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

Exact Optimization for the l^1-Compressive Sensing problem using a Modified Dantzig-Wolfe method

Alexandre Borghi · Jérôme Darbon · Sylvain Peyronnet

December 2, 2009

Abstract This paper considers the l^1-Compressive Sensing problem and presents an efficient algorithm that computes an exact solution. The idea consists in reformulating the problem such that it yields a modified Dantzig-Wolfe decomposition that allows to efficiently apply all standard simplex pivoting rules. Experimental results show the superiority of our approach compared to standard linear programming methods.

Keywords Compressive Sensing, Dantzig-Wolfe Decomposition, Pivoting Rules

1 Introduction

This paper describes an efficient algorithm that computes an exact solution of ℓ_1-Basis Pursuit/Compressive Sensing problems. It relies on a modification of the Dantzig-Wolfe approach that greatly improves performances compared to the original standard approach.

The ℓ_1-Basis Pursuit problem [15] corresponds to minimizing the ℓ_1-norm of a signal under some linear constraints. This problem has originally been used for decomposing a signal using few atoms of a predefined dictionary [15,40]. More recently this problem has received a lot of attention because of signal sampling through Compressive Sensing (CS). Following the seminal works of [23] and [11–13,45], the principle of Compressive Sensing states that a sparse signal in a chosen basis can be recovered through a ℓ_1-Basis Pursuit optimization problem. Formally it corresponds to solving the following problem:

$$(CS) \begin{cases} \min_u \|u\|_1 \\ s.t.\ Au = f \end{cases} \tag{1}$$

where $f \in \mathbb{R}^m$, are the observed data, $u \in \mathbb{R}^n$ is the signal of interest to reconstruct and the matrix $A \in \mathbb{R}^{m \times n}$ models the linear constraints. We refer the reader to [2,11,13,22,23] for the design of Compressive Sensing matrices. Let us note that such matrices are dense and yields a problem where the number of constraints m is typically much lower than the size of the signal n, i.e., $m \ll n$.

There is a vast literature of algorithms that *approximately* solve the CS problem [17,27,33,34,37, 44,46,45,51] for instance. Most of these approaches rely on iterative based thresholding/shrinkage

Research of J. Darbon has been supported by the Office of Naval Research through grant N000140710810.

A. Borghi and S. Peyronnet
Laboratoire de Recherche en Informatique (LRI), Université Paris- Sud XI, France.
E-mail: {borghi,syp}@lri.fr

J. Darbon
CMLA, ENS Cachan, CNRS, PRES UniverSud, France.
E-mail: darbon@cmla.ens-cachan.fr

that has been originally presented in [42] and [14] and with further developments and improvements such as [5,10,17,21,25,26]. These approaches lead to fast algorithms that compute approximate solution. Another class of approaches consists in considering greedy based algorithms [16, 47,48]. We refer the reader to [49] for a deep review of approximate computational methods for solving the CS problem.

Contrary to the above approaches, this paper presents an efficient algorithm that computes an *exact* solution to the CS problem. The idea consists in reformulating the problem as a linear one that allows to apply a standard decomposition method based on the simplex [19,20,39] known as the Dantzig-Wolfe decomposition. We further refine and improve this approach, so that all standard simplex pivoting rules can be *efficiently* applied. Dantzig-Wolfe decomposition relying on interior points methods has been studied by [41,52]. However, interior point methods are not well-tailored to problems with dense constraint matrix [1] like those we find in our case. This has led us to choose the standard Dantzig-Wolfe method as a starting point.

The remainder of this paper is as follows. We describe in section 2 the standard decomposition method. The latter is refined for the CS problem in section 3. This approach is further improved in section 4. Some experimental results along with comparisons to the simplex algorithm are presented in section 5. Eventually we draw some conclusions in section 6.

2 Standard Dantzig-Wolfe decomposition

In this section we introduce the Dantzig-Wolfe decomposition [19,20,39], which relies on delayed column generation to solve large scale linear programs. The main idea of Dantzig-Wolfe decomposition is to reformulate a linear program with a block-angular constraint matrix as an equivalent linear program. This equivalent program is called the Master Program (MP) and its formulation expresses the fact that a solution is a convex combination of extreme points of the polyhedron defined by the initial program constraints. The number of columns of MP is equal to the number of extreme points of the polyhedron defined by the original constraint matrix. A two-level iterative process is used to solve efficiently this extremely large linear program. At each iteration, a restricted MP with only a subset of columns is solved to find optimal dual simplex multipliers. The latter are then used to generate a negative reduced cost column. If no such column exists, optimality is reached and therefore an exact solution is found [20]. Dantzig-Wolfe decomposition has not been widely used for several reasons: it can be slower than standard simplex resolution, it suffers from the well-known tail effect, i.e., slow improvement at the end of the process, and it can be difficult to implement efficiently. However, some problems clearly benefit from this approach [24,39,43]. For instance, the Dantzig-Wolfe approach has been recently used with great interest by the Branch-and-Price community to improve performances of integer programming resolution [3,4,50].

The Dantzig-Wolfe decomposition arises from the reformulation of an initial linear program. The initial linear program has variables $x \in \mathbb{R}^n_+$, objective function coefficients $c \in \mathbb{R}^n$, coupling constraints $Ax = b$ where A is a $m \times n$ matrix and l independent angular blocks $D_{(i)}$ of constraints on vector variables $x_i \in \mathbb{R}^{k_i}_+$ where $n = \sum_{i=1}^{l} k_i$. Then, $x = (x_1, \ldots, x_l)^t$ and $c = (c_1, \ldots, c_l)^t$. This reformulated program writes as

$$\begin{cases} \min_x cx \\ s.t.\ Ax = b \\ \quad D_{(i)}x_i = b_i \\ x \geq 0 \end{cases} \tag{2}$$

while the overall constraint matrix form is

$$\begin{pmatrix} A_{(1)} & \cdots & A_{(l)} \\ D_{(1)} & & \\ & \ddots & \\ & & D_{(l)} \end{pmatrix}$$

We denote by $x(i)$ the i^{th} component of the vector x and by $A_{(i)}$ the block of columns of the matrix A that corresponds to the vector x_i. From the initial linear program (2), the Dantzig-Wolfe reformulation gives an equivalent linear program where variable vector is u with $\forall i, u(i) \in \mathbb{R}_+$, called the Master Program of the Dantzig-Wolfe decomposition [20]:

$$(Master\ Program) \begin{cases} \min_u \sum_j \theta(j)u(j) \\ s.t.\ \sum_j \gamma_j u(j) = b \\ \quad \sum_j u(j) = 1 \\ u(j) \geq 0 \end{cases} \tag{3}$$

where $\gamma_j = Ap_j$ and $\theta(j) = cp_j$, with $p_j \in \mathbb{R}^n$ the extreme points of the polyhedron defined by the constraints of the non-coupling linear program (2) (i.e., the constraints $D_{(i)}x_i = b_i$). Note that the constraint $\sum_j u(j) = 1$ with $\forall j, u(j) \geq 0$ in program (3) is a convexity constraint. It means that a solution of the initial linear program (2) is a convex combination of the extreme points p_j.

The reader should note that the linear program (3) has as many variables as the linear program (2) has extreme points, and has one additional constraint. The number of variables of the program (3) is therefore exponential with respect to the number of variables of program (2). Instead of using the simplex method on program (3), which would take far more time than a direct resolution of the original linear program (2), a column generation approach is used.

Let us now describe the column generation process. Let S_i be the convex bounded polyhedron defined by:

$$S_i = \{x_i | D_{(i)}x_i = b_i, x_i \in \mathbb{R}_+^{k_i}\}\ .$$

The set S is the union of all S_i and the set of all extreme points of program (2) denoted by p_j. It forms a bounded convex polyhedral set.

