

# Greedy methods, randomization approaches and multi-arm bandit algorithms for efficient sparsity-constrained optimization

A. Rakotomamonjy *Member, IEEE*, S. Koço, L. Ralaivola

**Abstract**—Several sparsity-constrained algorithms such as Orthogonal Matching Pursuit or the Frank-Wolfe algorithm with sparsity constraints work by iteratively selecting a novel atom to add to the current non-zero set of variables. This selection step is usually performed by computing the gradient and then by looking for the gradient component with maximal absolute entry. This step can be computationally expensive especially for large-scale and high-dimensional data. In this work, we aim at accelerating these sparsity-constrained optimization algorithms by exploiting the key observation that, for these algorithms to work, one only needs the coordinate of the gradient’s top entry. Hence, we introduce algorithms based on greedy methods and randomization approaches that aim at cheaply estimating the gradient and its top entry. Another of our contribution is to cast the problem of finding the best gradient entry as a best arm identification in a multi-armed bandit problem. Owing to this novel insight, we are able to provide a bandit-based algorithm that directly estimates the top entry in a very efficient way. Theoretical observations stating that the resulting inexact Frank-Wolfe or Orthogonal Matching Pursuit algorithms act, with high probability, similarly to their exact versions are also given. We have carried out several experiments showing that the greedy deterministic and the bandit approaches we propose can achieve an acceleration of an order of magnitude while being as efficient as the exact gradient when used in algorithms such as OMP, Frank-Wolfe or CoSaMP.

**Index Terms**—Sparsity, Orthogonal Matching Pursuit, Frank-Wolfe algorithm, Greedy methods, Best arm identification.

## I. INTRODUCTION

Over the last decade, there has been a large interest in inference problems featuring data of very high-dimension and a small number of observations. Such problems occur in a wide variety of application domains ranging from computational biology and text mining to information retrieval and finance. In order to learn from these datasets, statistical models are frequently designed so as to feature some sparsity properties. Hence, fueled by the large interest induced by these application domains, an important amount of research work in the machine learning, statistics and signal processing communities, has been devoted to sparse learning and as such, many algorithms

have been developed for yielding models that use only few dimensions of the data. To obtain these models, one typically needs to solve a problem of the form

$$\min_{\mathbf{w}} L(\mathbf{w}) \text{ subject to } \|\mathbf{w}\|_0 \leq K \quad (1)$$

where  $L(\mathbf{w})$  is a smooth convex objective function that measures a goodness of fit of the model,  $\|\mathbf{w}\|_0$  is the  $\ell_0$  pseudo-norm that counts the number of non-zero components of the vector  $\mathbf{w}$  and  $K$  is a parameter that controls the sparsity level.

One usual approach that has been widely considered is the use of a convex and continuous surrogate of the  $\ell_0$  pseudo-norm, namely an  $\ell_1$ -norm. The resulting problem is the well-known Lasso problem [1] and a large variety of algorithms for its resolution exist, ranging from homotopy methods [2] to the Frank-Wolfe (FW) algorithm [3]. In the same flavour, non-convex continuous penalties are also common solutions for relaxing the  $\ell_0$  pseudo-norm [4], [5]. Another possible approach is to consider greedy methods that provide local optimal solution to the problem (1). In this last context, a flurry of algorithms have been proposed, the most popular ones being the *Matching Pursuit* (MP) and the *Orthogonal Matching Pursuit* (OMP) algorithms [6], [7], [8]. One common point of the aforementioned algorithms for solving problem (1) is that they require, at each iteration, the computation of the objective function’s gradient. For large-scale and high-dimensional setting, computing the gradient at each iteration may be very time-consuming.

Stochastic gradient descent (SGD) algorithms are now classical methods for avoiding the computation of the full gradient in large-scale learning problems [9], [10]. Most of these works have been devoted to smooth composite optimization although some efforts addressing  $\ell_1$ -regularized problems exist [11]. Recently, these SGD algorithms have been further accelerated through the introduction of variance reduction methods for gradient estimation [12]. In the context of problem (1) where a non-smooth non-convex constraint appears, very few works have envisaged the use of stochastic optimization. Nguyen et al. [13] have proposed stochastic versions of a gradient pursuit algorithm. Following a similar direction, by exploiting inexact gradient information, we address the problem of accelerating sparsity-inducing algorithms such as Matching Pursuit, Orthogonal Matching Pursuit and the Frank-Wolfe algorithm with an  $\ell_1$  ball constraint. However, unlike stochastic gradient approaches, the acceleration we propose leverages on the fact that at each iteration, these algorithms seek for the gradient’s component that has the largest (absolute) value.

AR is with University of Rouen, LITIS Lab. Most of this work has been carried while he was visiting LIF at Aix-Marseille University. Email alain.rakoto@insa-rouen.fr

SK is with University of Rouen, LITIS LAB. Email: sokol.koco@gmail.com

LR is with Aix-Marseille University, LIF. Email: liva.ralaivola@lif.univ-mrs.fr

This work is supported by Agence Nationale de la Recherche, project GRETA 12-BS02-004-01.

Manuscript received August2015; revised XXX.

Hence, our main contribution in this paper is to propose novel algorithms that allow to efficiently find this top entry of the gradient. By doing so, our objective is to design novel efficient versions of MP, OMP and FW algorithms while keeping intact all the properties of these algorithms for sparse approximation. Indeed, this becomes possible if at each iteration of MP, OMP or FW, our inexact gradient-based estimation of the component with largest value is the same as the one obtained with the exact gradient.

We propose two approaches, the first one based on a greedy deterministic algorithm and the second one based on a randomized method, whose aim is to build an inexact estimation of the gradient so that its top entry is the same as the exact one. Next, by casting the problem as a best arm identification multi-armed bandit problem [14], we are able to derive an algorithm that directly estimates the best component of the gradient. Interestingly, these algorithms are supported by theoretical evidences that with high probability, they are able to retrieve the correct component of the gradient. As a consequence, we show that MP, OMP and FW algorithms employing these approaches for spotting the correct component of the gradient behave as their exact counterpart with high probability.

This paper is an extended version of the conference paper [15]. It provides full details on the context of the problem and proposes a novel key contribution based on multi-arm bandits. In addition, it gives enlightening insights compared to related works. Extended experimental analysis also strengthens the results compared to the conference version. The remainder of the paper is organized as follows. Section II presents sparsity-constrained algorithms, formalizes our problem and introduces the key observation on which we build our acceleration strategy. Section III provides the different algorithms we propose for efficiently estimating the extreme gradient component of interest. A discussion with related works is given in Section IV. Experimental results are depicted in Section V while Section VI concludes the work and presents different outlooks.

