as significant as for the other solvers. Clearly, this is an advantage for the compressed sensing scenario, where ℓ_1-minimization is often employed to recover *sparse* solutions to $Ax = b$.

To summarize, for the test instances with at most 8192 columns, ℓ_1-Homotopy with HOC integrated into it solves almost all instances, and does so much faster than all non-commercial alternatives and in most cases also faster than CPLEX. On the larger instances, which mostly have sparse matrices, CPLEX solves all instances and is the fastest algorithm. However, both SoPlex and ℓ_1-Homotopy (with HOC) are only fractions of a second slower, solving all or almost all instances, respectively, and therefore offer the best freely available codes in this regime.

It is noteworthy that this situation does not change when we only ask for acceptable solutions: ℓ_1-Homotopy with HOC is faster than all other solvers for instances up to column size 8192 (the next fastest are (usually, but not always) SPGL1 and ISAL1); beyond that size, CPLEX is fastest, followed by SoPlex and then again ℓ_1-Homotopy with HOC. Both ℓ_1-Magic and SolveBP/PDCO turn out to not be competitive even with HOC (although ℓ_1-Magic ends up in fourth place on the large-scale instances).

Table VII. Geometric means of the running times (in seconds) of selected algorithms *with* HOC, for instances that all could solve, grouped by number n of columns.

n	used/of		ℓ_1-Hom.	ISAL1	ℓ_1-Magic	SPGL1	CPLEX	SoPlex
1024	61/72	solved	71	70	61	62	72	72
		avg. t	0.07	0.35	1.31	0.12	0.34	0.78
		max t	1.16	1.81	2.01	1.08	2.56	15.66
1536	51/72	solved	69	68	54	53	72	72
		avg. t	0.05	0.44	2.06	0.13	0.58	1.17
		max t	1.20	1.35	3.10	1.17	4.55	34.49
2048	103/144	solved	139	134	108	107	144	144
		avg. t	0.11	1.02	5.03	0.26	1.36	3.27
		max t	1.26	2.59	14.69	4.09	17.24	90.14
3072	48/72	solved	65	65	52	50	72	72
		avg. t	0.19	2.36	14.84	0.52	3.95	9.33
		max t	1.69	12.26	26.67	6.60	43.36	113.40
4096	59/94	solved	87	88	64	65	94	94
		avg. t	0.18	3.48	9.42	0.58	1.64	3.54
		max t	1.01	30.98	30.03	7.92	60.89	204.03
6144	8/16	solved	16	15	8	11	16	16
		avg. t	0.07	6.90	0.23	0.36	0.02	0.03
		max t	0.44	25.63	0.82	0.97	0.03	0.05
8192	9/22	solved	20	20	10	11	22	22
		avg. t	0.15	15.07	4.07	1.17	0.21	0.31
		max t	0.42	67.41	72.54	8.77	2.17	2.96
12288	0/4	solved	4	4	2	1	4	4
		avg. t	–	–	–	–	–	–
		max t	–	–	–	–	–	–
16384	5/16	solved	14	16	8	11	16	16
		avg. t	0.41	21.41	1.09	0.59	0.12	0.20
		max t	1.85	32.05	6.96	1.59	0.26	0.40
24576	6/16	solved	16	16	7	8	16	16
		avg. t	0.29	68.31	1.43	2.79	0.17	0.27
		max t	0.35	185.28	4.24	6.18	0.31	0.46
32768	4/16	solved	16	15	5	7	16	16
		avg. t	0.63	122.13	4.53	3.85	0.35	0.56
		max t	1.75	135.06	8.42	8.16	0.36	0.56
49152	1/4	solved	4	4	1	2	4	4
		avg. t	0.65	220.72	6.27	4.79	0.44	0.68
		max t	0.65	220.72	6.27	4.79	0.44	0.68

ERC-based HOC success. As mentioned above, the observation that HOC is more often successful on instances constructed using the ERC can be explained by the following result.