Let μ be the simplex multiplier associated with the convexity constraint of program (3) (i.e., $\sum_j u(j) = 1$) and π the vector of simplex multipliers associated with the remaining constraints. To choose a variable $u(j)$ to enter the basis (i.e., a new column to generate from an extreme point p_j) the usual Dantzig rule is considered. The latter corresponds to choose the variable with minimum reduced cost. In our formulation, it means that the variable $u(j)$ corresponds to the solution p_j of

$$\min_{p_j}(c - \pi A)p_j - \mu\,. \tag{4}$$

Assuming that program (4) is a bounded optimization problem, its optimal solutions are extreme points of its feasible set and is therefore equivalent to

$$\begin{cases} \min_x \langle c - \pi A, x\rangle \\ s.t.\ D_{(i)}x_i = b_i, \forall i \\ x_i \geq 0, \forall i \end{cases} \tag{5}$$

The solution of problem (5) is the extreme point corresponding to the variable that will enter the basis. The usual simplex pivot is performed to find the variable that leaves the basis. The objective function of program (5) is additively separable and its constraints are independent. Therefore, problem (5) can be transformed into l independent subproblems. It yields

$$\forall i \in \{1, \ldots, l\}, \begin{cases} \min_{x_i} \langle c_i - \pi A_{(i)}, x_i\rangle \\ s.t.\ D_{(i)}x_i = b_i \\ x_i \geq 0 \end{cases} \tag{6}$$

The fact that the l subproblems are independent is of utmost importance since it enables to solve them in parallel without any communication. At each iteration, the subproblems (6) can be generally computed in polynomial time. Therefore it yields an element of an exponential-sized set in polynomial time.

The following algorithm performs the Dantzig-Wolfe decomposition and solves the original problem (2) through the reformulated problem (3):

Dantzig-Wolfe Algorithm [20]

Assume that an initial basic feasible solution of MP is known. We denote the associated basic matrix by B, the basic inverse matrix by B^{-1} and the simplex multipliers by (π, μ). We denote by A_i is the i^{th} column of the matrix A.

1. Solve the l subproblems (6) using the simplex multipliers π. This yields the optimal solutions $\hat{x}(i)$ and the optimal objective values of these l problems are $w(i)$.
2. Compute $w = \sum_{i=1}^{n} w(i) - \mu$
3. If $w \geq 0$, optimal solution is found. The optimal solution of problem (2) is then $\sum_{u(j)>0} u(j)p_j$.
4. If $w < 0$, the new column to enter the basis is:

$$\begin{pmatrix} \sum_i A_i \hat{x}(i) \\ 1 \end{pmatrix}$$

5. After multiplication by B^{-1}, apply the usual simplex pivot operation to obtain a variable to leave the basis. A new basic matrix B is then obtained and its associated inverse basic matrix B^{-1} can be computed as well as new simplex multipliers (π, μ).
6. Repeat from step 1.

Note that since the new basic matrix differs by only one column from an iteration to another, B^{-1} can be computed by multiplying a square matrix η with the previous inverse basic matrix. Such a matrix η is formed by the sum of a unit matrix of size $(m+1) \times (m+1)$ and a one column matrix of same size. Because of this particular form, the multiplication can be done in $O(m^2)$. We refer the reader to [19] for further details of this standard process. The overall complexity of an iteration is $O(mn)$.

If MP is nondegenerate, each iteration strictly decreases the objective function value. There is a finite number of possible bases and none is repeated. Therefore the method leads to the optimal solution in a finite number of iterations. The solution is then expressed as a convex combination of extreme points of the linear program (2).

3 Dantzig-Wolfe decomposition for Compressive Sensing

In this section, we aim to reformulate the Compressive Sensing problem to obtain a linear program whose constraint matrix is block-angular and whose subproblems solutions form a bounded convex polyhedral set. This will allow us to apply the decomposition method described in section 2.

The formulation (1) of the Compressive Sensing problem that can be rewritten as the following linear program [8]:

$$(Reformulated\, CS) \begin{cases} \min_y \langle y, \mathbb{1} \rangle \\ s.t.\ Ax = f \\ \quad x \leq y \\ -x \leq y \\ x \in \mathbb{R}^n, y \in \mathbb{R}^n_+ \end{cases} \tag{7}$$

As explained in section 2, the usual Dantzig-Wolfe decomposition assumes that the solution spaces of induced subproblems are bounded convex polyhedral sets. However, in our peculiar case this assumption does not hold since S is unbounded. This means that the method cannot be directly used on problem (7). General l^1 (and l^∞) minimization problems are prone to issues when using decomposition methods. Indeed, $x(i)$ variables are free (and $y(i)$ non-negative), therefore the notion of extreme points is senseless. However, $Ax = f$ can be solved to obtain an initial

solution x^0. Thus a bound K on variables $y(i)$ can be set to $||x^0||_1$. In the case of l^∞ minimization, K can be set to $||x^0||_\infty$. Let us note that one could define tighter bounds by setting a bound for each variable $y(i)$. This yields a linear program with bounded variables. Besides, the extreme points of the constraints are naturally defined.

Let us reorder variables $x(i)$ and $y(i)$ as $(x(0), y(0), x(1), y(1), \ldots, x(n), y(n))$. This yields a constraint matrix of the following form:

$$\begin{pmatrix} A_{1,1} & 0 \ldots & A_{1,n} & 0 \\ \vdots & \vdots & \vdots & \vdots \\ A_{m,1} & 0 \ldots & A_{m,n} & 0 \\ 1 & -1 & & \\ -1 & -1 & 0 & \\ 0 & 1 & & \\ & \ddots & & \\ & & 1 & -1 \\ 0 & & -1 & -1 \\ & & 0 & 1 \end{pmatrix}$$

This constraint matrix has a block-angular structure, which is mandatory to use the Dantzig-Wolfe decomposition (see section 2). This matrix is formed by the coupling constraint $A^{'}$ and n identical angular blocks of constraints. The matrix $A^{'}$ is the matrix A in the space of the reordered variables. Each block of constraints incorporates a bound constraint on its variables.

Following the reformulation (3) (see section 2), we have $\gamma_j \in \mathbb{R}^{2n}$ and $\theta \in \mathbb{R}^{2n}$ such that $\gamma_j = A^{'} p_j$ and $\theta(j) = cp_j = \sum_i p_j(i)$ (simplified thanks to the l^1-norm, which implies $\forall i \in \{1, \ldots, n\}, c(2i-1) = 0$ and $c(2i) = 1$). It yields the following program with variables $u(j) \in \mathbb{R}^{2n}$:

$$\begin{cases} \min_u \sum_j \theta(j) u(j) \\ s.t. \ \sum_j \gamma_j u(j) = f \\ \quad \sum_j u(j) = 1 \\ u_j \geq 0 \end{cases} \tag{8}$$

Let us now describe the column generation process for the Compressive Sensing problem. Let S_i be the convex bounded polyhedron set that corresponds to the i^{th} angular block of constraints, i.e.,

$$S_i = \{x(i), y(i) | x(i) - y(i) \leq 0, x(i) + y(i) \geq 0, y(i) \leq K, x(i) \in \mathbb{R}, y(i) \in \mathbb{R}_+\} \ .$$

The set L is the union of all S_i and forms a bounded convex polyhedral set.

In our case of bounded l^1-minimization, the n independent linear programs (6) that corresponds to the subproblems of the decomposition simplify to the following formula:

$$\forall i \in \{1, \ldots, n\}, \begin{cases} (0,0) & \text{if } |\bar{x}(i))| \leq 1, \\ (-sg(\bar{x}(i)) \times K, K) & \text{otherwise} \end{cases} \tag{9}$$

where $\bar{x}(i)$ is the reduced cost of the variable $x(i)$ and $sg(x) = 1$ if $sg(x) \geq 0$, -1 otherwise. Formula (9) can output 3 different values per subproblem: $(0,0), (-K, K), (K, K)$. Since there are n subproblems to solve at each iteration, the solution space L has 3^n elements. The vectors of L are denoted by p_j. Note that the solution given by formula (9) is computed at each iteration in linear time (n independent constant time subproblems). This improvement is specific to Compressive Sensing and let us recall that it takes in general polynomial time to compute such a solution (see section 2). The formulation (7) has a $(m+2n) \times (2n)$ sparse constraint matrix and the formulation (8) has a dense $(m+1) \times (3^n)$ constraint matrix.

Since $y(i)$ variables are only used to express the l^1-norm of the objective function as a linear expression, only variables $x(i)$ are interesting at the constraint level. Thus, the convex bounded polyhedron set that corresponds to the n block of constraints can be rewritten as follows:

$$S_i = \{x(i) | -K \leq x(i) \leq K, x(i) \in \mathbb{R}\} \ .$$

Formula (9) now defines the set L as $L = \{-K, 0, K\}^n$. It yields that p_j and γ_j lives in $\mathbb{R}^n$ instead of $\mathbb{R}^{2n}$. For this, γ_j is now defined as $\gamma_j = Ap_j$. Since $p_j \in \mathbb{R}^n$, the expression of $\theta(j)$ must be adapted as well. The components of c, i.e., the coefficients of the objective function, are equal to 0 for variables $x(i)$ and 1 for variables $y(i)$, and $y(i) = |x(i)|$. With $p_j \in \mathbb{R}^n$, it yields $\theta(j) = \sum_i |p_j(i)|$. From now on, we assume that $\forall j, p_j \in \mathbb{R}^n, \theta(j) = \sum_i |p_j(i)|$ and $\gamma_j = Ap_j$.

