Mach Learn (2016) 104:195–221
DOI 10.1007/s10994-016-5578-4

Fast and scalable Lasso via stochastic Frank–Wolfe methods with a convergence guarantee

Emanuele Frandi[1] · Ricardo Ñanculef[2] · Stefano Lodi[3] · Claudio Sartori[3] · Johan A. K. Suykens[1]

Received: 23 October 2015 / Accepted: 28 June 2016 / Published online: 21 July 2016

Abstract Frank–Wolfe (FW) algorithms have been often proposed over the last few years as efficient solvers for a variety of optimization problems arising in the field of machine learning. The ability to work with cheap projection-free iterations and the incremental nature of the method make FW a very effective choice for many large-scale problems where computing a sparse model is desirable. In this paper, we present a high-performance implementation of the FW method tailored to solve large-scale Lasso regression problems, based on a randomized iteration, and prove that the convergence guarantees of the standard FW method are preserved in the stochastic setting. We show experimentally that our algorithm outperforms several existing state of the art methods, including the Coordinate Descent algorithm by Friedman et al. (one of the fastest known Lasso solvers), on several benchmark datasets with a very large number of features, without sacrificing the accuracy of the model. Our results illustrate that the algorithm is able to generate the complete regularization path on problems of size up to four million variables in <1 min.

Editors: Thomas Gärtner, Mirco Nanni, Andrea Passerini, and Celine Robardet.

✉ Emanuele Frandi
emanuele.frandi@esat.kuleuven.be

Ricardo Ñanculef
jnancu@inf.utfsm.cl

Stefano Lodi
stefano.lodi@unibo.it

Claudio Sartori
claudio.sartori@unibo.it

Johan A. K. Suykens
johan.suykens@esat.kuleuven.be

1 ESAT-STADIUS, KU Leuven, Leuven, Belgium

2 Department of Informatics, Federico Santa María Technical University, Valparaíso, Chile

3 Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

Keywords Frank–Wolfe algorithm · Lasso · Large-scale regression

1 Introduction

Many machine learning and data mining tasks can be formulated, at some stage, in the form of an optimization problem. As constantly growing amounts of high dimensional data are becoming available in the Big Data era, a fundamental thread in research is the development of high-performance implementations of algorithms tailored to solving these problems in a very large-scale setting. One of the most popular and powerful techniques for high-dimensional data analysis is the *Lasso* (Tibshirani 1996). In the last decade there has been intense interest in this method, and several papers describe generalizations and variants of the Lasso (Tibshirani 2011). In the context of supervised learning, it was recently proved that the Lasso problem can be reduced to an equivalent SVM formulation, which potentially allows one to leverage a wide range of efficient algorithms devised for the latter problem (Jaggi 2014). For unsupervised learning, the idea of Lasso regression has been used in Lee et al. (2010) for bi-clustering in biological research.

From an optimization point of view, the Lasso can be formulated as an ℓ_1-regularized least squares problem, and large-scale instances must usually be tackled by means of an efficient first-order algorithm. Several such methods have already been discussed in the literature. Variants of Nesterov's Accelerated Gradient Descent, for example, guarantee an optimal convergence rate among first-order methods (Nesterov 2013). Stochastic algorithms such as Stochastic Gradient Descent and Stochastic Mirror Descent have also been proposed for the Lasso problem (Langford et al. 2009; Shalev-Shwartz and Tewari 2011). More recently, Coordinate Descent (CD) algorithms (Friedman et al. 2007, 2010), along with their stochastic variants (Shalev-Shwartz and Tewari 2011; Richtárik and Takâĉ 2014), are gaining popularity due to their efficiency on structured large-scale problems. In particular, the CD implementation of Friedman et al. mentioned above is specifically tailored for Lasso problems, and is currently recognized as one of the best solvers for this class of problems.

The contribution of the present paper in this context can be summarized as follows:

– We propose a high-performance stochastic implementation of the classical Frank–Wolfe (FW) algorithm to solve the Lasso problem. We show experimentally how the proposed method is able to efficiently scale up to problems with a very large number of features, improving on the performance of other state of the art methods such as the Coordinate Descent algorithm in Friedman et al. (2010).
– We include an analysis of the complexity of our algorithm, and prove a novel convergence result that yields an $\mathcal{O}(1/k)$ convergence rate analogous (in terms of expected value) to the one holding for the standard FW method.
– We highlight how the properties of the FW method allow to obtain solutions that are significantly more sparse in terms of the number of features compared with those from various competing methods, while retaining the same optimization accuracy.

On a broader level, and in continuity with other works from the recent literature (Ñanculef et al. 2014; Harchaoui et al. 2014; Signoretto et al. 2014), the goal of this line of research is to show how FW algorithms provide a general and flexible optimization framework, encompassing several fundamental problems in machine learning.

Structure of the paper

In Sect. 2, we provide an overview of the Lasso problem and its formulations, and review some of the related literature. Then, in Sect. 3, we discuss FW optimization and specialize the algorithm for the Lasso problem. The randomized algorithm used in our implementation is discussed in Sect. 4, and its convergence properties are analyzed. In Sect. 5 we show several experiments on benchmark datasets and discuss the obtained results. Finally, Sect. 6 closes the paper with some concluding remarks.

2 The Lasso problem

Suppose we are given data points (x_ℓ, y_ℓ), $\ell = 1, 2, \ldots, m$, where $x_\ell = (x_{\ell 1}, \ldots, x_{\ell p})^T \in \mathbb{R}^p$ are some predictor variables and y_ℓ the respective responses. A common approach in statistical modeling and data mining is the linear regression model, which predicts y_ℓ as a linear combination of the input attributes:

$$\hat{y}_\ell = \sum_{i=1}^{p} \alpha_i x_{\ell i} + \alpha_0 \, .$$

In a high-dimensional, low sample size setting ($p \gg m$), estimating the coefficient vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_p)^T \in \mathbb{R}^p$ using ordinary least squares leads to an ill-posed problem, i.e. the solution becomes not unique and unstable. In this case, a widely used approach for model estimation is regularization. Among regularization methods, the Lasso is of particular interest due to its ability to perform variable selection and thus obtain more interpretable models (Tibshirani 1996).

2.1 Formulation

Let X be the $m \times p$ design matrix where the data is arranged row-wise, i.e. $X = [x_1, \ldots, x_m]^T$. Similarly, let $y = (y_1, \ldots, y_m)^T$ be the m-dimensional response vector. Without loss of generality, we can assume $\alpha_0 = 0$ (e.g. by centering the training data such that each attribute has zero mean) (Tibshirani 1996). The Lasso estimates the coefficients as the solution to the following problem

$$\min_{\alpha} f(\alpha) = \tfrac{1}{2} \, \|X\alpha - y\|_2^2 \;\; \text{s.t.} \;\; \|\alpha\|_1 \leq \delta \, , \tag{1}$$

where the ℓ_1-norm constraint $\|\alpha\|_1 = \sum_i |\alpha_i| \leq \delta$ has the role of promoting sparsity in the regression coefficients. It is well-known that the constrained problem (1) is equivalent to the unconstrained problem

$$\min_{\alpha} \tilde{f}(\alpha) = \tfrac{1}{2} \, \|X\alpha - y\|_2^2 + \lambda \|\alpha\|_1 \, , \tag{2}$$

in the sense that given a solution α^* of (2) corresponding to a certain value of the parameter $\bar{\lambda}$, it is possible to find a $\bar{\delta}$ such that α^* is also a solution of (1), and vice versa. Since the optimal tradeoff between the sparsity of the model and its predictive power is not known a-priori, practical applications of the Lasso require to find a solution and the profiles of estimated coefficients for a range of values of the regularization parameter δ (or λ in the penalized formulation). This is known in the literature as the Lasso *regularization path* (Friedman et al. 2010).

2.2 Relevance and applications

The Lasso is part of a powerful family of regularized linear regression methods for high-dimensional data analysis, which also includes ridge regression (RR) (Hoerl and Kennard

1970; Hastie et al. 2009), the ElasticNet (Zou and Hastie 2005), and several recent extensions thereof (Zou 2006; Zou and Zhang 2009; Tibshirani et al. 2005). From a statistical point of view, they can be viewed as methods for trading off the bias and variance of the coefficient estimates in order to find a model with better predictive performance. From a machine learning perspective, they allow to adaptively control the capacity of the model space in order to prevent overfitting. In contrast to RR, which is obtained by substituting the ℓ_1 norm in (2) by the squared ℓ_2 norm $\sum_i |\alpha_i|^2$, it is well-known that the Lasso does not only reduce the variance of coefficient estimates but is also able to perform variable selection by shrinking many of these coefficients to zero. Elastic-net regularization trades off ℓ_1 and ℓ_2 norms using a "mixed" penalty $\Omega(\alpha) = \gamma\|\alpha\|_1 + (1-\gamma)\|\alpha\|_2$ which requires tuning the additional parameter γ (Zou and Hastie 2005). ℓ_p norms with $p \in [0, 1)$ can enforce a more aggressive variable selection, but lead to computationally challenging non-convex optimization problems. For instance, $p = 0$, which corresponds to "direct" variable selection, leads to an NP-hard problem (Weston et al. 2003).

Thanks to its ability to perform variable selection and model estimation simultaneously, the Lasso (and ℓ_1-regularization in general) is widely used in applications involving a huge number of candidate features or predictors. Examples of these applications include biomolecular data analysis, text regression, functional magnetic resonance imaging (fMRI) and sensor networks. In all these cases, the number dimensions or attributes p can far exceed the number of data instances m.

2.3 Related work

Problem (1) is a quadratic programming problem with a convex constraint, which in principle may be solved using standard techniques such as interior-point methods, guaranteeing convergence in few iterations. However, the computational work required *per iteration* as well as the memory demanded by these approaches make them practical only for small and medium-sized problems. A faster specialized interior point method for the Lasso was proposed in Kim et al. (2007), which however compares unfavorably with the baseline used in this paper (Friedman et al. 2010).

One of the first efficient algorithms proposed in the literature for finding a solution of (2) is the Least Angle Regression (LARS) by Efron et al. (2004). As its main advantage, LARS allows to generate the entire Lasso regularization path with the same computational cost as standard least-squares via QR decomposition, i.e. $\mathcal{O}(mp^2)$, assuming $m < p$ (Hastie et al. 2009). At each step, it identifies the variable most correlated with the residuals of the model obtained so far and includes it in the set of "active" variables, moving the current iterate in a direction equiangular with the rest of the active predictors. It turns out that the algorithm we propose makes the same choice but updates the solution differently using cheaper computations. A similar homotopy algorithm to calculate the regularization path has been proposed in Turlach (2005), which differs slightly from LARS in the choice of the search direction.

More recently, it has been shown by Friedman et al. that a careful implementation of the Coordinate Descent method (CD) provides an extremely efficient solver (Friedman et al. 2007, 2010; Friedman 2012), which also applies to more general models such as the ElasticNet proposed by Zou and Hastie (2005). In contrast to LARS, this method cyclically chooses one variable at a time and performs a simple analytical update. The full regularization path is built by defining a sensible range of values for the regularization parameter and taking the solution for a given value as warm-start for the next. This algorithm has been implemented into the Glmnet package and can be considered the current standard for solving this class of problems. Recent works have also advocated the use of Stochastic Coordinate Descent (SCD) (Shalev-Shwartz and Tewari 2011), where the order of variable updates is chosen randomly instead of cyclically. This strategy can prevent the adverse effects caused by possibly unfavorable orderings of the coordinates, and allows to prove stronger theoretical guarantees compared to the plain CD (Richtárik and Takáĉ 2014).