## II. SPARSE LEARNING ALGORITHM WITH EXTREME GRADIENT COMPONENT

In this section, we introduce the sparse learning problem we are interested in, as well as some algorithms that are frequently used for sparse learning. We then point out the prominent common trait of those algorithms, namely *the extreme gradient component* property, and discuss how its estimation can be employed for accelerating some classes of sparse learning algorithms.

### A. Framework

Consider the problem where we want to estimate a relation between a set of  $n$  samples gathered in a vector  $\mathbf{y} \in \mathbb{R}^n$  and the matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$ . In a sparse signal approximation problem,  $\mathbf{X}$  would be a matrix whose columns are the elements of a dictionary and  $\mathbf{y}$  the target signal, while in a machine learning problem, the  $i$ -th row of matrix  $\mathbf{X}$ , formalized as  $\mathbf{x}_i^\top$ ,  $\mathbf{x}_i \in \mathbb{R}^d$ , depicts the features of the  $i$ -th example and  $y_i$  is the label or target associated to that example. In the sequel, we denote as  $x_{i,j}$  the entry of  $\mathbf{X}$  at the  $i$ -th row and  $j$ -th column.

---

### Algorithm 1 Gradient Pursuit Algorithm

---

- 1: set  $k = 0$ , initialize  $\mathbf{w}_0 = \mathbf{0}$
  - 2: **for**  $k=0, 1, \dots$  **do**
  - 3:    $i^* = \arg \max_i |\nabla L(\mathbf{w}_k)|_i$
  - 4:    $\Gamma_k = \Gamma_k \cup \{i^*\}$
  - 5:    $\mathbf{w}_{k+1} = \arg \min L(\mathbf{w})$  over  $\Gamma_k$  with  $\mathbf{w}_{\Gamma_k^c} = \mathbf{0}$
  - 6: **end for**
- 

Our objective is to learn the relation between  $\mathbf{y}$  and  $\mathbf{X}$  through a linear model of the data denoted  $\mathbf{X}\mathbf{w}$  by looking for the vector  $\mathbf{w}$  that solves problem (1) when the objective function is of the form

$$L(\mathbf{w}) = \sum_i^n \ell(y_i, g(\mathbf{w}^\top \mathbf{x}_i)),$$

where  $\ell$  is an individual loss function that measures the discrepancy between a true value  $y$  and its estimation, and  $g(\cdot)$  is a given differentiable function that depicts the (potential) non-linear dependence of the loss to  $\mathbf{w}$ . Typically,  $\ell$  might be the least-square error function, which leads to  $L(\mathbf{w}) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$  or the logistic loss and we have  $L(\mathbf{w}) = \sum_{i=1}^n \log(1 + \exp\{-y_i \mathbf{x}_i^\top \mathbf{w}\})$ . In the sequel, we present two algorithms that solve problem (1) by a greedy method and by a continuous and a convex relaxation of the  $\ell_0$  pseudo-norm, respectively.

### B. Algorithms

1) *Gradient Pursuit*: This algorithm is a generalization of the greedy algorithm known as *Orthogonal Matching Pursuit* (OMP) to generic loss functions [16]. It can be directly applied to our problem given in Equation (1). Similarly to OMP, the gradient pursuit algorithm is a greedy iterative procedure, which, at each iteration, selects the largest absolute coordinate of the loss gradient. This coordinate is added to the set of already selected elements and the new iterate is obtained as the solution of the original optimization problem restricted to these selected elements while keeping the other ones at 0. The detailed procedure is given in Algorithm 1.

While conceptually simple, this algorithm comes with theoretical guarantees on its ability to recover the exact underlying sparsity pattern of the model, for different types of loss functions [7], [17], [18].

Several variations of this algorithm have been proposed ranging from methods exploring new pursuits directions instead of the gradient [16], to methods making only slight changes to the original one that have strongly impacted the ability of the algorithm to recover signals. For instance, the *CoSaMP* [19] and *GraSP* [20] algorithms select the top  $2K$  entries in the absolute gradient,  $K$  being the desired sparsity pattern, optimize over these entries and the already selected  $K$  ones, and prune the resulting estimate to be  $K$ -sparse. In addition to being efficient, these algorithms output sparse vectors whose distances from the true sparse optimum are bounded.

---

**Algorithm 2** Frank-Wolfe Algorithm
 

---

```

1: set  $k = 0$ , initialize  $\mathbf{w}_0 = \mathbf{0}$ 
2: for  $k=0,1, \dots$  do
3:    $\mathbf{s}_k = \arg \min_{\mathbf{s} \in C} \mathbf{s}^\top \nabla L(\mathbf{w}_k)$ 
4:    $\mathbf{d}_k = \mathbf{s}_k - \mathbf{w}_k$ 
5:   set, linesearch or optimize  $\gamma_k \in [0; 1]$ 
6:    $\mathbf{w}_{k+1} = \mathbf{w}_k + \gamma_k \mathbf{d}_k$ 
7: end for
  
```

---

2) *Frank-Wolfe algorithm*: The Frank-Wolfe (FW) algorithm aims at solving the following problem

$$\min_{\mathbf{w} \in C} L(\mathbf{w}) \quad (2)$$

with  $\mathbf{w} \in \mathbb{R}^d$ ,  $L$  a convex and differentiable function with Lipschitz gradient and  $C$  a compact subset of  $\mathbb{R}^d$ . The FW algorithm, given in Algorithm 2, is a straightforward procedure that solves problem 2 by first iteratively looking for a search direction and then updating the current iterate. The search direction  $\mathbf{s}_k$  is obtained from the following convex optimization problem (line 3 Alg 2)

$$\mathbf{s}_k = \arg \min_{\mathbf{s} \in C} \mathbf{s}^\top \nabla L(\mathbf{w}_k), \quad (3)$$

which may be efficiently solved, depending on the constraint set  $C$ . For achieving sparsity, we typically choose  $C$  as a  $\ell_1$ -norm ball, *e.g.*  $C = \{\mathbf{w} : \|\mathbf{w}\|_1 \leq 1\}$ , which turns (3) into a linear program. Despite its simplistic nature, the FW algorithm has been shown to be linearly convergent [21], [3]. Interestingly, it can also be shown that convergence is preserved as long as the following condition holds