Like many linear programs, our linear program (2) is degenerate. It simply comes from the fact that the optimal solution is degenerate because it has less than m nonzeros, with m the number of constraints. It implies that the algorithm may cycle, i.e., might not converge. Situations where cycling occurs have been studied in [31]. In practice, given a nondegenerate initial solution for problem (3), degeneracy seems to only appear when optimality is reached. The energy strictly decreases until the optimal solution is found. However convergence is not detected and the process does not stop by itself (i.e., the condition $w \geq 0$, in step 3 of the Dantzig-Wolfe Algorithm of section 2, never happens because of cycling) although the objective function value does not change anymore. We overcome this limitation by adding a test on the strict decrease of the energy. If this test is satisfied, Karush-Kuhn-Tucker (KKT) conditions [36,38] can therefore be checked to know if the energy will still decrease. In practice the first test seems to always find optimal solutions without needing more costly techniques such as checking KKT conditions.

Let us now briefly describe how to compute an initial basic feasible solution for problem (8). An initial basic feasible solution of problem (8) can be computed from a basic feasible solution of $Ax = f$. Indeed, each solution of $Ax = f$ can be represented as a solution u of $\sum_i (Ap_i)u(i) = f$ with $\sum u(i) = 1$ and $\forall i = 1, \ldots, 3^n, u(i) \geq 0$, i.e., a solution of the Dantzig-Wolfe decomposition. Solving $Ax = f$ is usually done by introducing m slack variables $z(i)$ (one per constraint) and minimizing $\sum_i z_i$. A simplex can therefore be used to solve this problem whose initial basic solution is $z = f$ and $x = (0, \ldots, 0)^t$. As we know that $Ax = f$ has at least one solution, we can solve $Ax = f$ by introducing a unique slack variable for all constraints. Using a unique slack variable to solve $Ax = f$ instead of m is usually faster and gives a better bound. However, it generally yields a solution with more zeros (i.e., more degeneracy) and there is no trivial initial solution. Note that since the method is based on the simplex, the quality of the initial solution has an important influence on the global method performance. We refer the reader to [9,19,39] for more details on finding an initial guess when using the simplex algorithm.
From a solution x^0 of $Ax = f$, one can compute an initial basis, a basic feasible solution and simplex multipliers (π, μ), which allows us to use the Dantzig-Wolfe algorithm (see section 2). If x^0 is nondegenerate, each of its m components corresponds to an element of L with exactly one nonzero component. These elements form the initial basis and a basic feasible solution is trivially obtained from it.

At each iteration, the Dantzig-Wolfe decomposition on the Compressive Sensing problem yields the exploration of 3^n elements, where n is the size of the signal. Even if this exploration is done implicitly in linear time thanks to equations (6) and (9) at each iteration, the whole resolution by Dantzig-Wolfe decomposition can be very slow.

Recall that in this section, we need to have a bounded solution space for the subproblems. Such a property has been derived with a consideration on the initial solution. In the next section, we use the specific way we define the bounded solution space for the subproblems to improve the performance of the method.

4 Improving performance through reduced decomposition

This section presents a modification of the method previously applied to the Compressive Sensing problem (see section 3) which yields a significant decrease in the maximum number of bases to explore and therefore enables the use of standard pivoting rules.

The main idea is to decompose the problem (7) in vectors with exactly one nonzero element (denoted q_i) instead of general extreme points (denoted p_i), plus the null vector. This reduced Dantzig-Wolfe decomposition enables to cope with only $2n+1$ vectors of this form (because of the sign and the null vector) and significantly decreases the symmetries of the problem.
Indeed, solutions of problem (8) are expressed as a convex combination of elements of $L = \{-K, 0, K\}^n$. Among all the vectors of L, we only consider those with at most one nonzero component. In other words, these vectors have the form $(0, \dots, 0, \pm K, 0, \dots, 0)^t$ or $(0, \dots, 0)^t$ and form the set M with $|M| = 2n+1$.

Moreover, we reduce the number of considered extreme points from exponential to linear. This yields that usual simplex rules can be used. Indeed, standard simplex rules on the reduced decomposition problem can output $2n$ different columns whereas Dantzig-Wolfe rule on the standard decomposition can output 3^n different columns, i.e., a rough estimate of $\binom{m}{2n}$ bases may be inspected instead of $\binom{m+1}{3^n}$.

4.1 Equivalence with original problem

Recall that the original Compressive Sensing problem has constraints of the form $Ax = f$. Assuming no degeneracy, any convex combination of m nonzero elements q_i corresponds to a solution of $Ax = f$ with m nonzero components, exactly as for a standard Dantzig-Wolfe decomposition (see section 2).

Moreover, each solution of $Ax = f$ can be represented as a solution v of $\sum_{i=1}^{2n+1}(Aq_i)v(i) = f$ with $\sum_{i=1}^{2n+1} v(i) = 1$ and $\forall i = 1, \dots, 2n+1, v(i) \geq 0$, i.e., a solution of the reduced Dantzig-Wolfe decomposition.
Recall that the basis of the program (8) has size $(m+1)\times(m+1)$. The initial basic feasible solution x^0 of the original problem, which is a solution of $Ax = f$, enables to compute K as $||x^0||_1$, which bounds the problem in the sense of the decomposition.
Let us denote by δ_i the vector that lives in $\mathbb{R}^n$ with all components at zero except the i^{th} component which equals 1. Then, the elements of M are written as

$$\begin{cases} q_{2i-1} = K\delta_i, & \forall i = 1, \dots, n, \\ q_{2i} = -K\delta_i, & \forall i = 1, \dots, n, \\ q_{2n+1} = (0, \dots, 0)^t. \end{cases} \tag{10}$$

We compute an initial basic solution v^0 for (8) from the initial basic feasible solution x^0 with the following formula:

$$\begin{cases} v^0(2i-1) = \frac{\max(0, x^0(i))}{||x^0||_1}, & \forall i = 1, \dots, n, \\ v^0(2i) = \frac{\max(0, -x^0(i))}{||x^0||_1}, & \forall i = 1, \dots, n, \\ v^0(2n+1) = 0. \end{cases} \tag{11}$$

It is worth noting that, at any iteration t and any $i \in \{1, \dots, n\}$, we cannot have $v^t(2i-1) \neq 0$ and $v^t(2i) \neq 0$ at the same time. With the previously defined q_i and $v^0(i)$, $x^0 = \sum_{i=1}^{2n+1}(Aq_i)v^0(i)$ and v^0 has exactly m nonzero components.
Since the bound K is computed from the initial feasible solution x^0 used by the optimization process, the reduced decomposition is only valid for the subproblem of the initial one where solutions are strictly better than the initial feasible solution. Assuming no degeneracy, the usual

pivoting rules of the simplex ensure this property at each iteration and then guarantees optimality of the solution of the reduced decomposition in the original problem sense.
The null vector is *mandatory* in this reduced decomposition and must be in base at each iteration. Let us denote by $x^t, t > 1$ the solution computed at the t^{th} iteration. Since pivoting rules guarantee strictly better solution as it is running, we get:

$$\forall t, \sum_i |x^{t+1}(i)| < \sum_i |x^t(i)| \ . \tag{12}$$

In terms of solutions of the reduced decomposition, since $x^t = \sum_{i=1}^{2n+1}(Aq_i)v^t(i)$, it is equivalent to:

$$\forall t, \sum_{i=1}^{m} v^{t+1}(i)|Aq_i^{t+1}| < \sum_{i=1}^{m} v^t(i)|Aq_i^t| \tag{13}$$

and

$$\forall t, \sum_{i=1}^{m+1} v^t(i) = 1 \ . \tag{14}$$

The sequence of $(\sum_{i=1}^{m} v^t(i))$ is strictly decreasing because for any given i the quantity $|Aq_i|$ is independent of the iterations t. Therefore, $v^t(m+1)$ acts as an accumulation term and enables $\sum_{i=1}^{m+1} v^t(i)$ to not exceed 1 when the solution improves in the l^1-norm sense. The convex combination is ensured during the whole optimization process thanks to the null vector being always present in base.
This shows that all basic feasible solution (except the initial one) has $v(2n+1) \neq 0$. Thus, the null vector needs to be in base at each iteration. In other words, we need to force this vector to be in the basis during all iterations.
As we shall see in the next subsection, the reduced cost of the null vector is 0. Thus, as the pivoting rules only consider variables for which reduced cost is strictly non positive, the null vector is never selected. If this vector is not forced to remain in base during the whole optimization process, we get an optimal solution of our reduced decomposition that is still feasible for the original Compressive Sensing problem but not necessarily optimal.