Other methods for ℓ_1-regularized regression may be considered. For instance, Zhou et al. recently proposed a geometrical approach where the Lasso is reformulated as a nearest point problem and solved using an algorithm inspired by the classical Wolfe method (Zhou et al. 2015). However, the popularity and proved efficiency of Glmnet on high-dimensional problems make it the chosen baseline in this work.

3 Frank–Wolfe optimization

One of the earliest constrained optimization approaches, the Frank–Wolfe algorithm (Frank and Wolfe 1956) has recently seen a sudden resurgence in interest from the optimization community (Clarkson 2010; Jaggi 2013), and several authors have pointed out how FW methods can be used as a principled and efficient alternative to solve several large-scale problems arising in machine learning, statistics, bioinformatics and related fields (Ñanculef et al. 2014; Frandi et al. 2015; Harchaoui et al. 2014; Signoretto et al. 2014). As argued in Sect. 3.2, the FW algorithm enjoys several properties that make it very attractive for this type of problems. Overall, though, the number of works showing experimental results for FW on practical applications is limited compared to that of the theoretical studies appearing in the literature. In the context of problems with ℓ_1-regularization or sparsity constraints, the use of FW has been discussed in Shalev-Shwartz et al. (2010), but no experiments are provided. A closely related algorithm has been proposed in Zhou et al. (2015), but its implementation has a high computational cost in terms of time and memory requirements, and is not suitable for solving large problems on a standard desktop or laptop machine. As such, the current literature does not provide many examples of efficient FW-based software for large-scale Lasso or l_1-regularized optimization. We aim to fill this gap by showing how a properly implemented stochastic FW method can improve on the performance of the current state of the art solvers on Lasso problems with a very large number of features.

3.1 The standard Frank–Wolfe algorithm

The FW algorithm is a general method to solve problems of the form

$$\min_{\alpha \in \Sigma} f(\alpha), \tag{3}$$

where $f : \mathbb{R}^p \rightarrow \mathbb{R}$ is a convex differentiable function, and $\Sigma \subset \mathbb{R}^p$ is a compact convex set. Given an initial guess $\alpha^{(0)} \in \Sigma$, the standard FW method consists of the steps outlined in Algorithm 1. From an implementation point of view, a fundamental advantage of FW is that the most demanding part of the iteration, i.e. the solution of the linear subproblem (4), has a computationally convenient solution for several problems of practical interest, mainly due to the particular form of the feasible set. The key observation is that, when Σ is a polytope (e.g. ℓ_1-ball of radius δ for the Lasso problem (1)), the search in step 3 can be reduced to a search among the vertices of Σ. This allows to devise cheap analytical formulas to find $u^{(k)}$, ensuring that each iteration has an overall cost of $\mathcal{O}(p)$. The fact that FW methods work with *projection-free* iterations is also a huge advantage on many practical problems, since a projection step to maintain the feasibility of the iterates (as needed by classical approaches such as proximal methods for Matrix Recovery problems) generally has a super-linear complexity (Lacoste-Julien et al. 2013).

Algorithm 1 The standard Frank–Wolfe algorithm

```
1: Input: an initial guess $\alpha^{(0)}$.
2: for $k = 0, 1, \ldots$ do
3:    Define a search direction $d^{(k)}$ by optimizing a linear model:
          $$u^{(k)} \in \underset{u \in \Sigma}{\operatorname{argmin}}\, (u - \alpha^{(k)})^T \nabla f(\alpha^{(k)}), \qquad d^{(k)} = u^{(k)} - \alpha^{(k)}. \tag{4}$$
4:    Choose a stepsize $\lambda^{(k)}$, e.g. via line-search:
          $$\lambda^{(k)} \in \underset{\lambda \in [0,1]}{\operatorname{argmin}}\, f(\alpha^{(k)} + \lambda d^{(k)}). \tag{5}$$
5:    Update: $\alpha^{(k+1)} = \alpha^{(k)} + \lambda^{(k)} d^{(k)}$.
6: end for
```

Another distinctive feature of the algorithm is the fact that the solution at a given iteration K can be expressed as a convex combination of the vertices $u^{(k)}$, $k = 0, \ldots, K-1$. Due to the incremental nature of the FW iteration, at most one new extreme point of Σ is discovered at each iteration, implying that at most k of such points are active at iteration k. Furthermore, this sparsity bound holds for the entire run of the algorithm, effectively allowing to control the sparsity of the solution as it is being computed. This fact carries a particular relevance in the context of sparse approximation, and generally in all applications where it is crucial to find models with a small number of features. It also represents, as we will show in our experiments in Sect. 5, one of the major differences between incremental, forward approximation schemes and more general solvers for ℓ_1-regularized optimization, which in general cannot guarantee to find sparse solutions along the regularization path.

3.2 Theoretical properties

We summarize here some well-known theoretical results for the FW algorithm which are instrumental in understanding the behaviour of the method. We refer the reader to (Jaggi 2013) for the proof of the following proposition. To prove the result, it is sufficient to assume that f has bounded curvature, which, as explained in Jaggi (2013), is roughly equivalent to the Lipschitz continuity of ∇f.

Proposition 1 [Sublinear convergence (Jaggi 2013)] *Let α^* be an optimal solution of problem* (3). *Then, for any $k \geq 1$, the iterates of Algorithm 1 satisfy*

$$f(\alpha^{(k)}) - f(\alpha^*) \leq \frac{4C_f}{k+2},$$

where C_f is the curvature constant of the objective function.

An immediate consequence of Proposition 1 is an upper bound on the iteration complexity: given a tolerance $\varepsilon > 0$, the FW algorithm finds an ε-*approximate solution*, i.e. an iterate $\alpha^{(k)}$ such that $f(\alpha^{(k)}) - f(\alpha^*) \leq \varepsilon$, after $\mathcal{O}(1/\varepsilon)$ iterations. Besides giving an estimate on the total number of iterations which has been shown experimentally to be quite tight in practice (Frandi et al. 2014, 2015), this fact tells us that the tradeoff between sparsity and accuracy can be partly controlled by appropriately setting the tolerance parameter. Recently, Garber and Hazan showed that under certain conditions the FW algorithm can obtain a convergence rate of $\mathcal{O}(1/k^2)$, comparable to that of first-order algorithms such as Nesterov's method (Garber and Hazan 2015). However, their results require strong convexity of the objective function and of the feasible set, a set of hypotheses which is not satisfied for several important ML problems such as the Lasso or the Matrix Recovery problem with trace norm regularization.

Another possibility is to employ a *Fully-Corrective* variant of the algorithm, where at each step the solution is updated by solving the problem restricted to the currently active vertices. The algorithm described in Zhou et al. (2015), where the authors solve the Lasso problem via a nearest point solver based on Wolfe's method, operates with a very similar philosophy. A similar case can be made for the LARS algorithm of Efron et al. (2004), which however updates the solution in a different way. The Fully-Corrective FW also bears a resemblance to the Orthogonal Matching Pursuit algorithms used in the Signal Processing literature (Tropp 2004), a similarity which has already been discussed in Clarkson (2010) and Jaggi (2013). However, as mentioned in Clarkson (2010), the increase in computational cost is not paid off by a corresponding improvement in terms of complexity bounds. In fact, the work in Lan (2014) shows that the result in Proposition 1 cannot be improved for any first-order method based on solving linear subproblems without strengthening the assumptions. Greedy approximation techniques based on both the vanilla and the Fully-Corrective FW have also been proposed in the context of approximate risk minimization with an ℓ_0 constraint by Shalev-Shwartz et al., who proved several strong runtime bounds for the sparse approximations of arbitrary target solutions (Shalev-Shwartz et al. 2010).

Finally, it is worth mentioning that the result of Proposition 1 can indeed be improved by using variants of FW that employ additional search directions, and allow under suitable hypotheses to obtain a linear convergence rate (Ñanculef et al. 2014; Lacoste-Julien and Jaggi 2014). It should be mentioned, however, that such rates only hold in the vicinity of the solution and that, as shown in Frandi et al. (2015), a large number of iterations might be required to gain substantial advantages. For this reason, we choose not to pursue this strategy in the present paper.

4 Randomized Frank–Wolfe for Lasso problems

A specialized FW algorithm for problem (1) can be obtained straightforwardly by setting Σ equal to the ℓ_1-ball of radius δ, hereafter denoted as $\odot_\delta$. In this case, the vertices of the feasible set (i.e., the candidate points among which $u^{(k)}$ is selected in the FW iterations) are $\mathcal{V}(\odot_\delta) = \{\pm\delta\mathbf{e}_i \;:\; i = 1, 2, \dots, p\}$, where $\mathbf{e}_i$ is the i-th element of the canonical basis. It is easy to see that the linear subproblem (4) in Algorithm 1 has a closed-form solution, given by:

$$u^{(k)} = -\delta \operatorname{sign}\left(\nabla f(\alpha^{(k)})_{i_*^{(k)}}\right)\mathbf{e}_{i_*^{(k)}} \equiv \tilde{\delta}^{(k)}\mathbf{e}_{i_*^{(k)}}, \quad i_*^{(k)} = \underset{i=1,\dots,p}{\operatorname{argmax}} \left|\nabla f(\alpha^{(k)})_i\right|. \tag{6}$$

In order to efficiently execute the iteration, we can exploit the form of the objective function to obtain convenient expressions to update the function value and the gradient after each FW iteration. The gradient of $f(\cdot)$ in (1) is

$$\nabla f(\alpha) = -X^T(y - X\alpha) = -X^T y + X^T X\alpha.$$

There are two possible ways to compute $\nabla f(\alpha^{(k)})_i$ efficiently. One is to keep track of the vector of residuals $R^{(k)} = \left(y - X\alpha^{(k)}\right) \in \mathbb{R}^m$ and compute $\nabla f(\alpha^{(k)})_i$ as

$$\nabla f\left(\alpha^{(k)}\right)_i = -z_i^T R^{(k)} = -z_i^T y + z_i^T X\alpha^{(k)}, \tag{7}$$

where $z_i \in \mathbb{R}^m$ is the i-th column of the design matrix X, i.e., the vector formed by collecting the i-th attribute of all the training points. We refer to this approach as the "method of residuals". The other way is to expand the second term in (7)

$$\nabla f\left(\alpha^{(k)}\right)_i = -z_i^T y + \sum_{j \neq 0} \alpha_j^{(k)} z_i^T z_j,$$

and keep track of the inner products $z_i^T z_j$ between z_i and the predictors z_j corresponding to non-zero coefficients of the current iterate. We call this the "method of active covariates". The discussion in the next subsections reveals that the first approach is more convenient if, at each iteration, we only need to access a small subset of the coordinates of $\nabla f(\alpha^{(k)})$. It is clear from (6) that after computing $\nabla f(\alpha^{(k)})_i$ for $i = 1, \ldots, p$ the solution to the linear subproblem in Algorithm 1 follows easily. The other quantities required by the algorithm are the objective value (in order to monitor convergence) and the line search stepsize in (5), which can be obtained as

$$\begin{aligned} f\left(\alpha^{(k)}\right) &= \frac{1}{2} y^T y + \frac{1}{2} S^{(k)} - F^{(k)}, \\ \lambda^{(k)} &= \lambda_* := \frac{S^{(k)} - \tilde{\delta} \nabla f\left(\alpha^{(k)}\right)_{i*} - F^{(k)}}{S^{(k)} - 2\tilde{\delta} G_{i*} + \tilde{\delta}^2 z_{i*}^T z_{i*}}, \end{aligned} \tag{8}$$

where $i* = i_*^{(k)}$, $G_{i*} = \nabla f(\alpha^{(k)})_{i*} + z_{i*}^T y$, and the terms $S^{(k)}$, $F^{(k)}$ can be updated recursively as

$$\begin{aligned} S^{(k+1)} &= (1 - \lambda_*)^2 S^{(k)} + 2\tilde{\delta}\lambda_*(1 - \lambda_*) G_{i*} + \tilde{\delta}^2 \lambda_*^2 z_{i*}^T z_{i*} \\ F^{(k+1)} &= (1 - \lambda_*) F^{(k)} + \tilde{\delta}\lambda_* z_{i*}^T y\ , \end{aligned}$$

with starting values $S^{(0)} = 0$ and $F^{(0)} = 0$. If we store the products $z_i^T y$ before the execution of the algorithm, the only non-trivial computation required here is $\nabla f(\alpha^{(k)})_{i*}$ which was already computed to solve the subproblem in (6).