THEOREM 5.1. *Let the support S^* of an optimal solution x^* of* (P$_1$) *obey the ERC* (8). *Then $w^* = -(A_{S^*}^\top)^\dagger \operatorname{sign}(x_{S^*}^*)$ is an optimal dual solution. Hence, in exact arithmetic, HOC applied to (A, b, x), using $\hat{w} = (A_S^\top)^\dagger \operatorname{sign}(x_S)$, returns success with $\hat{x} = x^*$, if $\operatorname{sign}(x_{S^*}) = \operatorname{sign}(x_{S^*}^*)$ and either the support was correctly estimated ($S = S^*$), or S contains S^*, obeys the ERC, and A_S has full column rank.*

PROOF. Let $S = S^*$ and $\operatorname{sign}(x_{S^*}) = \operatorname{sign}(x_{S^*}^*)$. For an arbitrary $j \notin S^*$, we have

$$|A_j^\top (A_{S^*}^\top)^\dagger \operatorname{sign}(x_{S^*}^*)| \leqslant \|A_j^\top (A_{S^*}^\top)^\dagger\|_1 \, \|\operatorname{sign}(x_{S^*}^*)\|_\infty \leqslant \operatorname{ERC}(A, S^*) < 1. \qquad (10)$$

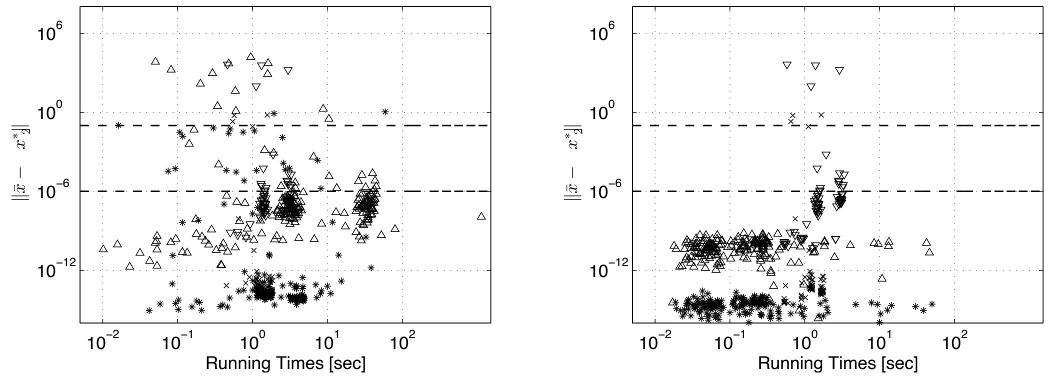


Fig. 11. Results for ℓ_1-Homotopy, modified to terminate as soon as a problem is recognized. The plot shows running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

Hence, $w^* = -(A_{S^*}^\top)^\dagger \operatorname{sign}(x^*_{S^*}) = -\hat{w}$ fulfills the optimality condition in Lemma 2.1. Moreover, $\hat{x}$ is feasible by construction and hence is optimal. Finally, $\hat{x}$ equals x^* since the ERC guarantees uniqueness of the solutions.

Now assume $\operatorname{sign}(x_{S^*}) = \operatorname{sign}(x^*_{S^*})$, $S \supset S^*$ with $\operatorname{rank}(A_S) = |S| \leqslant m$, and S obeys the ERC. Then, analogously to (10), one can see that $\hat{w} = (A_S^\top)^\dagger \operatorname{sign}(x_S)$ yields the dual feasible solution $-\hat{w}$. Since A_S has full column rank, $\hat{x}$ is the unique solution to $A_S\hat{x}_S = b$ with zero entries at positions $j \notin S$. Then $\hat{x}$ must also have zeros at $j \in S\backslash S^*$ (since x^* with support $S^* \subset S$ is primal feasible). Hence, since S^* obeys the ERC, $\hat{x} = x^*$. Moreover, because a subgradient at $\hat{x}$ can assume any value in $[-1, 1]$ for components in which $\hat{x}$ is 0, $A^\top\hat{w} \in \partial\|\hat{x}\|_1$. Thus, as the duality-gap vanishes ($\|\hat{x}\|_1 - b^\top\hat{w} = 0$), HOC returns success. $\square$