We have seen that the $2n+1$ elements of M are enough to express a solution of the original Compressive Sensing problem if the null vector is always forced to be in base. The next subsection shows the decomposition with vectors of M from a reduced cost point view.

4.2 From a reduced cost point of view

Let us note that all elements of L can be written as a sum of elements M. Let $p_{i,j}$ be a vector of L with exactly two nonzero components that is the sum of $q_i \in L$ and $q_j \in L$. We have $s_{q_i} = -1$ or $s_{q_i} = 1$ according to the sign of the nonzero component of the vector q_i. The reduced cost $\bar{v}(i)$ of the variable $v(i)$ that corresponds to the vector q_i for the decomposed problem is then

$$\bar{v}(i) = cq_i - \pi A q_i - \mu \ . \tag{15}$$

It can be rewritten as

$$\bar{v}(i) = (c(i) - \pi s_{q_i} A_i)K - \mu \ , \tag{16}$$

and the reduced cost $\bar{u}(i,j)$ of $p_{i,j}$ for the decomposed formulation is then

$$\bar{u}_{i,j} = (c(i) + c(j)) - \pi(s_{q_i} A_i + s_{q_j} A_j)K - \mu \ , \tag{17}$$

$$\bar{u}(i,j) = \bar{v}(i) + \bar{v}(j) + \mu \ . \tag{18}$$

Following subsection 4.1, we have $\mu = 0$ since the null vector is always in base. Generalization to vectors with more than two nonzero components is straightforward.

Recall that basic variables have reduced cost equals to zero. From property (18), one can see that each element of L (i.e., each extreme point of the original Dantzig-Wolfe decomposition), for which reduced cost is negative, implies that it exists at least one element of M (i.e., a one nonzero extreme point) with negative reduced cost. More precisely, three cases can occur:

(i) ($\bar{v}(i) \geq 0$ and $\bar{v}(j) \geq 0$) $\Leftrightarrow \bar{u}(i,j) \geq 0$, i.e., if no element of M can be chosen, no element of L can be chosen at all.
(ii) ($\bar{v}(i) \leq 0$ and $\bar{v}(j) \leq 0$) $\Leftrightarrow \bar{u}(i,j) \leq 0$, i.e., if an element of L can be chosen, the element $v(\arg\min(\bar{v}(i), \bar{v}(j)))$ of M can be chosen too.
(iii) ($\bar{v}(i) \leq 0$ and $\bar{v}(j) \geq 0$), the element $v(\arg(\bar{v}(i)))$ of M can be chosen.

These three cases tell us that if an extreme point with more than one nonzero has a strictly negative reduced cost, then at least one extreme point with exactly one nonzero has such a reduced cost too. Thanks to the chosen order, the extreme point with exactly one nonzero will be chosen. If no such extreme point if found, then no extreme point at all has a strictly negative reduced cost, and thus we have an optimal solution. It yields that if there is a solution for the standard Dantzig-Wolfe decomposition, there is a solution for our reduced decomposition.

4.3 Implications on standard simplex rules for our approach

In the standard Dantzig-Wolfe decomposition, at each iteration the pivoting rule is used to find a new element to enter the basis from a set of 3^n elements. Because this set is reduced to $2n + 1$ elements in the case of our reduced decomposition, usual pivoting rules developed for the simplex can be used. Table 1 lists some common pivoting rules, and more particularly the step consisting in choosing a new variable (column) to enter the basis. To choose a variable to leave the basis, the usual minimum ratio test is used [19]. If several solutions are output, the one with the lowest index is chosen.

rule name	entering variable
Dantzig	$\arg\min_i\{\bar{v}(i) \mid \bar{v}(i) < 0\}$
Steepest edge	$\arg\min_i\{\frac{\bar{v}(i)}{\sqrt{1+\sum_j (B^{-1}\times A)_{ij}}} \mid \bar{v}(i) < 0\}$
Bland	$\min\{i \mid \bar{v}(i) < 0\}$

Table 1: Pivoting rules

We now present these three rules in a general context and in our case of reduced decomposition:

– *Dantzig rule:* the Dantzig rule [18] corresponds to the rule used by the original simplex algorithm, that consists in choosing the most negative reduced cost. It corresponds to a visit of $O(n)$ variables, i.e., $O(nm)$ operations to compute all reduced costs.
For our reduced decomposition, it can be noted that case (ii) of last subsection leads to choose an extreme point of the reduced decomposition which can be less interesting from a reduced cost point of view than the extreme point of the original Dantzig-Wolfe decomposition. It comes from the fact that a simplex iteration makes a variable to enter and another to leave. Because a variable of the standard Dantzig-Wolfe decomposition is represented by several variables in the reduced decomposition, several iterations are required to accomplish an equivalent improvement on the solution. However, since the simplex pivoting rules work locally iteration by iteration, a given base of the reduced decomposition at a given iteration has more freedom for improvement than a standard one. Indeed, thanks to the significant decrease of

both symmetries in the representations of solutions and the number of bases to visit, fewer iterations are required to find a solution, i.e., on average each iteration much better improves the solution than for a standard Dantzig-Wolfe decomposition.

– *Steepest edge rule:* the Steepest edge rule [32,28] is the state-of-the-art pivoting rule. This rule is much more time consuming per iteration than the Dantzig rule and requires more memory. Indeed, it needs all coefficients of the matrix $B^{-1} \times A$, where B is current basic matrix. This rule leads to a far better new variable to work with and therefore enables to drastically decrease the required number of iterations to reach optimal solution. It has the same complexity as the Dantzig rule: $O(n)$ variables are visited with $O(nm)$ operations. Note that the huge amount of running time required to find the pivot has led to develop faster approximate versions of this rule, such as Devex [35].
  In terms of our reduced decomposition, the same comments as for the Dantzig rule can be made for the exact same reasons.
– *Bland rule:* the Bland minimum index rule [7] is a rule that prevents from cycling. It is one of the anti-cycling rules that can be used to ensure a finite number of iterations even if degeneracy occurs. This rule consists in choosing the first variable whose reduced cost is strictly negative. This implies that an order on the variable must be chosen. It has the same worst case time complexity as the Dantzig rule. In practice, the Bland minimum index rule requires much less time to compute (only few reduced costs are effectively computed) but commonly leads to a poorer new variable to work with.
  In the case of decomposition, one can see that the use of the Bland minimum index rule on standard Dantzig-Wolfe decomposition is equivalent to using the same rule on our reduced decomposition, provided an adequate order is chosen. Let us choose an arbitrary order that satisfies the following property on the extreme points of the original Dantzig-Wolfe decomposition: elements are increasingly sorted by the number of their nonzero components. We see from (i), (ii) and (iii) that if a variable must be chosen among the 3^n variables then an element from the $2n + 1$ first variables is chosen, i.e., an element of M. Given this order and Bland minimum index rule, this shows that these $2n + 1$ vectors are enough to find a solution.

4.4 Computational costs

The most time consuming operations of the standard Dantzig-Wolfe method are the selection of the variable to enter using the pivoting rule and the computation of the new vector γ associated with the new variable (performed by a matrix/vector multiplication). These two operations have a worst case running time complexity of $O(mn)$. In our reduced decomposition, the computation of γ has a running time complexity of $O(m)$ because the size n vector of the matrix/vector multiplication is reduced to a single nonzero value. The steepest edge, Dantzig, Bland and Dantzig-Wolfe rules give the same worst case complexity of $O(n)$ variables to visit, each of them requiring $O(m)$ operations at each iteration to compute their reduced cost. This yields an overall number of $O(mn)$ operations per iteration. In the case of Dantzig and steepest edge rules, the pivoting operation can be directly expressed as a matrix/vector multiplication. Besides, the steepest edge rule requires to maintain the simplex tableau, which equals $B^{-1} \times A$, where B is the current basic matrix. Instead of recomputing this matrix from scratch at each iteration, it can be updated in the same way as for B^{-1} (see section 2). This enables to compute the needed matrix in $O(mn)$. Since A is dense, computing the steepest matrix is very costly, and rarely yields better wall clock times compared to the use of the Dantzig rule.

Another interesting point is that for rules that need to compute all reduced costs, $2n$ variables are inspected at each iteration but at a cost of only n reduced cost computations. Indeed, considering formula (16), we see that the most time consuming operation is the dot product πA_i. It is independent of the sign and then can be computed only once for both signs.