4.1 Randomized Frank–Wolfe iterations

Although the FW method is generally very efficient for structured problems with a sparse solution, it also has a number of practical drawbacks. For example, it is well known that the total number of iterations required by a FW algorithm can be large, thus making the optimization prohibitive on very large problems. Even when (4) has an analytical solution due to the problem structure, the resulting complexity depends on the problem size (Ñanculef et al. 2014), and can thus be impractical in cases where handling large-scale datasets is required. For example, in the specialization of the algorithm to problem (1), the main bottleneck is the computation of the FW vertex $i_*^{(k)}$ in (6) which corresponds to examining all the p candidate predictors and choosing the one most correlated with the current residuals (assuming the design matrix has been standardized s.t. the predictors have unit norm). Coincidentally, this is the same strategy underlying well-known methods for variable selection such as LARS and Forward Stepwise Regression (see Sect. 1).

A simple and effective way to avoid this dependence on p is to compute the FW vertex approximately, by limiting the search to a fixed number of extreme points on the boundary of the feasible set Σ (Schölkopf and Smola 2001; Frandi et al. 2014). Specialized to the Lasso problem (1), this technique can be formalized as extracting a random sample $\mathcal{S} \subseteq \{1, \ldots, p\}$ and solving

$$u^{(k)} = -\delta \operatorname{sign}\left(\nabla f\left(\alpha^{(k)}\right)_{i_{\mathcal{S}}^{(k)}}\right) \mathbf{e}_{i_{\mathcal{S}}^{(k)}}, \qquad \text{where } i_{\mathcal{S}}^{(k)} = \operatorname*{argmax}_{i \in \mathcal{S}} \left|\nabla f\left(\alpha^{(k)}\right)_i\right|. \tag{9}$$

Formally, one can think of a randomized FW iteration as the optimization of an approximate linear model, built by considering the partial gradient $\nabla f(\alpha^{(k)})_{|\mathcal{S}^{(k)}}$, i.e. the projection of the gradient onto the subspace identified by the sampled coordinates (Wang and Qian 2014). The number of coordinates of the gradient that need to be estimated with this scheme is $|\mathcal{S}|$ instead of p. If $|\mathcal{S}| \ll p$, this leads to a significant reduction in terms of computational overhead. Our stochastic specialization of the FW iteration for the Lasso problem takes thus the form of Algorithm 2. After selecting the variable $z_{i*} \in \mathcal{S}$ best correlated with the current vector of residuals, the algorithm updates the current solution along the segment connecting $z_{i*} \in \mathcal{S}$ with $\alpha^{(k)}$. Note how this approach differs from a method like LARS, where the direction to move the last iterate is equiangular to all the active variables. It also differs from CD, which can make active more than one variable at each "epoch" or cycle through the predictors. The algorithm computes the stepsize by looking explicitly to the value of the objective, which can be computed analytically without increasing the cost of the iteration. Finally, the method updates the vector of residuals and proceeds to the next iteration.

Algorithm 2 Randomized Frank–Wolfe step for the Lasso problem

```
1: Choose the sampling set $\mathcal{S}$ (see Sect. 4.5).
2: Search for the predictor best correlated with the vector of residuals $R^{(k)} = \left(y - X\alpha^{(k)}\right)$:
       $i_*^{(k)} = \operatorname*{argmax}_{i \in \mathcal{S}} \left|\nabla f(\alpha^{(k)})_i\right| \equiv \left|z_i^T R^{(k)}\right|.$
3: Set $\tilde{\delta}^{(k)} = -\delta \operatorname{sign}\left(\nabla f(\alpha^{(k)})_{i_*}\right)$.
4: Compute the step-size $\lambda^{(k)}$ using (8).
5: Update the vector of coefficients as
       $\alpha^{(k+1)} = (1 - \lambda^{(k)})\alpha^{(k)} + \tilde{\delta}\lambda^{(k)}\mathbf{e}_{i_*^{(k)}}.$
6: Update the vector of residuals $R^{(k)}$
       $R^{(k+1)} = (1 - \lambda^{(k)})R^{(k)} + \lambda^{(k)}\left(y - \tilde{\delta} z_{i_*^{(k)}}\right).$    (10)
```

Note that, although in this work we only discuss the basic Lasso problem, extending the proposed implementation to the more general ElasticNet model of Zou and Hastie (2005) is straightforward. The derivation of the necessary analytical formulae is analogous to the one shown above. Furthermore, an extension of the algorithm to solve ℓ_1-regularized logistic regression problems, another relevant tool in high-dimensional data analysis, can be easily obtained following the guidelines in Friedman et al. (2010).

4.2 Complexity and implementation details

In Algorithm 2, we compute the coordinates of the gradient using the method of residuals given by Eq. (7). Due to the randomization, this method becomes very advantageous with respect to the use of the alternative method based on the active covariates, even for very large p. Indeed, if we denote by s the cost of performing a dot product between a predictor z_i and another vector in $\mathbb{R}^m$, the overall cost of picking out the FW vertex in step 1 of our algorithm is $\mathcal{O}(s|\mathcal{S}|)$. Using the method of the active covariates would instead give an overall cost of $\mathcal{O}(s|\mathcal{S}|\|\alpha^{(k)}\|_0)$, which is always worse. Note however that this method may be better than the method of the residuals in a deterministic implementation by using caching tricks as proposed in Friedman et al. (2007), Friedman et al. (2010). For instance, caching the dot products between all the predictors and the active ones and keeping updated all the coordinates of the gradient would cost $\mathcal{O}(p)$ except when new active variables appear in the solution, in which case the cost becomes $\mathcal{O}(ps)$. However, this would allow to find the FW vertex in $\mathcal{O}(p)$ operations. In this scenario, the fixed $\mathcal{O}(sp)$ cost of the method of residuals may be worse if the Lasso solution is very sparse. It is worth noting that the dot product cost s is proportional to the number of nonzero components in the predictors, which in typical high-dimensional problems is significantly lower than m.

In the current implementation, the term $\sigma_i := z_i^T y$ will be pre-computed for any $i = 1, 2, \ldots, p$ before starting the iterations of the algorithm. This allows to write (7) as $-z_i^T R^{(k)} = -\sigma_i + z_i^T X\alpha^{(k)}$. Equation (10) for updating residuals can therefore be replaced by an equation to update $p^{(k)} = X\alpha^{(k)}$, eliminating the dependency on m.

4.3 Relation to SVM algorithms and sparsity certificates

The previous implementation suggests that the FW algorithm will be particularly suited to the case $p \gg m$ where a regression problem has a very large number of features but not so many training points. It is interesting to compare this scenario to the situation in SVM problems. In the SVM case, the FW vertices correspond to training points, and the standard FW algorithm is able to quickly discover the relevant "atoms" (the support vectors), but has no particular advantage when handling lots of features. In contrast, in Lasso applications, where we are using the z_i's as training points, the situation is somewhat inverted: the algorithm should discover the critical features in at most $\mathcal{O}(1/\epsilon)$ iterations and guarantee that at most $\mathcal{O}(1/\epsilon)$ attributes will be used to perform predictions. This is, indeed, the scenario in which Lasso is used for several applications of practical interest, as problems with a very large number of features arise frequently in specific domains like bio-informatics, web and text analysis and sensor networks.

In the context of SVMs, the randomized FW algorithm has been already discussed in Frandi et al. (2014). However, the results in the mentioned paper were experimental in nature, and did not contain a proof of convergence, which is instead provided in this work. Note that, although we have presented the randomized search for the specific case of problem (1), the technique applies more generally to the case where Σ is a polytope (or has a separable structure with every block being a polytope, as in Lacoste-Julien et al. (2013)). We do not feel this hypothesis to be restrictive, as basically every practical application of the FW algorithm proposed in the literature falls indeed into this setting.

4.4 Convergence analysis

We show that the stochastic FW converges (in the sense of expected value) with the same rate as the standard algorithm. First, we need the following technical result.

Lemma 1 *Let $\mathcal{S}$ be picked at random from the set of all equiprobable κ-subsets of $\{1, \ldots, p\}$, $1 \le \kappa \le p$, and let v be any vector in $\mathbb{R}^p$. Then*

$$\mathbb{E}\left[\left(\sum_{i\in\mathcal{S}} \mathbf{e}_i\mathbf{e}_i^T\right)v\right] = \frac{\kappa}{p}\,v.$$

Proof Let $A_{\mathcal{S}} = \sum_{i\in\mathcal{S}} \mathbf{e}_i\mathbf{e}_i^T$ and $(A_{\mathcal{S}})_{ij}$ an element of $A_{\mathcal{S}}$. For $i \neq j$, $(A_{\mathcal{S}})_{ij} = 0$ and $\mathbb{E}\left[\,(A_{\mathcal{S}})_{ij}\,\right] = 0$. For $i = j$, $(A_{\mathcal{S}})_{ij}$ is a Bernoulli random variable with expectation $\frac{\kappa}{p}$. In fact, $(A_{\mathcal{S}})_{ii} = 1$ iff $i \in \mathcal{S}$; as there are $\binom{p-1}{\kappa-1}\kappa$-subsets of $\{1, \ldots, p\}$ containing i,

$$\mathbb{P}(i \in \mathcal{S}) = \binom{p-1}{\kappa-1}\binom{p}{\kappa}^{-1} = \kappa/p = \mathbb{P}\big((A_{\mathcal{S}})_{ii} = 1\big) = \mathbb{E}\left[\,(A_{\mathcal{S}})_{ii}\,\right].$$

Therefore, for $i \in \{1, \ldots, p\}$,

$$\mathbb{E}\left[\,(A_{\mathcal{S}})_i\,v\right] = \mathbb{E}\left[\sum_{j=1}^{p}(A_{\mathcal{S}})_{ij}\,v_j\right] = \sum_{j=1}^{p} v_j\,\mathbb{E}\left[\,(A_{\mathcal{S}})_{ij}\,\right] = \frac{\kappa}{p}\,v_i.$$

□

Note that selecting a random subset $\mathcal{S}$ of size κ to solve (9) is equivalent to (i) building a random matrix $A_{\mathcal{S}}$ as in Lemma 1, (ii) computing the restricted gradient $\tilde{\nabla} f = \frac{p}{\kappa}A_{\mathcal{S}}\nabla f(\alpha^{(k)})$ and then (iii) solving the linear sub-problem (6) substituting $\nabla f(\alpha^{(k)})$ by $\tilde{\nabla} f$. In other words, the proposed randomization can be viewed as approximating $\nabla f(\alpha^{(k)})$ by $\tilde{\nabla} f$. Lemma 1 implies that $\mathbb{E}[\tilde{\nabla} f] = \nabla f(\alpha^{(k)})$, which is the key to prove our main result.