$$\mathbf{s}_k^\top \nabla L(\mathbf{w}_k) \leq \min_{\mathbf{s} \in C} \mathbf{s}^\top \nabla L(\mathbf{w}_k) + \epsilon, \quad (4)$$

where  $\epsilon$  depends on the smoothness of  $L$  and the step-size  $\gamma_k$ . In general cases where the minimization problem (3) is expensive to solve, this condition suggests that an approximate solution  $\mathbf{s}_k$  may be sufficient, provided that the true gradient is available. Similarly, if an inexact gradient information  $\hat{\nabla}L(\mathbf{w}_k)$  is available, convergence is still guaranteed under the condition that  $\mathbf{s}_k^\top \hat{\nabla}L(\mathbf{w}_k) \leq \min_{\mathbf{s} \in C} \mathbf{s}^\top \nabla L(\mathbf{w}_k) + \epsilon$ , and some conditions relating  $\hat{\nabla}L(\mathbf{w}_k)$  and  $\nabla L(\mathbf{w}_k)$ . For instance, if  $C$  is a unit-norm ball associated with some norm  $\|\cdot\|$  and  $\mathbf{s}_k$  a minimizer of  $\min_{\mathbf{s} \in C} \mathbf{s}^\top \hat{\nabla}L(\mathbf{w}_k)$ , *i.e.*  $\mathbf{s}_k = \arg \min_{\mathbf{s} \in C} \mathbf{s}^\top \hat{\nabla}L(\mathbf{w}_k)$ , then in order to ensure convergence, it is sufficient to have  $\mathbf{s}_k$  so that [3]

$$\|\hat{\nabla}L(\mathbf{w}_k) - \nabla L(\mathbf{w}_k)\|_* \leq \epsilon, \quad (5)$$

where  $\|\cdot\|_*$  is the conjugate norm associated with  $\|\cdot\|$ .

Interestingly, the above equation provides a guarantee on the convergence on an inexact gradient Frank-Wolfe algorithm, but unfortunately, the precision needed on the inexact gradient is given with respect to the exact one. However, while condition (5) is impractical, it conveys the important idea that inexact gradient computation may be sufficient for solving sparsity-constrained optimization problems.

### C. Leveraging from the extreme gradient component estimation

As stated above, the gradient pursuit algorithm needs to solve at each iteration the following problem  $j^* = \arg \max_j |\nabla L(\mathbf{w}_k)|_j$ , *i.e.*, the goal is to find at each iteration the gradient's component with the largest absolute value.

In the Frank-Wolfe algorithm, similar situations where finding  $\mathbf{s}_k$  corresponds to looking for an extreme component of the (absolute) gradient, occur. For instance, when the constraint set is the  $\ell_1$ -norm ball  $C_1 = \{\mathbf{w} \in \mathbb{R}^d : \|\mathbf{w}\|_1 \leq 1\}$  or the positive simplex constraint  $C_2 = \{\mathbf{w} \in \mathbb{R}^d : \|\mathbf{w}\|_1 = 1, w_j \geq 0\}$  and we denote

$$\mathbf{s}^* = \arg \min_{\mathbf{s} \in C_1} \mathbf{s}^\top \nabla L(\mathbf{w}_k) \text{ and } j^* = \arg \max_j |\nabla L(\mathbf{w}_k)|_j,$$

or

$$\mathbf{s}^* = \arg \min_{\mathbf{s} \in C_2} \mathbf{s}^\top \nabla L(\mathbf{w}_k) \text{ and } j^* = \arg \min_j \nabla L(\mathbf{w}_k)|_j,$$

then  $\mathbf{s}^* = \mathbf{e}_{j^*}$ , with  $\mathbf{e}_{j^*}$  the  $j^*$ -th canonical vector of  $\mathbb{R}^d$ . Hence, as in Matching Pursuit, OMP or the gradient pursuit algorithm, for specific choices of  $C$ , we are not interested in the full gradient, nor in its extreme values, but only on the coordinate of the gradient component with the smallest, the largest (absolute) value.

Our objective in this paper is to leverage on these observations for accelerating sparsity-inducing algorithms which look for one extreme component of the gradient. In particular, we are interested in situations where the gradient is expensive to compute and we aim at providing computation-efficient algorithms that produce either approximated gradients whose extreme component of interest is the same as the exact gradient's one, or identify the extreme component without computing the whole (exact) gradient.

### III. LOOKING FOR THE EXTREME GRADIENT COMPONENT

This section formalizes the problem of identifying the extreme gradient component and provides different algorithms for its resolution. We first introduce a greedy approach, then a randomized one and finally exhibit how this problem can be cast as best arm identification in a multi-armed bandit framework.

#### A. The problem

As mentioned in Section II, we are interested in learning problems where the objective function is of the form

$$\sum_i \ell(y_i, g(\mathbf{w}^\top \mathbf{x}_i)).$$

The gradient of our objective function is thus given by

$$\nabla L(\mathbf{w}) = \sum_i \ell'(y_i, g(\mathbf{w}^\top \mathbf{x}_i)) g'(\mathbf{w}^\top \mathbf{x}_i) \mathbf{x}_i = \mathbf{X}^\top \mathbf{r} \quad (6)$$

where  $\mathbf{r} \in \mathbb{R}^n$  is the vector whose  $i$ -th component is  $\ell'(y_i, g(\mathbf{w}^\top \mathbf{x}_i)) g'(\mathbf{w}^\top \mathbf{x}_i)$ . This particular form entails that the gradient may be computed iteratively. Indeed, the sum  $\nabla L(\mathbf{w}) = \sum_{i=1}^n \mathbf{x}_i r_i$  is invariant to the order according to which index  $i$  describes the set  $[1, \dots, n]$ . This makes possible

**Algorithm 3** Greedy deterministic algorithm to compute  $\hat{\nabla}L$ **Input:**  $\mathbf{r}, \{\|\mathbf{x}_i\|\}, \mathbf{X}$ 

- 1: [values, indices]=sort  $\{|r_i|\|\mathbf{x}_i\|\}_i$  in decreasing order
- 2:  $\hat{\nabla}L_t = \mathbf{0}, t = 0$
- 3: **repeat**
- 4:  $i = \text{indices}(t + 1)$
- 5:  $\hat{\nabla}L_{t+1} = \hat{\nabla}L_t + \mathbf{x}_i r_i$
- 6:  $t = t + 1$
- 7: **until** stopping criterion over  $\hat{\nabla}L_{t+1}$  is met