In this paper, A is always described explicitly. In other words, all the coefficients of the matrix are stored in memory, and computations involving A are done through standard matrix operations. In the context of Compressive Sensing, the matrix A can be a sub-matrix of a transform such

as the Discrete Fourier Transform (DFT) or Discrete Cosine Transform (DCT). Therefore these fast transforms can be used instead of matrix operations. Since the time complexity for computing a Fast Fourier Transform (FFT) is $O(n \log n)$, this allows to decrease the running time complexity from $O(mn)$ to $O(max(m^2, n\ log\ n))$.

The standard Dantzig-Wolfe method allows to compute the n subproblems in parallel without communication thanks to independency (see section 2). All operations in $O(mn)$ involved in the reduced decomposition can be done in parallel (matrix/vector multiplications and premultiplication by matrix η for the steepest edge rule). When FFTs are used, they can also be computed in parallel .

5 Implementation and experiments

Every general purpose linear programming solver works with sparse representations and features fast algorithms adapted to them. This is due to the nature of problems usually solved by linear programming. Therefore, many research has been done with this hypothesis in mind. For an overview of the different theoretical and practical advances in linear programming, we refer the reader to [6].

Since most real world applications in linear programming involve sparse matrices, appropriate implementations of linear algebra must be written to achieve performance. On the contrary, high density matrices are used in the case of Compressive Sensing. Linearization of the Compressive Sensing problem (formulation (2)) makes a sparse matrix appear, for which standard implementations of linear programming algorithms are tailored. However, the decomposition method we used involves only dense matrices of maximum size $m \times n$ and computations of at most $O(mn)$ operations on them. All these operations can be easily and efficiently written due to the assured density of their operands. Therefore, the decomposition method we used yields very fast iterations thanks to basic properties of the matrices used in Compressive Sensing.

The computer used for our experiment is a Core 2 Duo 2GHz with 3MB of level 2 cache, 2GB of DDR3 RAM and runs Mac OS X 10.5.6. As reference point we use glpk 4.35, a standard sparse linear programming solver, compiled with gcc 4.3.2 and the fastest optimization flags found for it: `-O3 -fomit-frame-pointer`. Time and memory used are those reported by glpk. We compare our method to the fastest resolution scheme of glpk on formulation (7): the dual simplex without scaling nor presolve. Note that [29] has reported this simplex variant also gives the best behavior for another general purpose linear programming solver on similar problems.
Our implementation has been written in C++ and compiled with the same compiler and flags as for glpk. We use double precision floating points and no vectorization nor parallelization to perform a fair comparison with glpk. FFT computations are performed thanks to the FFTW 3.2.2 (Fastest Fourier Transform in the West) compiled with default flags. We refer the reader to [30] for further details on FFTW.

We consider two kinds of explicit matrices for Compressive Sensing: orthogonalized Gaussian matrices and partial Discrete Cosine matrix (DCT) matrices. Orthogonalized Gaussian matrices have their elements generated using i.i.d normal distributions $\mathcal{N}(0, 1)$ and their rows are orthogonalized. Partial DCT matrices are generated by randomly choosing, with uniform sampling, m rows from the full DCT. In the case of DCTs, we also implement a version of the method that takes benefit from fast transforms instead of explicit matrices. In this case, partial DCTs are computed as complex-to-complex FFTs with additional $O(n)$ pre and postprocessing steps for converting the results to real numbers. Indeed, direct DCT implementations are currently subject to performance issues and cannot be used for our purpose. Using this complex-to-complex FFT means an overhead of 4 times the memory required by directly applying DCT but allows to keep good time performances. Signals f have a nonzero ratio of $\frac{1}{10}$, i.e, $\frac{1}{10}$ of the m components of f are different from zero, and nonzeros are -1 and $+1$.

The following two figures only show results for orthogonalized Gaussian matrices since they yield almost same behaviors as for partial DCT matrices and since the latter is commonly solved through fast transforms. Complete results of the experiments (running time, number of iterations

and memory used) for both kinds of constraint matrices and representations of partial DCT matrices (explicit matrices or fast transforms) are given in appendix A. We use four different ratios $\frac{m}{n}$: $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$ and $\frac{1}{4}$. The reported results are the average of the results of 10 different instances.

Figure 1 shows how the considered decomposition methods behave. A standard Dantzig-Wolfe decomposition is compared to the reduced decomposition with the pivoting rules described in section 4. We can see that both in terms of running time and number of iterations, the *standard* Dantzig-Wolfe decomposition is the worst competitor. For the reduced decomposition, the steepest edge rule requires fewer iterations than the Dantzig rule, which itself requires fewer iterations than the Bland rule. In terms of running time, the Dantzig rule yields better results than the Bland rule. However, even if for the biggest instances considered, the steepest edge rule decreases the number of iterations by a factor of more than 2 with respect to the Dantzig rule. The steepest edge running time is slightly slower than for the Dantzig rule. Indeed, since we are solving dense instances, the steepest edge pivoting rule is very time consuming and the decrease of the number of iterations is not enough to compete with the Dantzig rule in terms of running time. On the contrary, for small instances of the problem, the steepest edge rule yields slightly better results than the Dantzig rule.

Figure 2 shows the comparison between the reduced decomposition with the dual simplex with the the steepest edge and the Dantzig rules. These last two rules gives the results for the reduced decomposition. In the case of the dual simplex, the steepest edge is up to 5.4 times faster than the Dantzig rule with 3.8 times fewer iterations. In terms of number of iterations, the reduced decomposition with a given rule yields fewer iterations than for the dual simplex. The same behavior occurs for the running time.

The Dantzig rule is the fastest rule (in terms of running time) for the reduced decomposition and it gives better running time and smaller number of iterations than the dual simplex, whatever the rule used for the latter. Any pivoting rule used for the dual simplex and for the reduced decomposition among the two considered rules (i.e., Dantzig and steepest edge), the reduced decomposition is faster.

It is worth to mention than the dual simplex is always better than the *standard* Dantzig-Wolfe decomposition. Moreover, for the smallest considered ratio $\frac{m}{n}$, i.e, $\frac{1}{32}$, and our experiment sizes, the Bland rule shows better running times than the dual simplex with the steepest edge rule (which is its better pivoting rule). However, the number of iterations is far worst. For other considered ratios $\frac{m}{n}$, the dual simplex is always faster than the reduced decomposition using the Bland rule in both running time and number of iterations.

Figure 3 compares the running time of our reduced decomposition using its fastest rule in terms of running time, i.e., the Dantzig rule, for both representations of the partial DCT matrices: explicit matrices and fast transform-based computations. For small instances, the matrix representation is the fastest. For both ratios $\frac{m}{n}$, the situation changes around $m = 64$ and the use of FFTs gives significant performance increases. For small instances the overhead of using FFTs, which has been previously discussed in this section, is no more hidden by the faster computations of the parts of the algorithm previously in $O(mn)$. Moreover, if the ratio $\frac{m}{n}$ is not small enough, the cost of the minimum ratio test to find the variable to leave the basis ($O(m^2)$) can be significant compared to the FFT computations ($O(n \log n)$). In this case, the overall improvement can be not as significant as for smaller ratios $\frac{m}{n}$.