Proposition 2 *Let α^* be an optimal solution of problem (3). Then, for any $k \geq 1$, the iterates of Algorithm 1 with the randomized search rule satisfy*

$$\mathbb{E}_{\mathcal{S}^{(k)}}\left[f\left(\alpha^{(k)}\right)\right] - f\left(\alpha^*\right) \leq \frac{4C_f}{k+2},$$

where $\mathbb{E}_{\mathcal{S}^{(k)}}$ denotes the expectation with respect to the k-th random sampling.

This result has a similar flavor to that in Lacoste-Julien et al. (2013), and the analysis is similar to the one presented in Wang and Qian (2014). However, in contrast to the above works, we do not assume any structure in the optimization problem or in the sampling. A detailed proof can be found in the "Appendix". As in the deterministic case, Proposition 2 implies a complexity bound of $\mathcal{O}(1/\varepsilon)$ iterations to reach an approximate solution $\alpha^{(k)}$ such that $\mathbb{E}_{\mathcal{S}^{(k)}}[f(\alpha^{(k)})] - f(\alpha^*) \leq \varepsilon$.

4.5 Choosing the sampling size

When using a randomized FW iteration it is important to choose the sampling size in a sensible way. Indeed, some recent works showed how this choice entails a tradeoff between accuracy (in the sense of premature stopping) and complexity, and henceforth CPU time (Frandi et al. 2014). This kind of approximation is motivated by the following result, which suggests that it is reasonable to pick $|\mathcal{S}| \ll p$.

Theorem 1 [(Schölkopf and Smola 2001), Theorem 6.33] *Let $\mathcal{D} \subset \mathbb{R}$ s.t. $|\mathcal{D}| = p$ and let $\mathcal{D}' \subset \mathcal{D}$ be a random subset of size κ. Then, the probability that the largest element in $\mathcal{D}'$ is greater than or equal to $\tilde{p}$ elements of $\mathcal{D}$ is at least $1 - (\frac{\tilde{p}}{p})^{\kappa}$.*

The value of this result lies in the ability to obtain probabilistic bounds for the quality of the sampling *independently of the problem size p*. For example, in the case of the Lasso problem, where $\mathcal{D} = \{|\nabla f(\alpha^{(k)})_1|, \ldots, |\nabla f(\alpha^{(k)})_p|\}$ and $\mathcal{D}' = \{|\nabla f(\alpha^{(k)})_i| \text{ s.t. } i \in \mathcal{S}\}$, it is easy to see that it suffices to take $|\mathcal{S}| \approx 194$ to guarantee that, with probability at least 0.98, $|\nabla f(\alpha^{(k)})_{i_{\mathcal{S}}^{(k)}}|$ lies between the 2% largest gradient components (in absolute value), independently of p. This kind of sampling has been discussed for example in Frandi et al. (2015).

The issue with this strategy is that, for problems with very sparse solutions (which is the case for strong levels of regularization), even a large confidence interval does not guarantee that the algorithm can sample a good vertex in most of the iterations. Intuitively, the sampling strategy should allow the algorithm to detect the set of vertices active at the optimum, which correspond, at various stages of the optimization process, to descent directions for the objective function. In sparse approximation problems, extracting a sampling set without relevant features may imply adding "spurious" components to the solution, reducing the sparseness of the model we want to find.

A more effective strategy in this context would be to ask for a certain probability that the sampling will include at least one of the "optimal" features. Letting $\mathcal{S}^*$ be the index set of the active vertices at the optimum, and denoting $s = |\mathcal{S}^*|$ and $\kappa = |\mathcal{S}|$, we have

$$P(\mathcal{S}^* \cap \mathcal{S} = \emptyset) = \prod_{j=0}^{\kappa-1} \left(1 - \frac{s}{p-j}\right) \leq \left(1 - \frac{s}{p}\right)^{\kappa}, \tag{11}$$

with the latter inequality being a reasonable approximation if $\kappa \ll p$. From (11), we can guarantee that $\mathcal{S}^* \cap \mathcal{S} \neq \emptyset$ with probability at least ρ by imposing:

$$\left(1 - \frac{s}{p}\right)^{\kappa} \leq (1-\rho) \;\Leftrightarrow\; \kappa \geq \frac{\ln(1-\rho)}{\ln\left(1 - \frac{s}{p}\right)} = \frac{\ln(\text{confidence})}{\ln(\text{sparseness})}. \tag{12}$$

On the practical side, this sampling strategy often implies taking a larger κ. Assuming that the fraction of relevant features (s/p) is constant, we essentially get the bound for κ provided by Theorem 1, which is independent of p. However, for the worst case $s/p \to 0$, we get

$$\frac{\ln(1-\rho)}{\ln\left(1 - \frac{s}{p}\right)} \approx \left(\frac{-\ln(1-\rho)}{s}\right) p, \tag{13}$$

which suggests to use a sampling size proportional to p. For this reason, for the problems considered in this paper, we chose to adopt a simple heuristic where κ is set to a small fraction of p. Despite its simplicity, this strategy works well in practice, as shown by the experiments in the next Section.

A more involved strategy, which exploits the incremental structure of the FW algorithm, would be using a large κ at early iterations and smaller values of κ as the solution gets more dense. The idea here is that if the optimal solution is very sparse the algorithm requires few expensive iterations to converge, while in contrast, when the solution is dense, it will require more, but faster, iterations (e.g. for a confidence $1 - \rho = 0.98$ and $s/p = 0.02$ the already mentioned $\kappa = 194$ suffices).

5 Numerical experiments

In this section, we assess the practical effectiveness of the randomized FW algorithm by performing experiments on both synthetic datasets and real-world benchmark problems with

Table 1 List of the benchmark datasets used in the experiments

Dataset	m	t	p	Type
Synthetic-10000	200	200	$10,000$	Synthetic
Synthetic-50000	200	200	$50,000$	Synthetic
Pyrim	74	$--$	$201,376$	Regression
Triazines	186	$--$	$635,376$	Regression
E2006-tfidf	$16,087$	$3,308$	$150,360$	Regression
E2006-log1p	$16,087$	$3,308$	$4,272,227$	Regression
Dorothea	800	150	$100,000$	Classification
URL-reputation	$2,172,130$	$220,000$	$3,231,961$	Classification
KDD2010-algebra	$8,407,752$	$510,302$	$20,216,830$	Classification

hundreds of thousands or even millions of features. The characteristics of the datasets are summarized in Table 1, where we denote by m the number of training examples, by t the number of test examples, and by p the number of features. The synthetic datasets were generated with the Scikit-learn function `make_regression` (Pedregosa et al. 2011). Each of them comes in two versions corresponding to a different number of relevant features in the true linear model used to generate the data (32 and 100 features for the problem of size $p = 10000$, and 158 and 500 features for that of size $p = 50000$). The real large-scale regression problems **E2006-tfidf** and **E2006-log1p**, the datasets **Pyrim** and **Triazines**, and the classification problem **KDD2010-algebra** are available from Chang and Lin (2011). The datasets **Dorothea** and **URL-reputation** can be downloaded from the UCI Machine Learning Repository (Lichman 2013).

In assessing the performance of our method, we used as a baseline the following algorithms, which in our view, and according to the properties summarized in Table 2, can be considered among the most competitive solvers for Lasso problems:

– The well-known CD algorithm by Friedman et al., as implemented in the Glmnet package (Friedman et al. 2010). This method is highly optimized for the Lasso and is widely considered as one the most popular and efficient solvers in this context.
– The SCD algorithm as described in Shalev-Shwartz and Tewari (2011), which is significant both for being a stochastic method and for having better theoretical guarantees than the standard cyclic CD.
– The Accelerated Gradient Descent with projections for both the regularized and the constrained Lasso, as this algorithm guarantees an optimal complexity bound. We choose as a reference the implementation in the SLEP package by Liu et al. (2009).

Among other possible first-order methods, the classical SGD suffers from a worse convergence rate, and its variant SMIDAS has a complexity bound which depends on p, thus we did not include them in our experiments. Indeed, the authors of Shalev-Shwartz et al. (2010) conclude that SCD is both faster and produces significantly sparser models compared to SGD. Finally, although the recent GeoLasso algorithm of Zhou et al. (2015) is interesting because of its close relationship with FW, its running time and memory requirements are clearly higher compared to the above solvers.

Since an appropriate level of regularization needs to be automatically selected in practice, the algorithms are compared by computing the entire regularization path on a range of values of the regularization parameters λ and δ (depending on whether the method solves

Table 2 Methods proposed for scaling the Lasso and their complexities

Approach	Form	Number of iterations	Complexity per iteration	Sparse Its.
Accelerated Gradient + Proj. (Liu and Ye 2009)	(1)	$\mathcal{O}(1/\sqrt{\epsilon})$	$\mathcal{O}(mp+p)\dagger_1$	No
Accelerated Gradient + Reg. Proj. (Liu and Ye 2010)	(2)	$\mathcal{O}(1/\sqrt{\epsilon})$	$\mathcal{O}(mp+p)\dagger_1$	No
Cyclic Coordinate Descent (CD) (Friedman et al. 2007, 2010)	(2)	Unknown	$\mathcal{O}(mp)\dagger_2$	Yes
Stochastic Gradient Descent (SGD) (Langford et al. 2009)	(2)	$\mathcal{O}(1/\epsilon^2)$	$\mathcal{O}(p)$	No
Stochastic Mirror Descent (Shalev-Shwartz and Tewari 2011)	(2)	$\mathcal{O}(\log(p)/\epsilon^2)$	$\mathcal{O}(p)$	No
GeoLasso (Zhou et al. 2015)	(1)	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(mp+a^2)$	Yes
Frank–Wolfe (FW) (Jaggi 2013)	(1)	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(mp)$	Yes
Stochastic Coord. Descent (SCD) (Richtárik and Takáĉ 2014)	(2)	$\mathcal{O}(p/\epsilon)$	$\mathcal{O}(m)\dagger_3$	Yes
Stochastic Frank–Wolfe	(1)	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(m\lvert\mathcal{S}\rvert)$	Yes

Here, a denotes the number of active features at a given iteration, which in the worst case is $a = \text{rank}(X) \leq \min(m, p)$. A method is said to have *sparse iterations* if a non trivial bound for the number of non-zero entries of each iterate holds at any moment. $\dagger_1\mathcal{O}(p)$ is required for the projections. $\dagger_2$ An iteration of cyclic coordinate descent corresponds to a complete cycle through the features. $\dagger_3$ An iteration of SCD corresponds to the optimization on a single feature

the penalized or the constrained formulation). Specifically, we first estimate two intervals $[\lambda_{\min}, \lambda_{\max}]$ and $[\delta_{\min}, \delta_{\max}]$, and then solve problems (2) and (1) on a 100-point parameter grid in logarithmic scale. For the penalized Lasso, we use $\lambda_{\min} = \lambda_{\max}/100$, where $\lambda_{\max}$ is determined as in the Glmnet code. Then, to make the comparison fair (i.e. to ensure that all the methods solve the same problems according to the equivalence in Sect. 2), we choose for the constrained Lasso $\delta_{\max} = \|\alpha_{\min}\|_1$ and $\delta_{\min} = \delta_{\max}/100$, where $\alpha_{\min}$ is the solution obtained by Glmnet with the smallest regularization parameter $\lambda_{\min}$ and a high precision ($\varepsilon = 10^{-8}$). The idea is to give the solvers the same "sparsity budget" to find a a solution of the regression problem.