**Output:**  $\hat{\nabla}L_{t+1}$ 

the computation, at each iteration  $t$ , of an approximate gradient  $\hat{\nabla}L_t$ . Denote  $\mathcal{I}_t$  the first  $t$  indices used for computing the sum and  $i_{t+1}$  the  $(t + 1)$ -th index, then, at iteration  $t$ ,  $\hat{\nabla}L_t = \sum_{i \in \mathcal{I}_t} \mathbf{x}_i r_i$  and

$$\hat{\nabla}L_{t+1} = \hat{\nabla}L_t + \mathbf{x}_{i_{t+1}} r_{i_{t+1}}. \quad (7)$$

According to this framework, our objective is then to find an efficient way for computing an approximate gradient  $\hat{\nabla}L_t$  for which the desired extreme entry is equal to the one of the exact gradient. Note that the computational cost of the exact gradient  $\mathbf{X}^\top \mathbf{r}$  is about  $O(nd)$  and a naive computation of the residual  $\mathbf{r}$  has the same cost, which leads to a global complexity of  $O(2nd)$ . However, because  $\mathbf{w}$  is typically a sparse vector in Gradient Pursuit or in the Frank-Wolfe algorithm, we compute the residual vector  $\mathbf{r}$  as  $\mathbf{r} = \mathbf{y} - \mathbf{X}_\Omega \mathbf{w}_\Omega$  where  $\Omega$  are the indices of the non-zero elements of the  $k$ -sparse vector  $\mathbf{w}$ . Hence, computing  $\mathbf{r}$  has only a cost of  $O(nk)$  and it does not require a full pass on all the elements of  $\mathbf{X}$ . The main objective of our contributions is to estimate the extreme entry of  $\mathbf{X}^\top \mathbf{r}$  with algorithms having a complexity lower than  $O(nd)$ . We propose and discuss, in the sequel, three different approaches.

*B. Greedy deterministic approach*

The first approach we propose is a greedy approach which, at iteration  $t$ , looks for the best index  $i$  so that  $r_i \mathbf{x}_i$  optimizes a criterion depending on  $\hat{\nabla}L_t$  and  $\nabla L$ .

Let  $\mathcal{I}_t$  be the set of indices of the examples chosen in the first  $t$  iterations for computing  $\hat{\nabla}L_t$ . At iteration  $t + 1$ , our goal is to find the example  $i^*$  that is the solution of the following problem:

$$i^* = \operatorname{argmin}_{i \in \{1, \dots, n\} \setminus \mathcal{I}_t} \|\nabla L - \hat{\nabla}L_t - \mathbf{x}_i r_i\|, \quad (8)$$

with  $\hat{\nabla}L_{t+1} = \hat{\nabla}L_t + \mathbf{x}_i r_i$ . The solution of this problem is thus the vector product  $r_i \mathbf{x}_i$  that has minimal norm difference compared to the current gradient estimation residual  $\nabla L - \hat{\nabla}L_t$ . While simple, this solution cannot be computed because  $\nabla L$  is not accessible. Hence, we have to resort to an approximation based on the following equivalent problem:

$$\begin{aligned} i^* &= \operatorname{argmax}_{i \in \{1, \dots, n\} \setminus \mathcal{I}_t} -\|\nabla L - \hat{\nabla}L_{t+1}\| \\ &= \operatorname{argmax}_{i \in \{1, \dots, n\} \setminus \mathcal{I}_t} \|\nabla L - \hat{\nabla}L_t\| - \|\nabla L - \hat{\nabla}L_{t+1}\| \end{aligned} \quad (9)$$

where we use the fact that  $\|\nabla L - \hat{\nabla}L_t\|$  does not depend on  $i$ . By upper bounding the above objective value, one can derive

**Algorithm 4** Computing  $\hat{\nabla}L$  as an empirical average**Input:**  $\mathbf{r}, \|\mathbf{x}_i\|, \mathbf{X}$ 

- 1: build  $\mathbf{p} \in \Delta_n^*$  e.g  $p_i \propto \frac{1}{n}$  or  $p_i \propto |r_i| \|\mathbf{x}_i\|$
- 2: **repeat**
- 3: random draw  $i \in 1, \dots, n$  according to  $\mathbf{p}$
- 4:  $\hat{\nabla}L_{t+1} = \hat{\nabla}L_t + \frac{1}{p_i} \mathbf{x}_i r_i$
- 5:  $t = t + 1$
- 6: **until** stopping criterion over  $\hat{\nabla}L_{t+1}$  is met

**Output:**  $\hat{\nabla}L_{t+1}$ 

the best choice of  $r_i \mathbf{x}_i$  that achieves the largest variation of the gradient estimation norm residual. Indeed, we have

$$\|\nabla L - \hat{\nabla}L_t\| - \|\nabla L - \hat{\nabla}L_{t+1}\| \leq \|-\hat{\nabla}L_t + \hat{\nabla}L_{t+1}\|$$

The right-hand side of this equation can be further simplified

$$\begin{aligned} \|\hat{\nabla}L_{t+1} - \hat{\nabla}L_t\| &= \|\hat{\nabla}L_t - \mathbf{x}_i r_i - \hat{\nabla}L_t\| \\ &= \|\mathbf{x}_i r_i\| = \|\mathbf{x}_i\| |r_i|. \end{aligned} \quad (10)$$

Equation (10) suggests that the index  $i$  that should be chosen at iteration  $t + 1$  is the one with the largest absolute residual weighted by its example norm. Simply put, in the first iteration, the algorithm chooses the index that leads to the largest value  $\|\mathbf{x}_i\| |r_i|$ , in the second one, it selects the second best, and so forth. That is, the examples are considered in a decreasing order with respect to the weighted value of their residuals. Pseudo-code of the approach is given in Algorithm 3.

Note that for this method, at iteration  $t = n$  we recover the exact gradient,

$$\hat{\nabla}L_n = \nabla L.$$

Hence, when  $t = n$ , we are assured to retrieve the correct extreme entry of the gradient at the expense of extra computations needed for running this greedy approach compared to a plain computation of the full gradient. On the other hand, if we stop this gradient estimation procedure before  $t = n$ , we save computational efforts at the risk of missing the correct extreme entry.

*C. Matrix-Vector Product as Expectations*

The problem of finding the extreme component of the gradient can also be addressed from the point of view of randomization, as described in Algorithm 4.