Recall that the Bland rule selects the first variable with non positive reduced cost to enter the basis. One may expect to perform few reduced cost computations in order to find it. However the use of fast transforms enables to compute *all* reduced costs in $O(n \log n)$ but does not have the versatility to compute only few of them in a more efficient way. Thus implementing the Bland

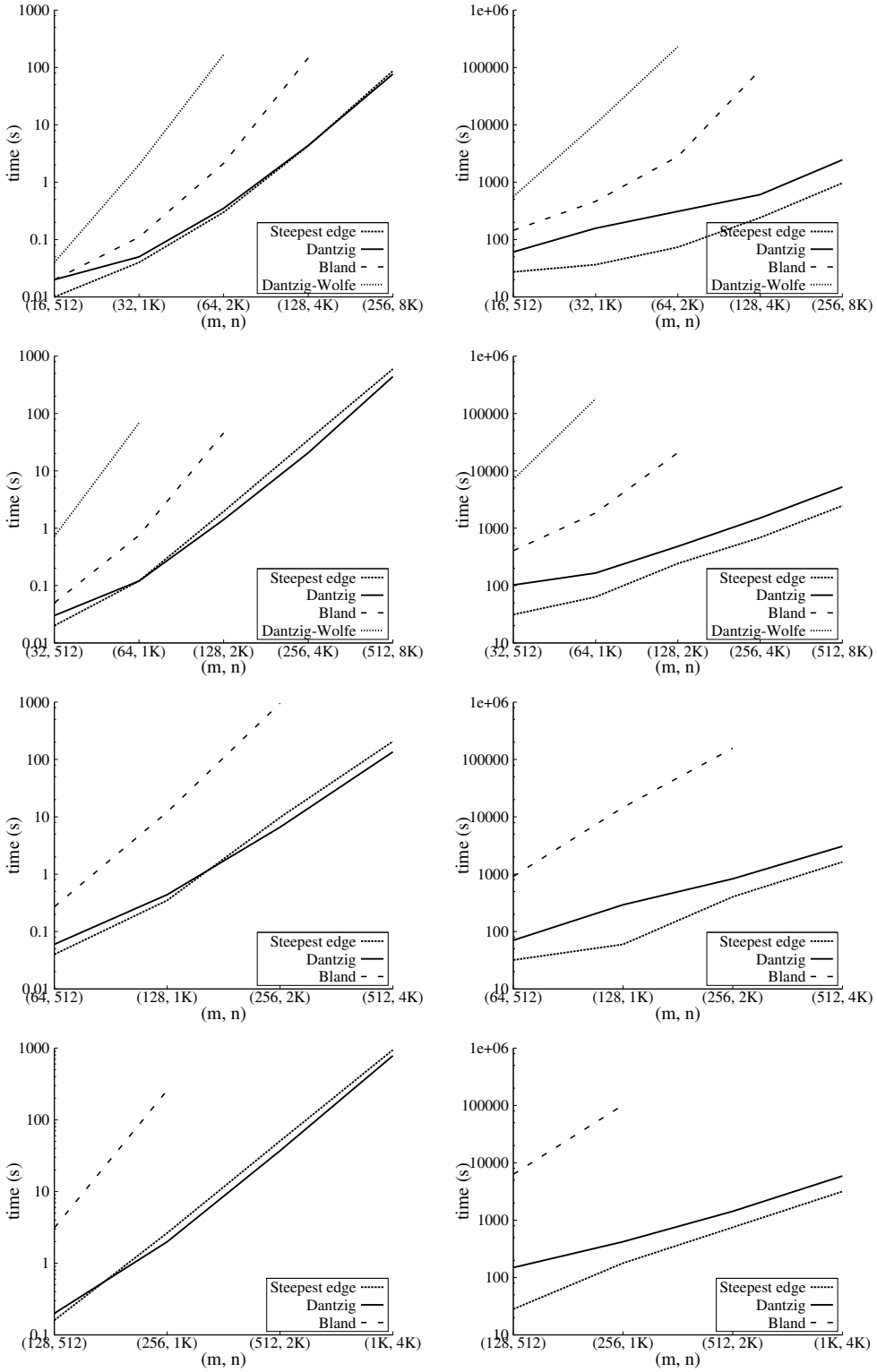

Fig. 1: Comparison of decompositions with various pivoting rules. Average of 10 instances with orthogonalized Gaussian matrices for various ratios $\frac{m}{n}$ (left: time (s); right: number of iterations; first line: $\frac{m}{n} = \frac{1}{32}$; second line: $\frac{m}{n} = \frac{1}{16}$; third line: $\frac{m}{n} = \frac{1}{8}$; fourth line: $\frac{m}{n} = \frac{1}{4}$).

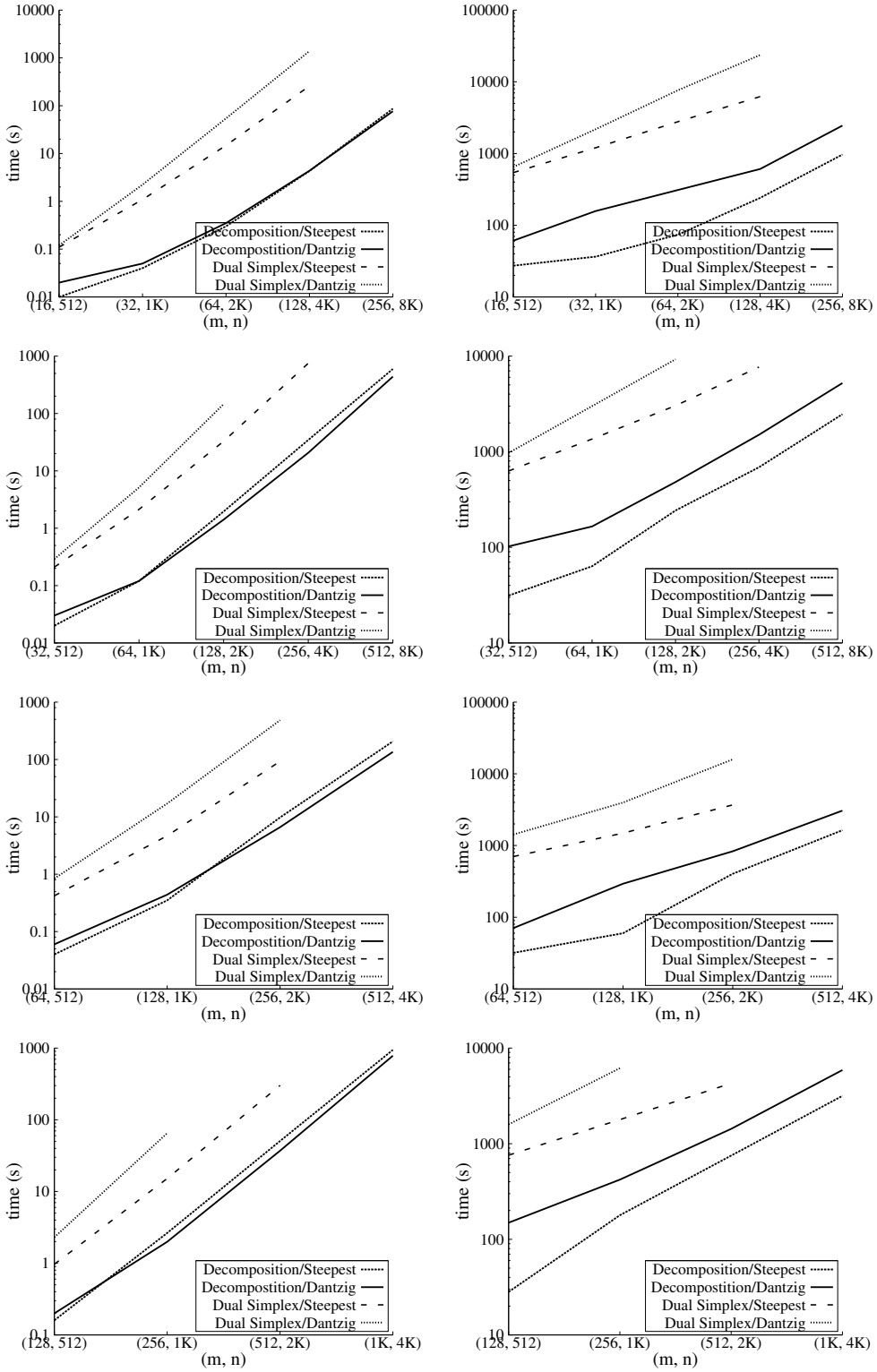

Fig. 2: Comparison of decomposition and dual simplex with best pivoting rules. Average of 10 instances with orthogonalized Gaussian matrices for various ratios $\frac{m}{n}$ (left: time (s); right: number of iterations; first line: $\frac{m}{n} = \frac{1}{32}$; second line: $\frac{m}{n} = \frac{1}{16}$; third line: $\frac{m}{n} = \frac{1}{8}$; fourth line: $\frac{m}{n} = \frac{1}{4}$).