Note that the ERC guarantees the existence of $\epsilon > 0$ such that the magnitude of $(A^T w^*)_j$ is smaller than $(1-\epsilon)$ for j outside of the support of x^*, and hence, some inaccuracy in the calculation of w^* is tolerated. Generally however, there is no guarantee that w^* yields the desired properties. Therefore, the approach taken by HOC (i.e., working with $\hat{w} \approx w^*$) does not need to be effective, as is reflected by the lower success rates on the second part of the test set, where the ERC does not hold. Nevertheless, even without a theoretical justification of the choice of $\hat{w}$, HOC still works in several cases on this test set part as well.

Moreover, from the observation that sufficient sparsity with respect to mutual coherence implies the ERC on the respective support—see [Fuchs 2004, Theorem 3]—we immediately deduce the following result.

Corollary 5.2. *Let x^* be an optimal solution of* (P$_1$) *with $\|x^*\|_0 < \frac{1}{2}\big(1 + \frac{1}{\mu(A)}\big)$, where $\mu(A)$ is the mutual coherence of A. If HOC is used with the correct support and exact calculations, then it returns success if and only if the outcome $\hat{x}$ equals x^*.*

5.3. The Behavior of ℓ_1-Homotopy

As mentioned above, the unconvincing behavior of ℓ_1-Homotopy is somewhat unexpected. One reason could be a large coherence among dictionary columns leading to (many) more breakpoints encountered along the homotopy solution path. However, ℓ_1-Homotopy produced unacceptable solutions for only 39 of the 114 instances with highly coherent dictionaries (HDR: 26/57, LDR: 13/57), while 52 of the remaining 75 instances (HDR: 20/31, LDR: 32/44) were solved (in the sense defined by (9)). Thus, this intuitive explanation cannot be verified.

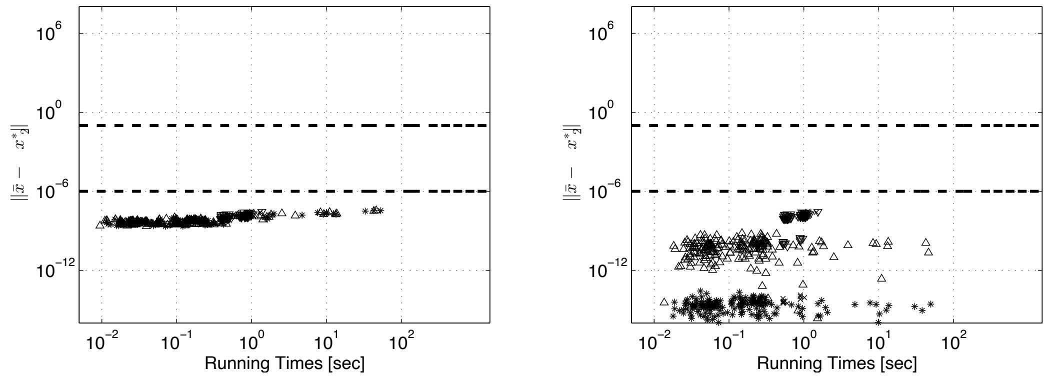


Fig. 12. Results for ℓ_1-Homotopy with final regularization parameter 10^{-9}. The plot shows running times versus distance to optimum in loglog scale. Left: without HOC. Right: with HOC. ERC-based instances: $\triangle$ (HDR), $\ast$ (LDR). Non-ERC-based instances: $\triangledown$ (HDR), $\times$ (LDR).