Warm-start strategy

As usually done in these cases, and for all the methods, we compute the path using a warm-start strategy where each solution is used as an initial guess for the optimization with the next value of the parameter. Note that we always start from the most sparse solution. This means that in the cases of CD, SCD and regularized SLEP we start from $\lambda_{\max}$ towards $\lambda_{\min}$, while for FW and constrained SLEP we go from $\delta_{\min}$ towards $\delta_{\max}$. Furthermore, since $\delta < \|\alpha^R\|_1$, where α^R is the unconstrained solution, we know that the solution will lie on the boundary, therefore we adopt a heuristic strategy whereby the previous solution is scaled so that its ℓ_1-norm is δ. Both algorithms are initialized with the null solution as the initial guess. Regarding the stopping criterion for each problem in the path, we stop the current run when $\|\alpha^{(k+1)} - \alpha^{(k)}\|_\infty \leq \varepsilon$ for all the algorithms. Other choices are possible (for example, FW methods are usually stopped using a duality gap-based criterion (Jaggi 2013)), but this is the condition used by default to stop CD in Glmnet. A value of $\varepsilon = 10^{-3}$ is used in the following experiments. All the considered algorithms have been coded in C++. The code and datasets used in this paper are available from public repositories on Github (https://github.com/efrandi/FW-Lasso) and Dataverse (https://goo.gl/PTQ05R), respectively. The SLEP package has a Matlab interface, but the key functions are all coded in C. Overall, we believe our comparison can be considered very fair. We executed the experiments on a 3.40GHz Intel i7 machine with 16GB of main memory running CentOS. For the randomized experiments, results were averaged over 10 runs.

5.1 "Sanity Check" on the synthetic datasets

The aim of these experiments is not to measure the performance of the algorithms (which will be assessed below on seven real-life datasets of large and very large size), but rather to compare their ability to capture the evolution of the most relevant features of the models, and discuss how this relates to their behaviour from an optimization point of view. To do this, we monitor the values of the 10 most relevant features along the path, as computed by both the CD and FW algorithms, and plot the results in Figs. 1 and 2. To determine the relevant variables, we use as a reference the regularization path obtained by Glmnet with $\varepsilon = 10^{-8}$ (which is assumed to be a reasonable approximation of the exact solution), and compute the 10 variables having, on average, the highest absolute value along the path. As this experiment is intended mainly as a sanity check to verify that our solver reconstructs the solution correctly, we do not include other algorithms in the comparison. In order to implement the random

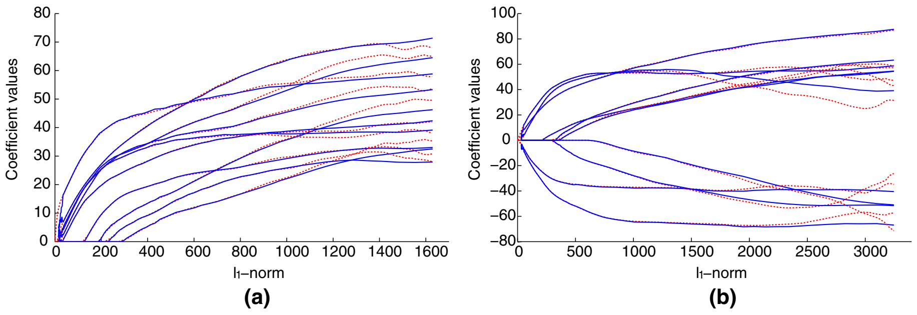

Fig. 1 Growth of the ten most significant features on the synthetic problem of size 10000, with 32 (**a**) and 100 (**b**) relevant features. Results for CD are in *red dashed lines*, and in *blue continuous lines* for FW (Color figure online)

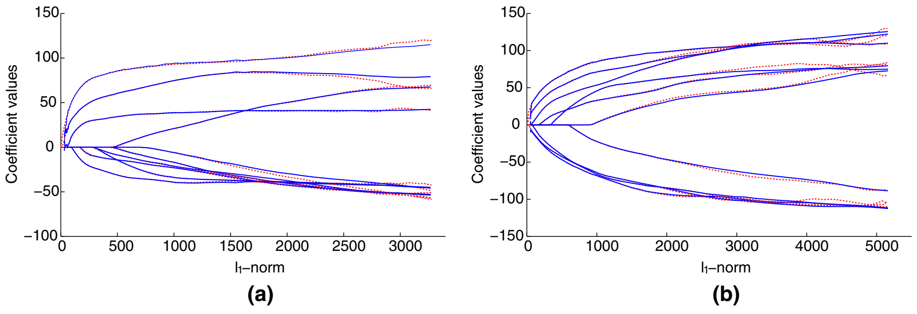

Fig. 2 Growth of the 10 most significant features on the synthetic problems of size 50000, with 158 (**a**) and 500 (**b**) relevant features. Results for CD are in *red dashed lines*, and in *blue continuous lines* for FW (Color figure online)

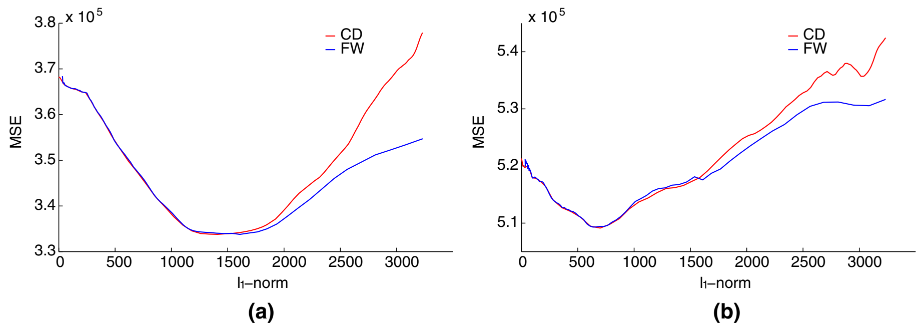

Fig. 3 Test error (ℓ_1-norm vs. MSE) for problems **Synthetic-10000** (100 relevant features) **Synthetic-50000** (158 relevant features). Results for CD are in *red*, and in *blue* for FW (Color figure online)

sampling strategy in the FW algorithm, we first calculated the average number of active features along the path, rounded up to the nearest integer, as an empirical estimate of the sparsity level. Then, we chose $|\mathcal{S}|$ based on the probability ρ of capturing at least one of the relevant features at each sampling, according to (13). A confidence level of 99% was used for this experiment, leading to sampling sizes of 372 and 324 points for the two problems of size 10000, and of 1616 and 1572 points for those of size 50000.

Figure 3 depicts, for two of the considered problems, the prediction error on the test set along the path found by both algorithms. It can be seen that both methods are able to find the same value of the best prediction error (i.e. to correctly identify the best model). The FW algorithm also seems to obtain slightly better results towards the end of the path, consistently with the fact that the coefficients of the most relevant variables tend to be more stable, compared with CD, when the regularization is weaker.

5.2 Results on large-scale datasets

In this section, we report the results on the problems **Pyrim**, **Triazines**, **E2006-tfidf**, **E2006-log1p**, **Dorothea**, **URL-reputation** and **KDD2010-algebra**. These datasets represent actual large-scale problems of practical interest. The **Pyrim** and **Triazines** datasets stem from two quantitative structure-activity relationship (QSAR) problems, where the biological responses of a set of molecules are measured and statistically related to the molecular structure on their surface. We expanded the original feature set by means of product features, i.e. modeling the response variable y as a linear combination of polynomial basis functions, as suggested in Huang et al. (2010). For this experiment, we used product features of order 5 and 4 respectively, which leads to large-scale regression problems with $p = 201,376$ and $p = 635,376$. Problems **E2006-tfidf** and **E2006-log1p** stem instead from the real-life NLP task of predicting the risk of investment (i.e. the stock return volatility) in a company based on available financial reports (Kogan et al. 2009). Finally, the three classification problems correspond to tasks related to molecular bioactivity prediction (**Dorothea**), malicious URL detection (**URL-reputation**), and educational data mining (**KDD1010-algebra**). For benchmarking purposes, these tasks were cast as Lasso problems by treating the binary responses as real continuous variables.

To implement the random sampling for the FW algorithm, we use the strategy described in Sect. 4.5, where we set $|\mathcal{S}|$ to a fixed, small fraction of the total number of features. Our choices are summarized in Table 3.

Table 3 Sizes of the sampling set $|\mathcal{S}|$ for the large-scale datasets

% of p	1 %	2 %	3 %
Pyrim	2,014	4,028	6,402
Triazines	6,354	12,708	19,062
E2006-tfidf	1,504	3,008	4,511
E2006-log1p	42,723	85,445	128,167
Dorothea	1,000	2,000	3,000
URL-reputation	32,320	64,640	96,959
KDD2010-algebra	202,169	404,337	606,505

Table 4 Results for the baseline solvers on the large-scale regression problems **Pyrim**, **Triazines**, **E2006-tfidf** and **E2006-log1p**

	CD	SCD	SLEP Reg.	SLEP Const.
Pyrim				
Time (s)	6.22e+00	1.59e+01	5.43e+00	6.86e+00
Iterations	2.54e+02	1.44e+02	1.00e+02	1.12e+02
Dot products	2.08e+07	2.90e+07	8.56e+07	1.29e+08
Active features	68.4	116.6	3, 349	13, 030
Triazines				
Time (s)	2.75e+01	8.42e+01	4.27e+01	5.93e+01
Iterations	2.62e+02	1.59e+02	1.01e+02	1.11e+02
Dot products	6.80e+07	1.01e+08	2.87e+08	4.67e+08
Active features	150.0	330.8	29,104	130,303
E2006-tfidf				
Time (s)	9.10e+00	2.19e+01	1.24e+01	2.27e+01
Iterations	3.48e+02	2.01e+02	1.06e+02	2.50e+02
Dot products	2.04e+07	3.03e+07	5.97e+07	1.37e+08
Active features	149.5	275.3	444.8	724.3
E2006-log1p				
Time (s)	1.60e+02	4.92e+02	1.00e+02	1.42e+02
Iterations	3.55e+02	1.99e+02	1.11e+02	1.18e+02
Dot products	5.73e+08	8.50e+08	1.78e+09	2.85e+09
Active features	281.3	1, 158.2	12,806	54,704

As a measure of the performance of the considered algorithms, we report the CPU time in seconds, the total number of iterations and the number of requested dot products (which account for most of the required running time for all the algorithms)[1] along the entire regularization path. Note that, in assessing the number of iterations, we consider one complete cycle of CD to be roughly equivalent to a full deterministic iteration of FW (since in both cases the complexity is determined by a full pass through all the coordinates) and to p random coordinate explorations in SCD. Finally, in order to evaluate the sparsity of the solutions, we report the average number of active features along the path. Results are displayed in

[1] Note that the SLEP code uses highly optimized libraries for matrix multiplication, therefore matrix-vector computations can be faster than naive C++ implementations.