The approach consists in considering the computation of  $\mathbf{X}^\top \mathbf{r}$  as an the expectation of a given random variable. Recall that  $\mathbf{X}$  is composed of the vectors  $\{\mathbf{x}_i^\top\}_{i=1}^n$  and  $\mathbf{x}_i \in \mathbb{R}^d$ . Hence, the matrix-vector product  $\mathbf{X}^\top \mathbf{r}$  can be rewritten:

$$\mathbf{X}^\top \mathbf{r} = \sum_{i=1}^n r_i \mathbf{x}_i. \quad (11)$$

From now on, given some integer  $n$ ,  $\Delta_n^*$  denotes the interior of the probabilistic simplex of size  $n$ :

$$\Delta_n^* \doteq \left\{ \mathbf{p} = [p_1 \cdots p_n] : \sum_{i=1}^n p_i = 1, p_i > 0, i = 1, \dots, n \right\}. \quad (12)$$

For any element  $\mathbf{p} = (p_i)_{i \in [n]} \in \Delta_n^*$  we introduce a random vector  $C$  that takes value in the set

$$C \doteq \{\mathbf{c}_i \doteq r_i \mathbf{x}_i / p_i : i = 1, \dots, n\}$$

so that  $\mathbb{P}(C = \mathbf{c}_i) = p_i$ . This way,

$$\mathbb{E}C = \sum_{i=1}^n p_i \mathbf{c}_i = \sum_{i=1}^n p_i r_i \mathbf{x}_i / p_i = \sum_{i=1}^n r_i \mathbf{x}_i = \mathbf{X}^\top \mathbf{r} \quad (13)$$

Hence, if  $C_1, \dots, C_s$  are independent copies of  $C$  and  $\hat{C}^s$  is defined as:  $\hat{C}^s \doteq \frac{1}{s} \sum_{i=1}^s C_i$  then  $\mathbb{E}\hat{C}^s = \mathbf{X}^\top \mathbf{r}$ .  $\hat{C}^s$  is thus an estimator of the matrix-vector product that we are interested in *i.e.* the gradient of our objective function.

According to the above, a relevant approach for estimating the extreme component of the gradient is to randomly sample  $s$  copies of  $C$ , to average them and then to look for the extreme component of this estimated gradient.

Interestingly, this approach based on randomized matrix multiplication can be related to our deterministic approach. Indeed, a result given by [22] (Lemma 4) says that the element  $\mathbf{p} \in \Delta_n^*$  that minimizes  $\mathbb{E}[\|\mathbf{X}^\top \mathbf{r} - \hat{C}^s\|_F^2]$  is such that

$$p_i \propto |r_i| \|\mathbf{x}_i\|. \quad (14)$$

It thus suggests that vectors  $\mathbf{c}_i$  of large values of  $|r_i| \|\mathbf{x}_i\|$  have higher probability to be sampled. This resonates with the greedy deterministic approach in which vectors  $\mathbf{x}_i r_i$  are accumulated in the order of the decreasing values of  $|r_i| \|\mathbf{x}_i\|$ .

#### D. Best arm identification and multi-armed bandits

The two preceding approaches seek at approximating the gradient, at a given level of accuracy that has yet to be defined, and then at evaluating the coordinate of its extreme component.

Yet, the problem of finding the extreme component coordinate of a gradient vector obtained from matrix-vector multiplication can also be posed as the problem of finding the best arm in a multi-armed bandit problem. In a nutshell, given a slot machine with multiple arms, the goal in the bandit problem is to find the arm that maximizes the expected reward or minimizes the loss. For this, an iterative procedure is used, where at each step, a forecaster selects an arm, based on his previous actions, and receives a reward or observes a loss. Depending on how the reward is obtained, the problem can be stochastic (the reward/loss is drawn from a probability distribution) or non-stochastic. Bubeck et al. [14] propose an extensive review of these methods for various settings.

We cast our problem of finding the extreme gradient component as a best arm identification problem as follows. In the remainder of this section, we suppose that we look for a minimum gradient component and thus we look for the arm with minimal loss instead of maximal reward. We consider that the arms are the components of the gradient (we have  $d$  arms) and at each pull of a given arm, we observe a loss that is built from a term of the gradient matrix-vector multiplication  $\mathbf{X}^\top \mathbf{r}$ , as made clear in the sequel.

In a stochastic setting, we consider a similar framework as the one we described in Section III-C. We model the loss

obtained from the  $k$ -th pull of arm  $j \in [1, \dots, d]$  as a random variable  $V$ , independent of  $k$ , that takes value in the set

$$\{v_{j,i} \doteq r_i x_{i,j} / p_i : i = 1, \dots, n\}$$

so that  $\mathbb{P}(V = v_{j,i}) = p_i$ . From this definition, the expected loss of arm  $j$  is

$$\mathbb{E}V = \sum_{i=1}^n p_i v_{j,i} = \sum_{i=1}^n r_i x_{i,j} = (\mathbf{X}^\top \mathbf{r})_j,$$

which is the  $j$ -th component of our gradient vector. In this setting, given a certain number of pulls, one pull providing a realization of the random variable  $V$  of the chosen arm, the objective of a best arm identification algorithm is to provide recommendation about the arm with minimal expected loss, which in our case is the coordinate of smallest value in the  $d$ -dimensional vector  $\mathbf{X}^\top \mathbf{r}$ .

Several algorithms for identifying this best arm have been provided in the literature. Most of them are built around an empirical average loss statistic. This latter can be computed, after  $s$  pulls of an arm  $j$ , as  $\hat{V}_{j,s} = \frac{1}{s} \sum_{t=1}^s v_{j,i(k,t)}$ , where  $i(k,t)$  is the index  $i$  drawn at the  $k$ -th pull for arm  $j$ . Some of the most interesting algorithms are the *successive reject* [23] and *successive halving* [24] algorithms which, given a fixed budget of pulls, iteratively discard after some predefined number of pulls (say  $s$ ) the worst arm or the worst-half arms, respectively, according to the values  $\{\hat{V}_{j,s}\}_{j=1}^d$ . These approaches are relatively simple and the *successive halving* approach is depicted in detail in Algorithm 5. They directly provide some guarantees on the probability to correctly estimate the extreme component of the gradient. For instance, for a fixed budget of pulls  $T$ , under some minor and easily satisfied conditions on  $\{v_{j,i}\}$ , the *successive halving* algorithm correctly identifies the minimum gradient component with probability at least  $1 - 3 \log_2 d \cdot \exp(-\frac{T}{8H_2 \log_2 d})$  where  $H_2$  is a problem-dependent constant (the larger  $H_2$  is, the harder the problem is).