The implementation identifies two problematic cases (sign mismatches and degeneracy), but ignores them for up to 500 more iterations. Terminating the algorithm as soon as one of these cases occurs yields the results depicted in Figure 11. Compared to Figure 3, the improvement is clear, even without incorporating HOC (now only 13 of the 114 highly coherent instances yield unacceptable solutions (HDR: 11, LDR: 2), and 101 (HDR: 46, LDR: 55) are solved.) Nevertheless, the general impression remains the same: One would expect the homotopy method to perform better, in particular, not to deliver unacceptable results. Thus, we suspect numerical issues to be the reason for this unexpected bad performance.

Moreover, theory only guarantees convergence of the homotopy method if the final regularization parameter is set to 0. However, if we use 10^{-9} instead of 0, we get a completely different picture, see Figure 12, and compare with Figures 3 and 11. It turns out that ℓ_1-Homotopy with final regularization parameter 10^{-9}—although technically not an *exact* ℓ_1-solver—solves all instances with high accuracy and is also the fastest algorithm in the vast majority of cases. It is noteworthy that, in this case, the final results are actually worse if we terminate as soon as the implementation detects a problematic case. Moreover, HOC apparently does not improve the general performance and does not convincingly justify its induced running time overhead, although it is small.

6. CONCLUDING REMARKS

Our experiments do not indicate a clear winner among the various exact ℓ_1-solvers. While CPLEX and SoPlex were the only ones to solve all instances to within distance 10^{-6} of the true optimum, they have the potential drawback of requiring the matrices in explicit form, while all other algorithms could also handle matrices given implicitly via matrix-vector product oracles. Moreover, CPLEX is proprietary, while all other implementations are freely available.

Currently, only the ISAL1 implementation contains the Heuristic Optimality Check (HOC) by default. However, the results clearly indicate that this is a good idea for all solvers (with the possible exception of YALL1), producing more accurate solutions and typically reducing the running times at the same time. Considering the various implementations as they actually can be downloaded to date, ISAL1 (containing HOC) may be a good alternative to CPLEX and SoPlex when small- to medium-scale are to be solved. However, once HOC is also integrated into the other solvers, ℓ_1-Homotopy becomes the best code on such instances, followed closely by SPGL1. For large-scale instances with sparse matrices, the two LP solvers fare best; however, ℓ_1-Homotopy with HOC is very close behind.

The average performance of ℓ_1-Homotopy could be improved significantly by a slight modification of the code and by integrating HOC. Deviating from theoretical requirements, ℓ_1-Homotopy with final regularization parameter 10^{-9} instead of 0 in fact turned out to be the fastest reliable solver. With this parameter choice, using HOC does not really seem beneficial (the accuracy is slightly improved and a slight overhead is introduced). On the other hand, we also observed from a few experiments, that using an even larger regularization parameter (e.g., 10^{-4}) still allowed for extracting the optimal support from the point produced by ℓ_1-Homotopy by a simple hard-thresholding scheme like (1); a theoretical justification is given by the results in [Tropp 2006; Lorenz et al. 2011b] on exact support recovery. This suggests that it may, in principle, be worth integrating HOC into ℓ_1-Homotopy. In particular, this might hold true for a variant in which the homotopy path is not followed exactly but only approximately, see [Mairal and Yu 2012], which might avoid some numerical problems of the homotopy method.

ACKNOWLEDGMENTS

We thank the anonymous referees for their useful comments and suggestions, which helped improve this paper.

REFERENCES

M. Salman Asif. 2008. *Primal Dual Pursuit—A Homotopy Based Algorithm for the Dantzig Selector*. Master's thesis. Georgia Institute of Technology.

Stephen Becker, Jérôme Bobin, and Emmanuel J. Candès. 2011. NESTA: A Fast and Accurate First-Order Method for Sparse Recovery. *SIAM Journal on Imaging Sciences* 4, 1 (2011), 1–39.