Table 5 Results for the baseline solvers on the large-scale classification problems **Dorothea**, **URL-reputation** and **KDD2010-algebra**

	CD	SCD	SLEP Reg.	SLEP Const.
Dorothea				
Time (s)	3.34e+00	1.17e+01	4.10e+00	6.19 e+00
Iterations	4.45e+02	2.99e+02	2.30e+02	3.01e+02
Dot products	1.22e+07	2.99e+07	6.92e+07	1.02e+08
Active features	134.8	153.3	211.1	731.5
URL-reputation				
Time (s)	2.55e+02	7.86e+02	1.66e+03	5.65e+03
Iterations	4.44e+02	3.01e+02	1.77e+02	5.92e+02
Dot products	4.53e+08	9.73e+08	1.98e+09	4.89e+009
Active features	53.4	77.9	126.8	52.44
KDD2010				
Time (s)	6.15e+02	2.33e+03	8.86e+02	4.12e+03
Iterations	2.27e+02	1.59e+02	1.04e+02	2.22e+02
Dot products	2.08e+09	3.21e+09	8.92e+09	1.88e+10
Active features	906.0	1,444.1	1,825.4	1,978.5

Table 6 Performance metrics for stochastic FW on the large-scale regression problems **Pyrim**, **Triazines**, **E2006-tfidf** and **E2006-log1p**

	FW 1%	FW 2%	FW 3%
Pyrim			
Time (s)	2.28e−01	4.47e−01	6.60e−01
Speed-up	27.3×	13.9×	9.4×
Iterations	2.77e+02	2.80e+02	2.77e+02
DotProd	7.61e+05	1.53e+06	2.28e+06
Active features	27.6	28.1	27.9
Triazines			
Time (s)	2.61e+00	5.31e+00	8.19e+00
Speed-up	10.5×	5.2×	3.4×
Iterations	7.15e+02	7.29e+02	7.43e+02
DotProd	5.18e+06	1.06e+07	1.61e+07
Active features	120.6	117.5	118.7
E2006-tfidf			
Time (s)	8.83e−01	1.76e+00	2.74e+00
Speed-up	10.3×	5.2×	3.3×
Iterations	1.27e+03	1.35e+03	1.41e+03
DotProd	1.97e+06	4.35e+06	6.84e+06
Active features	123.7	125.8	127.1
E2006-log1p			
Time (s)	1.93e+01	4.14e+01	6.59e+01
Speed-up	8.3×	3.9×	2.4×
Iterations	1.75e+03	1.91e+03	1.99e+03
DotProd	7.90e+07	1.71e+08	2.68e+08
Active features	196.9	199.8	203.7

Table 7 Performance metrics for stochastic FW on the large-scale classification problems **Dorothea**, **URL-reputation** and **KDD2010-algebra**

	FW 1%	FW 2%	FW 3%
Dorothea			
Time (s)	1.31e−01	2.48e−01	3.78e−01
Speed-up	25.5×	13.5×	8.84×
Iterations	8.04e+02	8.09e+02	8.46e+02
DotProd	9.17e+05	1.83e+06	2.84e+06
Active features	50.9	52.5	54.8
URL-reputation			
Time (s)	1.50e+01	2.27e+01	3.13e+01
Speed-up	17.0×	11.2×	8.15×
Iterations	5.33e+02	5.50e+02	5.66e+02
DotProd	2.04e+07	4.20e+07	6.45e+07
Active features	25.2	26.5	28.1
KDD2010			
Time (s)	1.68e+02	3.42e+02	5.21e+02
Speed-up	3.66×	1.80×	1.18×
Iterations	2.70e+03	2.78e+03	2.83e+03
DotProd	4.53e+08	1.16e+09	1.77e+09
Active features	423.6	428.0	433.6

Tables 4, 5 (baseline methods) and Tables 6, 7 (stochastic FW). In the latter, the speed-ups with respect to the CD algorithm are also reported. It can be seen how for all the choices of the sampling size the FW algorithm allows for a substantial improvement in computational performance, as confirmed by both the CPU times and the machine-independent number of requested dot products (which are roughly proportional to the running times). The plain SCD algorithm performs somewhat worse than CD, something we attribute mainly to the fact that the Glmnet implementation of CD is a highly optimized one, using a number of *ad-hoc* tricks tailored to the Lasso problem that we decided to preserve in our comparison. If we used a plain implementation of CD, we would expect to obtain results very close to those exhibited by SCD.

Furthermore, FW is always able to find the sparsest solution among the considered methods. The extremely large gap in average sparsity between FW and CD on one side, and the SLEP solvers on the other, is due to the fact that the latter compute in general dense iterates. Although the Accelerated Gradient Descent solver is fast and competitive from an optimization point of view, providing always the lower number of iterations as predicted by the theory, it is not able to keep the solutions sparse along the path. This behavior clearly highlights the advantage of using incremental approximations in the context of sparse recovery and feature selection. Importantly, note that the small number of features found by FW is not a result of the randomization technique: it is robust with respect to the sampling size, and additional experiments performed using a deterministic FW solver revealed that the average number of nonzero entries in the solution does not change even if the randomization is completely removed.

To better assess the effect of using an incremental algorithm in obtaining a sparse model, we plot in Fig. 4 the evolution of the number of active features along the path on problems **E2006-tfidf** and **E2006-log1p**. It can be clearly seen how CD and FW (with the latter performing

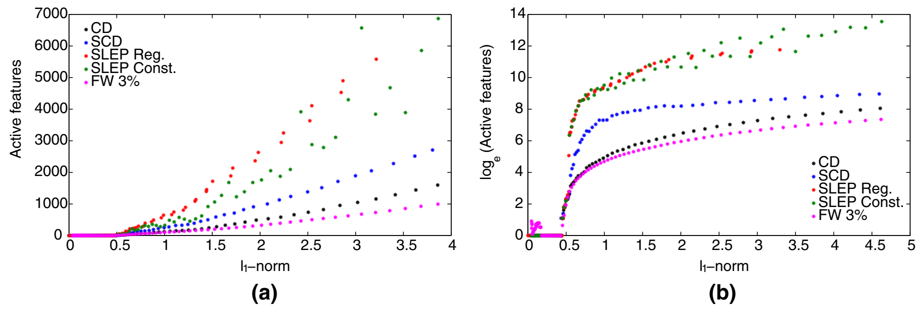

Fig. 4 Sparsity patterns (ℓ_1-norm vs. active coordinates) for problems **E2006-tfidf** (**a**) and **E2006-log1p** (**b**). The latter is plotted in a natural logarithmic scale due to the high number of features found by the SLEP solvers (Color figure online)

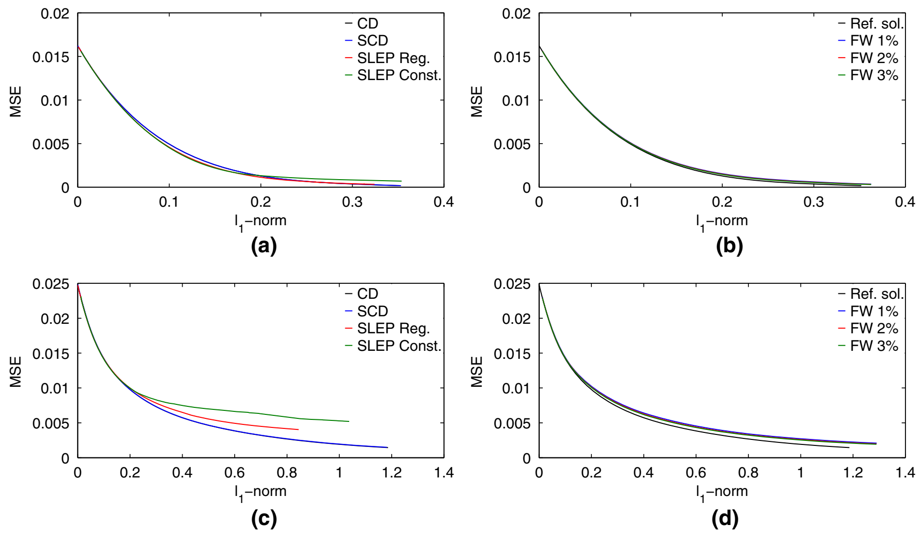

Fig. 5 Training *error curves* (ℓ_1-norm vs. MSE) for problems **Pyrim** (*top*) and **Triazines** (*bottom*) for CD, SCD and SLEP (*left*) and stochastic FW (*right*) (Color figure online)

the best overall) are able to recover sparser solutions, and can do so without losing accuracy in the model, as we discuss next.

In order to evaluate the accuracy of the obtained models, we plot in Figs. 5–7 the mean square error (MSE) against the ℓ_1-norm of the solution along the regularization path, computed both on the original training set (curves 5a–d, 6a,c and 7a,c) and on the test set (curves 6b,d and 7b,d). Figure 5 reports only the training error, as the corresponding problems did not come with a test set. Note that the value of the objective function in problem (1) coincides with the mean squared error (MSE) on the training set, therefore the training error plots effectively depict the convergence of the FW algorithms. For the sake of completeness, we also report in Fig. 8 the training and test error rates for one of the classification problems, **Dorothea**. First of all, we can see how the decrease in the objective value is basically identical in all cases, which indicates that with our sampling choices the use of a randomized algorithm does not affect the optimization accuracy. Second, the test error curves show that the predictive capability of all the FW models is competitive with that of the models found by

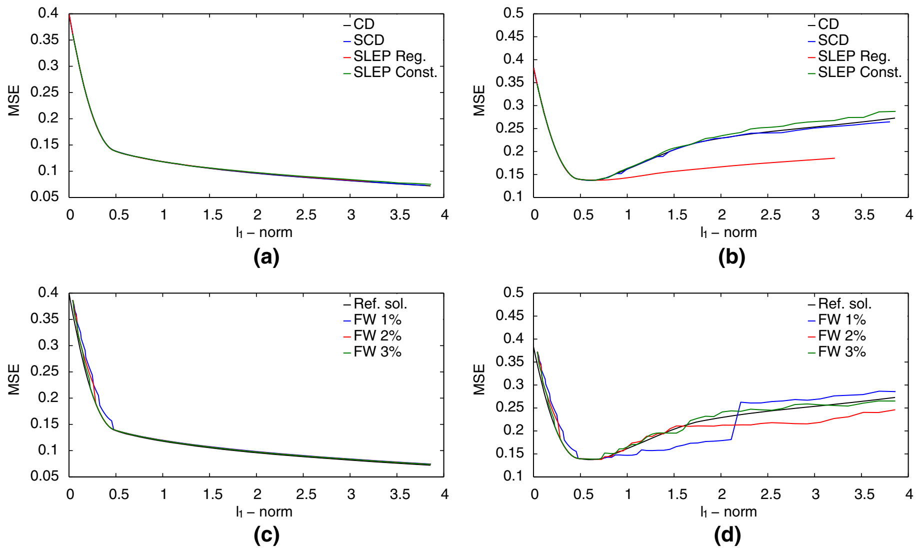

Fig. 6 *Error curves* (ℓ_1-norm vs. MSE) for problem **E2006-tfidf**: on *top*, training error (**a**) and test error (**b**) for CD, SCD and SLEP; on *bottom*, training error (**c**) and test error (**d**) for stochastic FW (Color figure online)