However, these two algorithms have the drawbacks to work on individual entries  $x_{i,j}$  of the matrix  $\mathbf{X}$ . Hence, overload due to single memory accesses compared to those needed for accessing chunks of memory may hinder the computational gain obtained by identifying the component with minimal gradient value without computing the full gradient.

For the *successive halving* algorithm, one way of overcoming this issue is to consider *non i.i.d* sampling of the arm's loss. As such, we consider that at each iteration, the losses generated by pulling the remaining arms come from the same component  $j$  of the residual. This approach has the advantage of working on the full vector  $\mathbf{x}_i r_i$ , allowing thus efficient memory caching, instead of on individual elements of  $\mathbf{X}$ .

Let  $\mathcal{A}_{s'}$  and  $\mathcal{A}$  denote the sets of components  $i$  that have been drawn after  $s'$  and the subsequent  $s^\dagger$  pulls such that  $s = s' + s^\dagger$  respectively, then the loss for arm  $j$  after  $s$  pulls can be defined as

$$\hat{V}_{j,s} = \hat{V}_{j,s'} + \sum_{i \in \mathcal{A}} \frac{r_i x_{i,j}}{p_i}, \quad (15)$$

where  $\hat{V}_{j,0} = 0$  and the set  $\mathcal{A}$  is the same for all arms. In practice, as implemented in the numerical simulations we

---

**Algorithm 5** Successive Halving to find the minimum gradient component
 

---

- 1: **inputs:**  $\mathbf{X}$ ,  $\mathbf{r}$ , set  $T$  the budget of pulls
  - 2: Initialize  $S_0 = [d]$
  - 3:  $\hat{V}_{j,0} = 0, \forall j \in S_0$
  - 4: **for**  $\ell = 0, 1, \dots, \lceil \log_2(d) \rceil - 1$  **do**
  - 5: Pull each arm in  $S_\ell$  for  $r_\ell = \lfloor \frac{T}{|S_\ell| \lceil \log_2(n) \rceil} \rfloor$  additional times and compute resulting loss (*non-iid*)  $\hat{V}_{\cdot, R_\ell}$  in Equation (15) or (*non-stochastic*)  $v_{\cdot, R_\ell}$  in Equation (16) with  $R_\ell = \sum_{u=0}^{\ell} r_u$
  - 6: Sort arms in  $S_\ell$  by increasing value of losses
  - 7: Define  $S_{\ell+1}$  as the  $\lfloor |S_\ell|/2 \rfloor$  indices of arms with smallest values
  - 8: **end for**
  - 9: **output:** index of the best arm
- 

provide, each component  $i$  of  $\mathcal{A}$  is randomly chosen according to a uniform distribution over  $1, \dots, n$ . The *non-iidness* of the stochastic process comes from that the arm losses are dependent through the set  $\mathcal{A}$ . However, this does not hinder the fact that empirical loss  $\hat{V}_{j,s}$  is still a relevant estimation of the expected loss of arm  $j$ .

Non-stochastic best arm identification has been barely studied and only a very recent work has addressed this problem [25]. In this latter work, the main hypothesis about the non-stochasticity of the loss is that they are assumed to be generated before the game starts. This is exactly our situation since before each estimation of the minimum gradient component, all losses are given by  $x_{i,j}r_i$  and thus can be computed beforehand. In this non-stochastic setting [25], the framework is that the  $k$ -th pull of an arm  $j$  provides a loss  $v_{j,k}$  and the objective of the bandit algorithm is to identify  $\arg \min_j \lim_{k \rightarrow \infty} v_{j,k}$ , assuming that such limits exist for all  $j$ . Again we can fit our problem of finding the extreme gradient component (here the minimum) into this framework by defining the loss for a given arm at pull  $k$  as

$$v_{j,k} = \begin{cases} 0 & \text{if } k = 0 \\ v_{j,k-1} + r_{\tau_k} x_{\tau_k, j} & \text{if } 1 \leq k \leq n \\ v_{j,k-1} & \text{if } k > n \end{cases} \quad (16)$$

where  $\tau$  is a predefined or random permutation of the rows of the vector  $\mathbf{r}$  and the matrix  $\mathbf{X}$ , and  $\tau_k$  its  $k$ -th entry. In practice, we choose  $\tau$  to be the same for all the arms for computational reasons as explained above, but in theory this is not necessary [25]. According to this loss definition, we have

$$\lim_{k \rightarrow \infty} v_{j,k} = \sum_{k=1}^n r_{\tau_k} x_{\tau_k, j} = \sum_{k=1}^n r_k x_{k, j} = (\mathbf{X}^\top \mathbf{r})_j.$$

Hence, an algorithm that recommends the best arm after a given number of pulls, will return the index of the minimum component in our gradient. Interestingly, the algorithm proposed by Jamieson et al. [25] for solving the non-stochastic best arm identification problem is also the one used in the stochastic setting namely the *successive halving* algorithm (Alg. 5). This algorithm can be shown to work as is despite the dependence between arm losses. Indeed, each round-robin

pull of the surviving arms can have dependent values, and as long as the algorithm does not adapt to the observed losses during the middle of a round-robin pull [25].

As already mentioned, for a fixed budget  $T$  of pulls, this *successive halving* bandit algorithm comes with theoretical guarantee in its ability to identify the best arm. In the stochastic case, the probability of success depends on the number of arms, the number of pulls  $T$  and on a parameter denoting the hardness of the problem (see Theorem 4.1 in [24]). In the non-stochastic case, the budget of pulls needed for guaranteeing the correct recovery of the best arm essentially depends on a function  $\gamma_j(k)$  such that  $|v_{j,k} - \lim_{k \rightarrow \infty} v_{j,k}| \leq \gamma_j(k)$  and on a parameter denoting the gap between any of  $(\mathbf{X}^\top \mathbf{r})_j$  and the smallest component of  $\mathbf{X}^\top \mathbf{r}$  which is not accessible unless we compute the exact gradient.

One may note the strong resemblance between the matrix-vector product approximation as given in (13) and the *non-iid* bandit strategy, as in this latter setting, we consider the full vector  $\mathbf{x}_i r_i$  to compute  $\hat{V}_{\cdot, s}$  for all remaining arms. This *non-iid* strategy can also be related to the non-stochastic setting if we choose  $\tau$  as a random permutation of  $1, \dots, n$ . In addition, in the non-stochastic bandit setting, we can recover the greedy deterministic approach if we assume that the permutation  $\tau$  defines a re-ordering of  $\|\mathbf{x}_i\| \|r_i\|$  in decreasing order, then the accumulation given in Equation (16) is exactly the one given in the greedy deterministic approach. This is the choice of  $\tau$  we have considered in our experiments. Multi-armed bandit framework and the gradient approximation approaches use thus similar ways for computing the criteria used for estimating the best arm. The main difference resides in the fact that with multi-armed bandit, one is directly provided with the estimation of the best arm.