Ernesto G. Birgin, José M. Martínez, and Marcos Raydán. 2000. Nonmonotone Spectral Projected Gradient Methods on Convex Sets. *SIAM Journal on Optimization* 10, 4 (2000), 1196–1211.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, US-NY.

Jian-Feng Cai, Stanley Osher, and Zuowei Shen. 2009. Linearized Bregman iterations for compressed sensing. *Math. Comp.* 78, 267 (2009), 1515–1536.

Paolo M. Camerini, Luigi Fratta, and Francesco Maffioli. 1975. On improving relaxation methods by modified gradient techniques. *Mathematical Programming Study* 3 (1975), 26–34.

Emmanuel J. Candès, Justin Romberg, and Terence Tao. 2006. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory* 52, 2 (2006), 489–509.

Giacomo D'Antonio and Antonio Frangioni. 2009. Convergence Analysis of Deflected Conditional Approximate Subgradient Methods. *SIAM Journal on Optimization* 20, 1 (2009), 357–386.

Geoffrey M. Davies, Stéphane G. Mallat, and Zhifeng Zhang. 1994. Adaptive time-frequency decompositions. *Optical Engineering* 33, 7 (1994), 2183–2191.

David L. Donoho. 2006. Compressed Sensing. *IEEE Transactions on Information Theory* 52, 4 (2006), 1289–1306.

David L. Donoho and Xiaoming Huo. 2001. Uncertainty Principles and Ideal Atomic Decomposition. *IEEE Transactions on Information Theory* 47, 7 (2001), 2845–2862.

David L. Donoho and Yaakov Tsaig. 2008. Fast Solution of ℓ_1-Norm Minimization Problems When the Solution May Be Sparse. *IEEE Transactions on Information Theory* 54, 11 (2008), 4789–4812.

Junbo Duan, Charles Soussen, David Brie, Jérôme Idier, and Yu-Ping Wang. 2011. A sufficient condition on monotonic increase of the number of nonzero entry in the optimizer of L1 norm penalized least-square problem. (2011). arXiv:1104.3792 [stat.ML].

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least angle regression. *The Annals of Statistics* 32, 2 (2004), 407–499.

Mário A. T. Figueiredo, Robert D. Nowak, and Stephen J. Wright. 2007. Gradient Projection for Sparse Reconstruction: Applications to Compressed Sensing and Other Inverse Problems. *IEEE Journal of Selected Topics in Signal Processing* 4 (2007), 586–597. Issue 1.

Jean-Jacques Fuchs. 2004. On Sparse Representations in Arbitrary Redundant Bases. *IEEE Transactions on Information Theory* 50, 6 (2004), 1341–1344.

Markus Grasmair, Markus Haltmeier, and Otmar Scherzer. 2011. Necessary and sufficient conditions for linear convergence of ℓ^1-regularization. *Communications on Pure and Applied Mathematics* 64, 2 (2011), 161–182.

Michael Held, Philip Wolfe, and Harlan P. Crowder. 1974. Validation of subgradient optimization. *Mathematical Programming* 6 (1974), 62–88.

Victor Klee and George J. Minty. 1972. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, CA, 1969; dedicated to the memory of Theodore S. Motzkin)*. Academic Press, New York, US-NY, 159–175.

Torbjörn Larsson, Michael Patriksson, and Ann-Brith Strömberg. 1996. Conditional Subgradient Optimization – Theory and Applications. *European Journal of Operations Research* 88, 2 (1996), 382–403.

Churlzu Lim and Hanif D. Sherali. 2005. Convergence and Computational Analyses for Some Variable Target Value and Subgradient Deflection Methods. *Computational Optimization and Applications* 34, 3 (2005), 409–428.

Andreas Löbel. 1997. *Optimal Vehicle Scheduling in Public Transit*. Ph.D. Dissertation. Technische Universität Berlin.

Dirk A. Lorenz. 2013. Constructing test instances for Basis Pursuit Denoising. *IEEE Transactions on Signal Processing* 61, 5 (2013), 2010–2014.