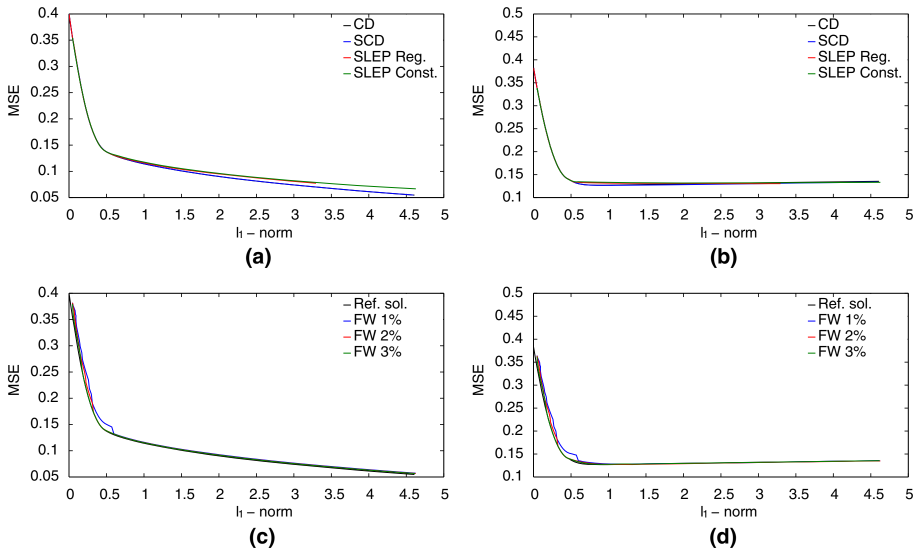

Fig. 7 *Error curves* (ℓ_1-norm vs. MSE) for problem **E2006-log1p**: on *top*, training error (**a**) and test error (**b**) for CD, SCD and SLEP; on *bottom*, training error (**c**) and test error (**d**) for stochastic FW (Color figure online)

the CD algorithm (particularly in the case of the larger problem **E2006-log1p**). Looking at Figs. 6 and 7, it is also important to note that in all cases the best model, corresponding to the minimum of the test error curves, is found for a relatively low value of the constraint parameter, indicating that sparse solutions are preferable and that solutions involving more variables tend to cause overfitting, which is yet another incentive to use algorithms that can naturally induce sparsity. Again, it can be seen how the minima of all the curves coincide, indicating

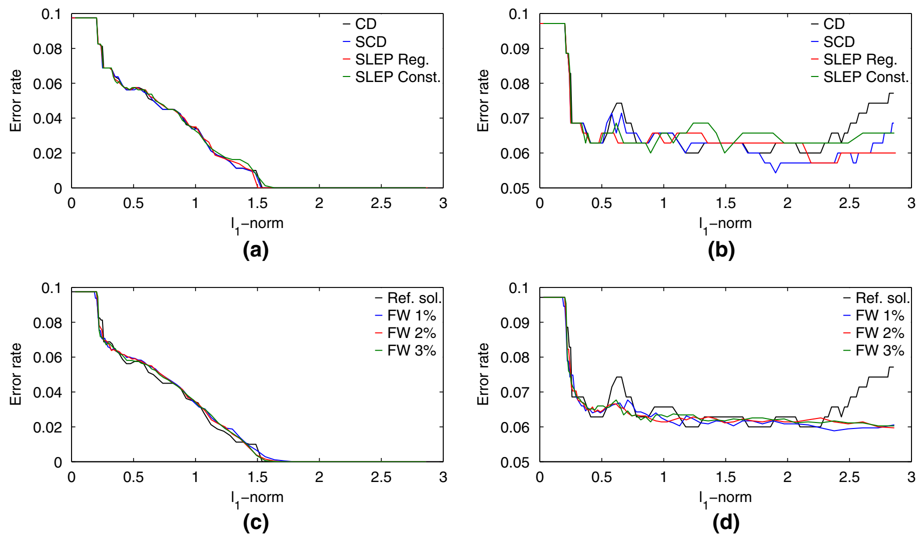

Fig. 8 *Error curves* (ℓ_1-norm vs. misclassification rate) for problem **Dorothea**: on *top*, training error (**a**) and test error (**b**) for CD, SCD and SLEP; on *bottom*, training error (**c**) and test error (**d**) for stochastic FW (Color figure online)

that all the algorithms are able to correctly identify the best compromise between sparsity and training error. The fact that we are able to attain the same models obtained by a highly efficient algorithm (tailored for the Lasso) such as Glmnet using a sampling size as small as 3% of the total number of features is particularly noteworthy. Combined with the consistent advantages in CPU time over other competing solvers and its attractive sparsity properties, it shows how the randomized FW represents a solid, high-performance option for solving high-dimensional Lasso problems. Finally, we note that even on the classification problem **Dorothea** FW is able to obtain more accurate models than CD, particularly towards the end of the path. We remark, though, that this experiment has mainly an illustrative purpose, and that solving classification tasks is not among the aims of the algorithm presented here.

6 Conclusions and perspectives

In this paper, we have studied the practical advantages of using a randomized Frank–Wolfe algorithm to solve the constrained formulation of the Lasso regression problem on high-dimensional datasets involving a number of variables ranging from the hundred thousands to a few millions. We have presented a theoretical proof of convergence based on the expected value of the objective function. Our experiments show that we are able to obtain results that outperform those of other state-of-the-art solvers such as the Glmnet algorithm, a standard among practitioners, without sacrificing the accuracy of the model in a significant way. Importantly, our solutions are consistently more sparse than those found using several popular first-order methods, demonstrating the advantage of using an incremental, greedy optimization scheme in this context.

In a future work, we intend to address the issue of whether it is possible to find suitable sampling conditions which can lead to a stronger stochastic convergence result, i.e. to certifiable probability bounds for approximate solutions. A more detailed convergence analysis taking into account higher order moments beyond the expected value would also be in our view a valuable contribution. Finally, we remark that the proposed approach can be readily extended to other similar problems such as ElasticNet or more general ℓ_2-regularized problems such as logistic regression, or to related applications such as the sparsification of SVM models. Another possibility to tackle various Lasso formulations is to exploit an equivalent formulation in terms of SVMs, an area where FW methods have already shown promising results. Together, all these elements strengthen the conclusions of our previous research work, showing that FW algorithms can provide a complete and flexible framework to efficiently solve a wide variety of large-scale machine learning and data mining problems.

Acknowledgments The authors wish to thank three anonymous reviewers for their valuable comments. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC AdG A-DATADRIVE-B (290923). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information. Research Council KUL: GOA/10/09 MaNet, CoE PFV/10/002 (OPTEC), BIL12/11T; Flemish Government: FWO: Projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); Ph.D./Postdoc grants; iMinds Medical Information Technologies SBO 2014; IWT: POM II SBO 100031; Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012–2017). The second author received funding from CONICYT Chile through FONDECYT Project 1130122 and DGIP-UTFSM 24.14.84. The first author thanks the colleagues from the Department of Computer Science and Engineering, University of Bologna, for their hospitality during the period in which this research was conceived.

Appendix: Proof of Proposition 2

Let f be a convex differentiable real function on $\mathbb{R}^p$. Given $\mathcal{S} \subseteq \{1, \ldots, p\}$, we define the *restricted gradient* of f with respect to $\mathcal{S}$ as its scaled projection i.e.

$$\tilde{\nabla}_{\mathcal{S}} f(\cdot) = \frac{p}{\kappa}\Big(\sum_{i\in\mathcal{S}} \mathbf{e}_i \mathbf{e}_i^T\Big)\nabla f(\cdot), \tag{14}$$

where $\kappa = |\mathcal{S}|$. We define the *curvature* constant of f over a compact set Σ as

$$C_f = \sup_{x\in\Sigma, y\in\Sigma_x} \frac{1}{\lambda^2}\Big(f(y) - f(x) - (y-x)^T\nabla f(x)\Big)\ , \tag{15}$$

where $\Sigma_x = \{y \in \Sigma : y = x + \lambda(s-x), s \in \Sigma, \lambda \in (0,1]\}$.

For any $\alpha \in \Sigma$ we define its primal gap and duality gap as

$$h(\alpha) = f(\alpha) - f(\alpha^*) \tag{16}$$

$$g(\alpha) = \max_{u\in\Sigma} (\alpha - u)^T \nabla f(\alpha)\ , \tag{17}$$

respectively. Convexity of the function f implies that $f(\alpha) + (u-\alpha)^T\nabla f(\alpha)$ is nowhere greater than $f(\alpha)$. Therefore,

$$g(\alpha) \geq h(\alpha)\ \forall\alpha \in \Sigma\ . \tag{18}$$

For any iterate $\alpha^{(k)}$ generated by our algorithm, we define its *expected primal gap and duality gap* as

$$h_{k+1} = \mathbb{E}_{\mathcal{S}^{(k)}}\big[h(\alpha^{(k+1)})\big] \tag{19}$$

$$g_{k+1} = \mathbb{E}_{\mathcal{S}^{(k)}}\big[g(\alpha^{(k+1)})\big]\ , \tag{20}$$

respectively. Here we denote by $\mathcal{S}^{(k)}$ the random subset of $\{1, \ldots, p\}$ used to approximate the gradient at each iteration. Clearly,

$$g_k \geq h_k \ \forall k. \tag{21}$$

Lemma 2 *Let* $\alpha_\lambda^{(k+1)} = \alpha^{(k)} + \lambda(u^{(k)} - \alpha^{(k)})$ *be a step in the direction of*

$$u^{(k)} \in \underset{u \in \Sigma}{argmin} \left(u - \alpha^{(k)}\right)^T \tilde{\nabla}_{\mathcal{S}^{(k)}} f\left(\alpha^{(k)}\right), \tag{22}$$

with step-size $\lambda \in (0, 1]$. *Then*

$$h_{k+1}(\lambda) = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha_\lambda^{(k+1)})\right] \leq h_k - \lambda g_k + \lambda^2 C_f. \tag{23}$$

Proof Since $\alpha^{(k)}, u^{(k)} \in \Sigma$ and $\alpha_\lambda^{(k+1)} \in \Sigma_{\alpha^{(k)}}$, it follows from (15) that

$$f\left(\alpha_\lambda^{(k+1)}\right) \leq f\left(\alpha^{(k)}\right) + \lambda\left(u^{(k)} - \alpha^{(k)}\right)^T \nabla f\left(\alpha^{(k)}\right) + \lambda^2 C_f\ .$$

After some algebraic manipulations, we have

$$\mathbb{E}_{\mathcal{S}^{(k)}}\left[f\left(\alpha^{(k+1)}\right)\right] \leq f\left(\alpha^{(k)}\right) + \lambda\,\mathbb{E}_{\mathcal{S}^{(k)}}\left[\left(u^{(k)} - \alpha^{(k)}\right)^T \tilde{\nabla}_{\mathcal{S}^{(k)}} f\left(\alpha^{(k)}\right)\right] + \lambda^2 C_f. \tag{24}$$

Since $\mathbb{E}_{\mathcal{S}^{(k)}}[\tilde{\nabla}_{\mathcal{S}}^{(k)} f(\alpha^{(k)})] = \nabla f(\alpha^{(k)})$, by the definition of $u^{(k)}$ and by the order preserving and linearity properties of expectation, we obtain

$$\begin{aligned}
\mathbb{E}_{\mathcal{S}^{(k)}}\left[\left(u^{(k)} - \alpha^{(k)}\right)^T \tilde{\nabla}_{\mathcal{S}^{(k)}} f\left(\alpha^{(k)}\right)\right] &= \mathbb{E}_{\mathcal{S}^{(k)}}\left[\min_{u \in \Sigma}\left(u - \alpha^{(k)}\right)^T \tilde{\nabla}_{\mathcal{S}^{(k)}} f\left(\alpha^{(k)}\right)\right] \\
&\leq \min_{u \in \Sigma} \mathbb{E}_{\mathcal{S}^{(k)}}\left[\left(u - \alpha^{(k)}\right)^T \tilde{\nabla}_{\mathcal{S}^{(k)}} f\left(\alpha^{(k)}\right)\right] \\
&= \min_{u \in \Sigma}\left(u - \alpha^{(k)}\right)^T \nabla f\left(\alpha^{(k)}\right) \\
&= -g\left(\alpha^{(k)}\right).
\end{aligned} \tag{25}$$

Substitution into (24) and expectation with respect to $\mathcal{S}^{(k-1)}$ finally yield

$$\mathbb{E}_{\mathcal{S}^{(k)}}\left[f\left(\alpha_\lambda^{(k+1)}\right)\right] \leq \mathbb{E}_{\mathcal{S}^{(k-1)}}\left[f\left(\alpha^{(k)}\right)\right] - \lambda\mathbb{E}_{\mathcal{S}^{(k-1)}}\left[g\left(\alpha^{(k)}\right)\right] + \lambda^2 C_f.$$

Subtracting $f(\alpha^*)$ from both sides, (19) and (20) yield the result. □

Lemma 3 *The initialization* $\alpha^{(0)} = u^*$ *with* $u^* \in \arg\min_{u \in \mathcal{V}(\Sigma)} f(u)$ *guarantees* $h_k \leq C_f\ k \geq 0$.