### E. Stopping criteria

In the greedy deterministic and randomized methods introduced in this section, we have no clues on how many elements  $r_i \mathbf{x}_i$  have to be accumulated in order to achieve a sufficient approximation of the gradient or in the multi-armed bandit approach how many pulls we need to draw. Here, we discuss two possible stopping criteria for the non-bandit algorithms: one that holds for any approach and a second one that holds only for the Frank-Wolfe algorithm in the deterministic sampling case. Discussion on the budgets that needs to be allocated to the bandit problem is also provided.

1) *Stability condition:* For the sake of simplicity, we limit the exposition to the search of the smallest component of the gradient, although the approach can be generalized to other cases.

Denote by  $j^*$  the coordinate such that  $j^* = \arg \min_j \nabla L(\mathbf{w}_k)|_j$  and let  $T_s$  be the maximal number of iterations or samplings allowed for computing the inexact gradient (for instance, in the greedy deterministic approach,  $T_s = n$ ). Our objective is to estimate  $j^*$  with the fewest number  $t$  of iterations. For this to be possible, we make the hypothesis that there exists an iteration  $t_1$ ,  $t_1 \leq T_s$ , and

$$j^* = \arg \min_j \hat{\nabla} L_t(\mathbf{w}_k)|_j \quad \forall t : t_1 \leq t \leq T_s$$

in other words, we suppose that starting from a given number of iterations  $t_1$ , the gradient approximation is sufficiently accurate so that the updates of the gradient will leave the minimum coordinate unchanged. Formally, this condition means that  $\forall t \in [t_1, \dots, n]$ , we have

$$\left[ \hat{\nabla} L_{t+} \sum_{u=0}^{T_s-t} U_{t+u} \right]_{j^*} \leq \left[ \hat{\nabla} L_{t+} \sum_{u=0}^{T_s-t} U_{t+u} \right]_j \quad \forall j \in [1, \dots, d].$$

where each  $U_{t+i} = r_{i(t+u)} \mathbf{x}_{i(t+u)}$ ,  $i(t+u)$  being an index of samples that depends how the greedy or randomized strategy considered. However, checking the above condition is as expensive as computing the full gradient, thus we propose an estimation of  $j^*$  based on an approximation of this inequality, by truncating the sum to few iterations on each side. Basically, this consists in evaluating  $j^*$  at each iteration and checking whether this index has changed over the last  $N_s$  iterations. We refer to this criterion as the *stability criterion*, parametrized by  $N_s$ .

2) *Error bound criterion*: In Section II-B2, we discussed that the convergence of the Frank-Wolfe inexact algorithm can be guaranteed as long as the norm difference between the approximate gradient and the exact one could be upper-bounded by some quantity  $\epsilon$ . Formally, this means that the iterations over gradient approximation can be stopped as soon as  $\|\hat{\nabla} L_t - \nabla L\|_\infty \leq \epsilon$ , where  $\epsilon$  depends on the curvature of the function  $L(\mathbf{w})$  [3]. In practice, the criterion  $\|\hat{\nabla} L_t - \nabla L\|_\infty$  cannot be computed as it depends on the exact gradient but it can be upper-bounded by a term that is accessible. For the greedy deterministic approach, by norm equivalence, we have

$$\|\hat{\nabla} L_t - \nabla L\|_\infty = \left\| \sum_{i \notin \mathcal{I}_t} \mathbf{x}_i r_i \right\|_\infty \leq \sum_{i \notin \mathcal{I}_t} \|\mathbf{x}_i\|_\infty |r_i| \leq \epsilon \quad (17)$$

Hence if the norms  $\{\|\mathbf{x}_i\|_\infty\}$  have been precomputed beforehand, this criterion can be easily evaluated at each gradient update iteration.

3) *Pull budget for the bandit*: In multi-armed bandit algorithms, one typically specifies the number of pulls  $T$  available for estimating the best arm. As such,  $T$  can be considered as hyperparameter of the algorithm. A possible strategy for removing the dependency of the bandit algorithm to this pull budget, is to use the *doubling trick* [25], which consists in running the algorithm with a small value of  $T$  and then repeatedly doubling it until some stopping criterion is met. The algorithm can then be adapted so as to exploit the loss computations that have already been carried from iteration to iteration. However, this strategy needs a stopping criterion for the *doubling trick*. According to Theorem 1 in [25], there exists a lower bound of pulls for which the algorithm is guaranteed to return the best arm. Hence, the following heuristic can be considered: if  $T'$  and  $2T'$  number of pulls return the same best arm, then we conjecture that the proposed arm is indeed the best one. One can note that this idea is similar to the above-described stability criterion. While this strategy is appealing, in the experiment, we have just fixed the budget of pulls  $T$  to a fixed predefined value.

## IV. DISCUSSION

This section provides comments and discussions of the approaches we proposed compared to existing works.

### A. Relation and gains compared to OMP and variants

Several recent works on sparse approximation have proposed theoretically founded algorithms. These works include OMP [7], [26], greedy pursuit [16], [27], CoSaMP [19] and several others like [28]. Most of these algorithms make use of the top absolute entry of the gradient vector at each iteration. The work presented in this paper is strongly related to these ones as we share the same quest for the top entry. Indeed, the proposed methodology provides tools that can be applied to many sparse approximation algorithms including the aforementioned ones. What makes our work novel and compelling is that at each iteration, the gradient is computed with as few information as possible. If the stopping criterion for estimating this gradient is based on a maximal number of samples — *e.g.* we are interested in constructing the best approximation of the gradient from only 20% of the samples —, our approach can be interpreted as a method for computing the gradient on a limited budget. Hence, the proposed method allows to obtain a gain in the computational time needed for the estimation of the gradient. On the downside, if other stopping criteria are used (alone or jointly with the budget criterion), this gain may be partly impaired by further computations needed for their estimation. As an example, the *stability criterion* induces a  $O(d)$  overhead at each iteration due to the max computation.