Dirk A. Lorenz, Marc E. Pfetsch, and Andreas M. Tillmann. 2011a. An Infeasible-point Subgradient Method Using Adaptive Approximate Projections. (2011). arXiv:1104.5351 [math.OC].

Dirk A. Lorenz, Stefan Schiffler, and Dennis Trede. 2011b. Beyond convergence rates: Exact inversion with Tikhonov regularization with sparsity constraints. *Inverse Problems* 27 (2011), 085009.

Julien Mairal and Bin Yu. 2012. Complexity analysis of the Lasso regularization path. (2012). arXiv:1205.0079 [stat.ML].

Dmitry M. Malioutov, Müjdat Çetin, and Alan S. Willsky. 2005. Homotopy Continuation for Sparse Signal Representation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05)*, Vol. 5. Institute of Electrical and Electronics Engineers (IEEE), New York, US-NY, 733–736.

Arkadi S. Nemirovskiy and Boris T. Polyak. 1984. Iterative methods for solving linear ill-posed problems under precise information. I. *Izvestiya Akademii Nauk SSSR. Tekhnicheskaya Kibernetika* 2 (1984), 13–25, 203.

Yurii Nesterov. 2005. Smooth minimization of non-smooth functions. *Mathematical Programming* 103, 1 (2005), 127–152.

Michael R. Osbourne, Brett Presnell, and Berwin A. Turlach. 2000. A new approach to variable selection in least squares problems. *IMA J. Numer. Anal.* 20, 3 (2000), 389–402.

Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. 1993. Orthogonal Matching Pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annual Asilomar Conference on Signals, Systems and Computers*, Vol. 1. Pacific Grove, US-CA, 40–44.

Andrzej Ruszczyński. 2006. *Nonlinear Optimization*. Princeton University Press, Princeton, US-NJ.

Naum Z. Shor. 1985. *Minimization Methods for Non-Differentiable Functions*. Springer, New York, US-NY.

Joel A. Tropp. 2004. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory* 50, 10 (2004), 2231–2242.

Joel A. Tropp. 2006. Just relax: Convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory* 52, 3 (March 2006), 1030–1051.

Joel A. Tropp and Anna C. Gilbert. 2007. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory* 53, 12 (2007), 4655–5666.

Ewout van den Berg and Michael P. Friedlander. 2008. Probing the Pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing* 31, 2 (2008), 890–912.

Yilun Wang and Wotao Yin. 2010. Sparse signal reconstruction via iterative support detection. *SIAM Journal on Imaging Sciences* 3, 3 (2010), 462–491.

Zaiwen Wen, Wotao Yin, Donald Goldfarb, and Yin Zhang. 2010. A fast algorithm for sparse reconstruction based on shrinkage, subspace optimization and continuation. *SIAM Journal on Scientific Computing* 32, 4 (2010), 1832–1857.

Roland Wunderling. 1996. *Paralleler und objektorientierter Simplex-Algorithmus*. Ph.D. Dissertation. Technische Universität Berlin. In German.

Allen Y. Yang, Zihan Zhou, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. 2010. A Review of Fast ℓ_1-Minimization Algorithms for Robust Face Recognition. (2010). arXiv:1007.3753 [cs.CV].

Junfeng Yang and Yin Zhang. 2011. Alternating direction algorithms for ℓ_1-problems in compressive sensing. *SIAM Journal on Scientific Computing* 33, 1 (2011), 250–278.

Wotao Yin. 2010. Analysis and generalizations of the linearized Bregman model. *SIAM Journal on Imaging Sciences* 3, 4 (2010), 856–877.

Wotao Yin, Stanley J. Osher, Donald Goldfarb, and Jérôme Darbon. 2008. Bregman Iterative Algorithms for ℓ^1-Minimization with Applications to Compressed Sensing. *SIAM Journal on Imaging Sciences* 1, 1 (2008), 143–168.