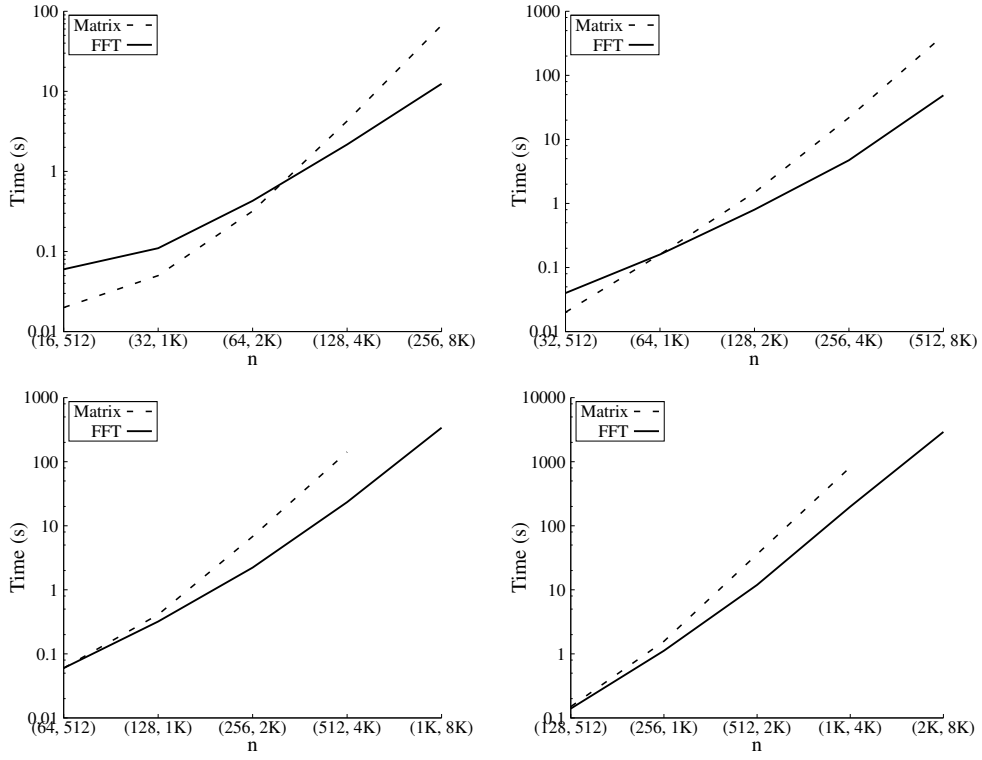

Fig. 3: Partial DCT matrix versus FFT running times using the Dantzig rule with various ratios $\frac{m}{n}$ (top left: $\frac{m}{n} = \frac{1}{32}$; top right: $\frac{m}{n} = \frac{1}{16}$; bottom left: $\frac{m}{n} = \frac{1}{8}$; bottom right: $\frac{m}{n} = \frac{1}{4}$).

rule when fast transforms are available has a cost similar to the Dantzig rule (which requires to compute all reduced cost). Thus, such a rule is not competitive when fast transforms are used.

To sum up, the reduced decomposition using the Dantzig rule is the fastest method considered in this paper. We have not observed a cycling issue for the considered instances of the CS problem. Note that one could use the Bland rule to ensure a finite number of iterations.

6 Conclusion

We have proposed a modified Dantzig-Wolfe decomposition for the Compressive Sensing problem. We have shown that our approach allows to efficiently implement any simplex pivoting rules. Besides, we have exhibited that our decomposition with the Dantzig rule yields the best results and outperforms any other standard approaches relying on the simplex algorithm. Note that our decomposition can be easily parallelized (as for the original Dantzig-Wolf decomposition) and also easily vectorized since the matrices are dense. This work is left as future work.

A Appendix

Complementary results for all approaches with all considered pivoting rules are given in this appendix. Table 2 and table 3 give the number of iterations and the running time for the partial DCT matrices and the orthogonalized Gaussian matrices, respectively. Table 4 and table 5 give the memory consumption for the partial DCT matrices and the orthogonalized Gaussian matrices, respectively. Eventually, table 6 compares the running time results for partial DCT matrices implemented through explicit matrices versus fast transforms.

		Decomposition								Dual Simplex			
		Steepest Edge		Dantzig		Bland		Dantzig-Wolfe		Steepest Edge		Dantzig	
m	n	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
16	512	26.0	0.01	26.2	0.02	33.7	0.02	25.5	0.02	531.1	0.10	608.8	0.10
32	1024	36.2	0.04	167.7	0.05	1320.8	0.13	13389.6	2.56	1225.0	1.08	2320.7	2.13
64	2048	80.5	0.31	274.3	0.32	10067.3	3.15	275771.2	199.31	2799.5	14.72	7737.0	49.61
128	4096	233.1	4.43	659.7	4.26	99905.2	162.76			6028.5	247.20	24730.0	1499.80
256	8192	963.8	87.11	2131.4	66.87								
32	512	20.0	0.02	79.4	0.02	692.1	0.05	7664.7	0.8	635.7	0.21	974.1	0.36
64	1024	67.2	0.12	239.0	0.16	7440.4	1.35	215982.4	80.9	1438.7	2.30	2774.2	6.65
128	2048	205.6	1.84	482.7	1.48	74701.6	54.58			3119.6	34.91	9619.6	157.20
256	4096	646.5	30.80	1430.2	21.87					7622.4	722.70		
512	8192	2588.3	655.83	5022.2	420.04								
64	512	26.8	0.04	115.9	0.06	3170.8	0.38	95687.7	19.6	687.5	0.44	1186.7	0.97
128	1024	84.8	0.40	248.0	0.41	34079.8	14.26			1452.1	4.87	4076.5	15.95
256	2048	388.5	9.49	822.7	6.80	388105.3	828.93			3463.7	92.23	14188.5	508.47
512	4096	1602.2	213.23	3365.1	141.67								
128	512	45.2	0.16	81.0	0.15	11243.7	3.14			732.0	0.94	1794.2	2.21
256	1024	104.2	2.23	249.2	1.57	149173.6	165.58			1757.7	15.07	5195.8	58.90
512	2048	903.3	58.14	1647.8	35.93					4253.3	294.53		
1024	4096	3357.9	1000.73	6225.8	820.72								

Table 2: Single threaded running time and iteration number for partial DCT matrices. Average from 10 instances. Time in seconds. Empty cells where too long to compute (i.e., execution takes more than 1500 seconds) and are not reported.

		Decomposition								Dual Simplex			
		Steepest Edge		Dantzig		Bland		Dantzig-Wolfe		Steepest Edge		Dantzig	
m	n	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time	#iter	time
16	512	27.3	0.01	61.0	0.02	144.9	0.02	561.8	0.04	542.3	0.11	649.3	0.12
32	1024	36.5	0.04	157.6	0.05	464.2	0.11	10394.9	2.01	1203.7	1.09	2181.5	2.23
64	2048	73.8	0.30	310.0	0.35	2807.3	2.13	230864.7	167.41	2778.7	14.78	7615.6	54.31
128	4096	240.8	4.31	608.2	4.37	90572.4	146.83			6226.2	258.10	23673.4	1389.90
256	8192	967.4	86.83	2452.7	76.57								
32	512	31.3	0.02	102.3	0.03	405.3	0.05	7030.7	0.72	628.7	0.21	972.2	0.29
64	1024	63.4	0.12	165.3	0.12	1845.1	0.75	180575.0	68.13	1356.5	2.13	2998.6	5.13
128	2048	242.8	1.96	481.9	1.41	20767.1	45.76			3029.5	32.76	9255.5	145.60
256	4096	687.5	34.47	1503.5	20.63					7677.8	749.00		
512	8192	2466.2	596.47	5228.1	438.93								
64	512	31.9	0.04	70.6	0.06	913.8	0.27	62929.1	13.12	701.8	0.42	1421.4	0.84
128	1024	59.8	0.35	293.3	0.44	14596.0	12.09			1488.8	4.68	3970.6	17.05
256	2048	403.2	9.76	831.5	6.60	156885.0	953.36			3684.1	92.76	15893.2	483.48
512	4096	1633.5	206.53	3067.9	135.60								
128	512	28.2	0.16	149.3	0.20	6314.6	3.11			759.1	0.97	1592.1	2.30
256	1024	178.7	2.64	420.9	1.99	101441.2	257.58			1787.3	15.18	6160.8	65.09
512	2048	750.9	50.57	1428.8	37.04					4285.0	301.48		
1024	4096	3163.0	944.66	5897.4	782.60								

Table 3: Single threaded running time and iteration number for orthogonalized Gaussian matrices. Average from 10 instances. Time in seconds. Empty cells where too long to compute (i.e., execution takes more than 1500 seconds) and are not reported.