Proof First, note that $h_{k+1} \leq h_k \forall k \geq 0$. Indeed,

$$\min_{\lambda \in [0,1]} h\left(\alpha_\lambda^{(k+1)}\right) = \min_{\lambda \in [0,1]} h\left(\alpha^{(k)} + \lambda\left(u^{(k)} - \alpha^{(k)}\right)\right) \leq h\left(\alpha^{(k)}\right).$$

Thus

$$\begin{aligned}
h_{k+1} = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha^{(k+1)})\right] &= \mathbb{E}_{\mathcal{S}^{(k)}}\left[\min_{\lambda \in [0,1]} h\left(\alpha_\lambda^{(k+1)}\right)\right] \\
&\leq \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha^{(k)})\right] = h_k.
\end{aligned}$$

Now, from Lemma 2, any step in the direction of (22) with step size $\lambda \in (0, 1]$ satisfies

$$h_{k+1}(\lambda) = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha_\lambda^{(k+1)})\right] \leq h_k - \lambda g_k + \lambda^2 C_f \leq h_k - \lambda h_k + \lambda^2 C_f \ .$$

Suppose $h_k > C_f$. In this case, as $-\lambda h_k + \lambda^2 C_f < 0$, we can choose $\lambda = 1$ to obtain

$$h_{k+1}(\lambda)|_{\lambda=1} < h_k \ .$$

But

$$h_{k+1}(\lambda)|_{\lambda=1} = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(u^{(k)})\right] \geq \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(u^*)\right] = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha^{(0)})\right] = h_0 \ .$$

Thus, $h_0 < h_k$. This is a contradiction, since $h_{k+1} \leq h_k \forall k \geq 0$. □

Lemma 4 *At each iteration k of Algorithm 2,*

$$h_{k+1} \leq h_k - \frac{h_k^2}{4C_f}. \tag{26}$$

Proof At iteration k, Algorithm 2 updates $\alpha^{(k)}$ by a line search in the direction of (22). Hence

$$h_{k+1} = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha^{(k+1)})\right] = \mathbb{E}_{\mathcal{S}^{(k)}}\left[\min_{\lambda \in (0,1]} h\left(\alpha_\lambda^{(k+1)}\right)\right]. \tag{27}$$

By the order preserving and linearity properties of expectation

$$\mathbb{E}_{\mathcal{S}^{(k)}}\left[\min_{\lambda \in (0,1]} h(\alpha_\lambda^{(k+1)})\right] \leq \min_{\lambda \in (0,1]} \mathbb{E}_{\mathcal{S}^{(k)}}\left[h\left(\alpha_\lambda^{(k+1)}\right)\right]. \tag{28}$$

From lemma 2, we have that any step in the direction of (22) with step size λ satisfies

$$h_{k+1}(\lambda) = \mathbb{E}_{\mathcal{S}^{(k)}}\left[h(\alpha_\lambda^{(k+1)})\right] \leq h_k - \lambda g_k + \lambda^2 C_f \leq h_k - \lambda h_k + \lambda^2 C_f. \tag{29}$$

Combining (29) and (28) produces

$$h_{k+1} = \min_{\lambda \in (0,1]} h_{k+1}(\lambda) \leq \min_{\lambda \in (0,1]} \left(h_k - \lambda h_k + \lambda^2 C_f\right). \tag{30}$$

From lemma 3, $h_k < 2C_f$. Thus, the minimum at the right hand side is obtained for $\bar{\lambda} = h_k/2C_f$ (take derivative, equal to 0, solve and check that $\bar{\lambda} < 1$). Substituting this value of λ yields

$$h_{k+1} \leq h_k - \frac{h_k^2}{2C_f} + \frac{h_k^2}{4C_f} = h_k - \frac{h_k^2}{4C_f}. \tag{31}$$

□

Proof of Proposition 2 With the above results in hand, we can now prove the convergence result in the main paper i.e.

$$h_k = \mathbb{E}_{\mathcal{S}^{(k)}}\left[f\left(\alpha^{(k+1)}\right)\right] - f(\alpha^*) \leq \frac{4C_f}{k+2}.$$

Proof We prove the claim by induction on k. The base case $k = 1$ is trivial to verify from lemma 3 (as $4/3 > 1$). Now, from Lemma 4 and the inductive hypothesis $h_k \leq \frac{4C_f}{k+2}$, we obtain

$$h_{k+1} \leq h_k - \frac{h_k^2}{4C_f} \leq \frac{h_k}{1 + \frac{h_k}{4C_f}} = \frac{1}{\frac{1}{h_k} + \frac{1}{4C_f}} \leq \frac{1}{\frac{k+2}{4C_f} + \frac{1}{4C_f}} = \frac{4C_f}{(k+1)+2}.$$

which completes the inductive step and yields the claimed bound. □

References

Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. https://www.csie.ntu.edu.tw/~cjlin/libsvm.

Clarkson, K. (2010). Coresets, sparse greedy approximation, and the Frank–Wolfe algorithm. *ACM Transactions on Algorithms*, *6*(4), 63:1–63:30.

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, *32*(2), 407–499.

Frandi, E., Ñanculef, R., & Suykens, J. A. K. (2014). Complexity issues and randomization strategies in Frank–Wolfe algorithms for machine learning. In *7th NIPS workshop on optimization for machine learning*.

Frandi, E., Ñanculef, R., & Suykens, J. A. K. (2015). A PARTAN-accelerated Frank–Wolfe algorithm for large scale SVM classification. In *Proceedings of the international joint conference on neural networks 2015*.

Frank, M., & Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, *1*, 95–110.

Friedman, J. (2012). Fast sparse regression and classification. *International Journal of Forecasting*, *28*, 722–738.

Friedman, J., Hastie, T., Höfling, H., & Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, *1*(2), 302–332.

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22.

Garber, D., & Hazan, E. (2015). Faster rates for the Frank–Wolfe method over strongly-convex sets. In *Proceedings of the 32nd ICML*.

Harchaoui, Z., Juditski, A., & Nemirovski, A. (2014). Conditional gradient algorithms for norm-regularized smooth convex optimization. *Mathematical Programming Series A*, *13*(1), 1–38.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning*. New York: Springer New York Inc.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Huang, L., Jia, J., Yu, B., Chun, B. G., Maniatis, P., & Naik, M. (2010). Predicting execution time of computer programs using sparse polynomial regression. In *Advances in neural information processing systems* (pp. 883–891).

Jaggi, M. (2013). Revisiting Frank–Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th international conference on machine learning*.

Jaggi, M. (2014). An equivalence between the Lasso and support vector machines. In J. A. K. Suykens, M. Signoretto, & A. Argyriou (Eds.), *Regularization, optimization, kernels, and support vector machines, chap 1* (pp. 1–26). Boca Raton: Chapman & Hall/CRC.

Kim, S. J., Koh, K., Lustig, M., Boyd, S., & Gorinevsky, D. (2007). An interior-point method for large-scale l 1-regularized least squares. *IEEE Journal of Selected Topics in Signal Processing*, *1*(4), 606–617.

Kogan, S., Levin, D., Routledge, B. R., Sagi, J. S., & Smith, N. A. (2009). Predicting risk from financial reports with regression. In *Proceedings of the NAACL '09* (pp 272–280).

Lacoste-Julien, S., & Jaggi, M. (2014). An affine invariant linear convergence analysis for Frank–Wolfe algorithms. arXiv:1312.7864v2.

Lacoste-Julien, S., Jaggi, M., Schmidt, M., & Pletscher, P. (2013). Block-coordinate Frank–Wolfe optimization for structural SVMs. In *Proceedings of the 30th international conference on machine learning*.

Lan, G. (2014). The complexity of large-scale convex programming under a linear optimization oracle. arXiv:1309.5550v2.

Langford, J., Li, L., & Zhang, T. (2009). Sparse online learning via truncated gradient. In *Advances in neural information processing systems* (pp. 905–912).

Lee, M., Shen, H., Huang, J. Z., & Marron, J. S. (2010). Biclustering via sparse singular value decomposition. *Biometrics*, *66*(4), 1087–1095.

Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml.

Liu, J., & Ye, J. (2009). Efficient euclidean projections in linear time. In *Proceedings of the 26th international conference on machine learning*, (pp. 657–664). New York: ACM.

Liu, J., & Ye, J. (2010). Efficient ℓ_1/ℓ_q norm regularization. arXiv:1009.4766.

Liu, J., Ji, S., & Ye, J. (2009). SLEP: Sparse learning with efficient projections. http://www.yelab.net/software/SLEP/. Arizona State University.

Ñanculef, R., Frandi, E., Sartori, C., & Allende, H. (2014). A novel Frank–Wolfe algorithm: Analysis and applications to large-scale SVM training. *Information Sciences*, *285*, 66–99.

Nesterov, Y. (2013). Gradient methods for minimizing composite functions. *Mathematical Programming Series B*, *140*(1), 125–161.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Richtárik, P., & Takáĉ, M. (2014). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming Series A*, *144*(1), 1–38.

Schölkopf, B., & Smola, A. (2001). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge: MIT Press.

Shalev-Shwartz, S., & Tewari, A. (2011). Stochastic methods for ℓ_1-regularized loss minimization. *Journal of Machine Learning Research*, *12*, 1865–1892.

Shalev-Shwartz, S., Srebro, N., & Zhang, T. (2010). Trading accuracy for sparsity in optimization problems with sparsity constraints. *SIAM Journal on Optimization*, *20*(6), 2807–2832.

Signoretto, M., Frandi, E., Karevan, Z., & Suykens, J. A. K. (2014). High level high performance computing for multitask learning of time-varying models. In *IEEE CIBD 2014*.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B*, *58*(1), 267–288.

Tibshirani, R. (2011). Regression shrinkage and selection via the lasso: A retrospective. *Journal of the Royal Statistical Society Series B*, *73*(3), 273–282.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., & Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society Series B*, *67*(1), 91–108.

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

Turlach, B.A. (2005). On algorithms for solving least squares problems under an l_1 penalty or an l_1 constraint. In *Proceedings of the American Statistical Association, Statistical Computing Section* (pp. 2572–2577).

Wang, Y., & Qian, X. (2014). Stochastic coordinate descent Frank–Wolfe algorithm for large-scale biological network alignment. In *GlobalSIP14—Workshop on genomic signal processing and statistics*.

Weston, J., Elisseeff, A., Schölkopf, B., & Tipping, M. (2003). Use of the zero norm with linear models and kernel methods. *Journal of Machine Learning Research*, *3*, 1439–1461.

Zhou, Q., Song, S., Huang, G., & Wu, C. (2015). Efficient Lasso training from a geometrical perspective. *Neurocomputing*, *168*, 234–239.

Zou, H. (2006). The adaptive Lasso and its oracle properties. *Journal of the American Statistical Association*, *101*(476), 1418–1429.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B*, *67*, 301–320.

Zou, H., & Zhang, H. H. (2009). On the adaptive elastic-net with a diverging number of parameters. *Annals of Statistics*, *37*(4), 1733.