References

1. G. Astfalk, I. Lustig, R. Marsten, and D. Shanno. The interior-point method for linear programming. *IEEE Software*, 9(4):61–68, 1992.
2. W. U. Bajwa, J. D. Haupt, G. M. Raz, S. J. Wright, and R. D. Nowak. Toeplitz-structured compressed sensing matrices. In *Proceedings of the 14th IEEE/SP Workshop on Statistical Signal Processing (SSP)*, pages 294–298, Aug. 2007.
3. C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.
4. C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

		Decomposition		Dual Simplex	
m	n	Steepest Edge	Dantzig/Bland/Dantzig-Wolfe	Steepest Edge	Dantzig
16	512	0.2	0.2	1.4	1.5
32	1024	0.8	0.6	4.2	4.6
64	2048	3.2	2.2	13.7	15.6
128	4096	12.7	8.6	49.0	52.0
256	8192	50.1	34.0		
32	512	0.4	0.3	2.2	2.3
64	1024	1.7	1.2	6.9	6.9
128	2048	6.5	4.5	24.6	26.2
256	4096	25.8	17.8	92.8	
512	8192	102.6	70.6		
64	512	0.9	0.6	3.5	3.5
128	1024	3.5	2.5	12.5	12.5
256	2048	13.7	9.7	47.0	47.0
512	4096	54.3	38.3		
128	512	1.9	1.4	6.4	6.4
256	1024	7.6	5.6	24.0	24.0
512	2048	30.2	22.2	93.0	
1024	4096	120.4	88.4		

Table 4: Memory used for partial DCT matrices. Average from 10 instances. Memory in megabytes.

		Decomposition		Dual Simplex	
m	n	Steepest Edge	Dantzig/Bland/Dantzig-Wolfe	Steepest Edge	Dantzig
16	512	0.2	0.2	1.5	1.6
32	1024	0.8	0.6	4.1	4.3
64	2048	3.2	2.2	13.7	14.8
128	4096	12.7	8.6	49.0	49.0
256	8192	50.1	34.0		
32	512	0.4	0.3	2.1	2.3
64	1024	1.7	1.2	6.9	7.7
128	2048	6.5	4.5	24.6	26.2
256	4096	25.8	17.8	92.8	
512	8192	102.6	70.6		
64	512	0.9	0.6	3.5	3.5
128	1024	3.5	2.5	12.5	12.5
256	2048	13.7	9.7	48.6	48.6
512	4096	54.3	38.3		
128	512	1.9	1.4	6.4	6.4
256	1024	7.6	5.6	24.0	24.0
512	2048	30.2	22.2	93.0	
1024	4096	120.4	88.4		

Table 5: Memory used for orthogonalized Gaussian matrices. Average from 10 instances. Memory in megabytes.

5. J. Bioucas-Dias and M. Figueiredo. A new TwIST: two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Trans. on Image Processing*, 16(12):2992–3004, 2007.
6. R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50:3–15, 2002.
7. R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.
8. P. Bloomfield and W. Steiger. *Least Absolute Deviations: Theory, Applications, and Algorithms*. Birkhauser, 1983.
9. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
10. K. Bredies and D. A. Lorenz. Iterated hard shrinkage for minimization problems with sparsity constraints. *SIAM Journal on Scientific Computing*, 30(2):657–683, 2008.
11. E. Candès and J. Romberg. Quantitative robust uncertainty principles and optimally sparse decompositions. *Foundations of Computational Mathematics*, 6:227–254, 2006.
12. E. Candès, J. Romberg, and T Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. on Information Theory*, 52(2):489–509, 2006.
13. E. Candès and T Tao. Near optimal signal recovery from random projections: Universal encoding strategies? *IEEE Trans. on Information Theory*, 52(12):5406–5426, 2006.
14. A. Chambolle, R. A. DeVore, N.-Y. Lee, and B. J. Lucier. Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage. *IEEE Trans. on Image Processing*, 7:319–335, 1998.

m	n	matrix	FFT
16	512	0.02	0.06
32	1024	0.05	0.11
64	2048	0.32	0.43
128	4096	4.26	2.17
256	8192	66.87	12.45
32	512	0.02	0.04
64	1024	0.16	0.16
128	2048	1.48	0.80
256	4096	21.87	4.73
512	8192	420.04	48.72
64	512	0.06	0.06
128	1024	0.41	0.32
256	2048	6.80	2.22
512	4096	141.67	23.38
1024	8192		339.08
128	512	0.15	0.14
256	1024	1.57	1.12
512	2048	35.93	11.93
1024	4096	820.72	198.44
2048	8192		2913.37

Table 6: Partial DCT matrix versus FFT running times using the Dantzig rule with various ratios $\frac{m}{n}$. Empty cells where too long to compute (i.e., execution takes more than 1500 seconds) and are not reported.

15. Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.
16. A. Cohen, W. Dahmen, and R. DeVore. Compressed sensing and best k-term approximation. *J. Amer. Math. Soc.*, 22:211–231, 2009.
17. P. Combettes and J.-C. Pesquet. Proximal thresholding algorithm for minimization over orthonormal bases. *SIAM Journal on Optimization*, 18(4):1351–1376, 2007.
18. G. B. Dantzig. Programming in a linear structure. USAF, Washington D.C., 1948.
19. G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, 1963.
20. G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programming. *Econometrica*, 9(4), 1961.
21. I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications in Pure and Applied Mathematics*, 57(11):1413–1457, 2004.
22. R. A. DeVore. Deterministic constructions of compressed sensing matrices. *Journal of Complexity*, 4–6(23):918–925, 2007.
23. D. L. Donoho. Compressed sensing. *IEEE Trans. on Information Theory*, 52(4):1289–1306, 2006.
24. B. P. Dzielinski and R. E. Gomory. Optimal programming of lot sizes, inventory and labor allocations. *Management Science*, 11(9):874–890, 1965.
25. M. Elad. Why simple shrinkage is still relevant for redundant representation? *IEEE Trans. on Information Theory*, 52:5559–5569, 2006.
26. M. Figueiredo and R. Nowak. An EM algorithm for wavelet-based image restoration. *IEEE Trans. on Image Processing*, 12(8):906–916, 2003.
27. M. Figueiredo, R. Nowak, and S. Wright. Gradient projection for sparse reconstruction: application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing*, 1(3):586–598, 2007.
28. J. J. H. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.
29. M. P. Friedlander and M. A. Saunders. Discussion: the Dantzig selector: statistical estimation when p is much larger than n. *Annals of Statistics*, 35(6):2385–2391, December 2007.
30. M. Frigo and S.G. Johnson. FFTW: an adaptive software architecture for the FFT. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 3, pages 1381–1384, May 1998.
31. S. I. Gass and S. Vinjamuri. Cycling in linear programming problems. *Computers and Operations Research*, 31(2):303–311, 2004.
32. D. Goldfarb and J. Reid. A practicable steepest edge simplex algorithm. *Mathematical Programming*, 12:361–371, 1977.
33. T. Goldstein and S. Osher. The split Bregman method for l1 regularized problems. Technical Report CAM 08-29, UCLA, 2008.
34. R. Griesse and D. A. Lorenz. A semismooth Newton method for Tikhonov functionals with sparsity constraints. *Inverse Problems*, 24(3), 2008. 2008.
35. P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5(1):1–28, 1973.
36. W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, Chicago, IL, USA, 1939.
37. S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale l_1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):606–617, 2007.

38. H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. Berkeley, California, 1951.
39. L. S. Lasdon. *Optimization Theory for Large Systems*. The Macmillan Company, New York, 1970.
40. S. Mallat. *A wavelet tour of signal processing*. Academic Press, 1999.
41. R. K. Martinson and J. Tind. An interior point method in Dantzig-Wolfe decomposition. *Computers and Operations Research*, 26(12):1195–1216, 1999.
42. R. Nowak and M. Figueiredo. Fast wavelet-based image deconvolution using the em algorithm. In *Proceedings of the 35th Asilomar Conference on Signals, Systems, and Computers*, pages 371–275, 2001.
43. T.-H. Tcheng. *Scheduling of a Large Forestry-Cutting Problem by Linear Programming Decomposition*. PhD thesis, University of Iowa, 1966.
44. R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, 58:267–288, 2006.
45. J. Tropp. Just relax: Convex programming methods for identifying sparse signals. *IEEE Trans. on Information Theory*, 51(3):1030–1051, 2006.
46. J. A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Trans. on Information Theory*, 50(10):2231–2242, 2004.
47. J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Info. Theory*, 50(10):2231–2242, 2004.
48. J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Signal Processing*, 86:572–588, 2006.
49. J. A. Tropp and S. J. Wright. Computational methods for sparse solution of linear inverse problems. Technical Report ACM Report 2009-01, Caltech, April 2009.
50. F. Vanderbeck. On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48(1):111–128, 2000.
51. W. Yin, S. Osher, D. Goldfarb, and J. Darbon. Bregman iterative algorithms for l1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168, 2008.
52. G. Zhao. Interior-point methods with decomposition for solving large-scale linear programs. *JOTA*, 102:169–192, 1999.