### B. Relation with other stochastic MP/FW approaches

Some prior works from the literature are related to the approaches we have proposed in the present paper. Chen et al. [29] have recently introduced a stochastic version of a Matching Pursuit algorithm in a Bayesian context. Their principal contribution was to define some prior distribution over each component of the vector  $\mathbf{w}$  and then to sample over this distribution so as to estimate  $\mathbf{w}$ . In their approach, the sparsity pattern related to Matching Pursuit is controlled by the prior distribution which is assumed to be a mixture of two distributions one of which induces sparsity. While this approach is indeed stochastic, it strongly differs from ours in the way stochasticity is in play. As we will discuss in the next subsection, our framework is more related to stochastic gradient than to the stochastic sampling of Chen et al.

Stronger similarities with our work appear in the work of Peel et al. [30]. Indeed, they propose to accelerate the atom selection procedure by randomly selecting a subset of atoms as well as a subset of example for computing  $\mathbf{X}^\top \mathbf{r}$ . This idea is also the base of our work. However, an essential difference appears as we do not select a subset of atoms. By doing so, we are ensured not to discard the top entry of  $\mathbf{X}^\top \mathbf{r}$  and thus we can guarantee for instance that our bandit approaches are able to retrieve this top entry with high probability given enough budget of pulls.

Stochastic variants of the Frank-Wolfe algorithm have been recently proposed by Lacoste-julien et al. [31] and Ouyang

et al. [32]. These works are mostly tailored for solving large-scale SVM optimization problem and do not focus on sparsity.

### C. About stochasticity

The randomization approach for approximating the gradient, introduced in Section III-C, involves random sampling of the columns. In the extreme situation where only a single column  $i$  is sampled, we thus have  $\hat{\nabla}L = \mathbf{x}_i r_i$ , and the method we propose boils down to a stochastic gradient method. In the context of sparse greedy approximation, the first work devoted to stochastic gradient approximation has been recently released [13]. Nguyen et al. [13] show that their stochastic version of the iterative hard thresholding algorithm, or the gradient matching pursuit algorithm which aim at greedily solving a sparse approximation problem with arbitrary loss functions, behave properly in expectation.

The randomized approach we propose in this work goes beyond the stochastic gradient method for greedy approximation, since it also provides a novel approach for computing stochastic gradient. Indeed, we differ from their setting in several important aspects:

- First, in our stochastic gradient approximation, we always consider a number of samples larger than 1. As such, we are essentially using a stochastic mini-batch gradient.
- Second, the size of the mini-batch is variable (depending on the stopping criterion considered) and it depends on some heuristics that estimates on the fly the ability of the approximate gradient to retrieve the top entry of the true gradient.
- Finally, one important component of our approach is the importance sampling used in the stochastic mini-batch sampling. This component theoretically helps in reducing the error of the gradient estimation [33]. In the context of matrix multiplication approximation we used for developing the randomized approach in Section III-C, theoretical results of Drineas et al. [22] have also shown there exists an importance sampling that minimizes the expectation of the Frobenius norm of the matrix multiplication approximation. Our experiments corroborate these results showing that, compared to a uniform sampling, importance sampling clearly enhances the efficiency in retrieving the top entry of the true gradient.

All these differences make our randomized algorithm not only clearly distinguishable from stochastic gradient approaches, but also harder to analyze. We thus defer for further work the theoretical analyses of such a stochastic adaptive-size mini-batch gradient coupled with an importance sampling approach.

### D. Theoretical considerations

Although a complete theoretical analysis of the algorithm is out of the scope of this paper, an interesting property deserves to be mentioned here. Note that, unlike stochastic gradient approaches, our algorithm is built upon inexact gradients that hopefully have the same minimum component as the true gradient. If this latter fact occurs along the iterations of the

FW or OMP algorithms, then all the properties (e.g linear convergence, exact recovery property...) of these algorithms apply. Based on the probability of recovering an exact minimum component of the gradient at each iteration, we show below a bound on the probability of our algorithm to recover at a given iteration  $K$  of OMP or FW, the same sequence of minimum components as the one obtained with exact gradient.

Suppose that at each iteration  $t$  of OMP or FW, our algorithm for estimating the minimum component correctly identifies this component with a probability at least  $1 - P_t$  and there exists  $\bar{P}$  so that  $P_t > \bar{P}$ ,  $\forall t \leq K$ , then the probability of identifying the correct sequence of minimum component is at least  $1 - K\bar{P}$ . We get this thanks to the following reasoning. Denote  $B(t) = \{I^1 = i_e^1, I^2 = i_e^2, \dots, I^t = i_e^t\}$  the event that our algorithm outputs the exact sequence of minimum components up to iteration  $t$ ,  $I^t$  and  $i_e^t$  being the coordinates retrieved with the inexact and exact gradient. Similarly, we note  $A(t) = \{I^t = i_e^t\}$  the event of retrieving, at iteration  $t$ , the correct component of the gradient. We assume that  $\mathbb{P}(B(0)) = 1$ . Formally, we are interested in lower-bounding the probability of  $B(K)$ . By definition, we have

$$\begin{aligned} \mathbb{P}(B(K)) &= \mathbb{P}(A(K)|B(K-1))\mathbb{P}(B(K-1)) \\ &= \mathbb{P}(B(0)) \prod_{t=1}^K \mathbb{P}(A(t)|B(t-1)). \end{aligned}$$

Note that this equation captures all the time dependencies that occur during the FW or the OMP algorithm. Since  $\mathbb{P}(A(t)|B(t-1)) \geq 1 - P_t$ , we have

$$\mathbb{P}(B(K)) \geq \prod_{t=1}^K (1 - P_t) \geq (1 - \bar{P})^K \geq (1 - K\bar{P})$$

where in the last inequality, we used the fact that  $(1 - u)^K \geq (1 - Ku)$  for  $0 \leq u \leq 1$ .

For instance, in the *successive halving* algorithm, we have  $P_t = 3 \log_2 d \cdot \exp\left(-\frac{T}{8H_2(t)\log_2 d}\right)$ , where  $H_2(t)$  is a iteration-dependent constant [24] and  $T$  the number of pulls. Thus, if we define  $\bar{H}$  so that  $\forall t, \bar{H} \geq H_2(t)$ , we have  $\bar{P} = 3 \log_2 d \cdot \exp\left(-\frac{T}{8\bar{H}\log_2 d}\right)$  and

$$\mathbb{P}(B(K)) \geq 1 - 3K \log_2 d \cdot \exp\left(-\frac{T}{8\bar{H}\log_2 d}\right).$$

We can see that the probability of our OMP or FW having the same behaviour as their exact counterpart decreases with the number  $K$  of iterations and the number  $d$  of dimensions of the problems and increases with the number of pulls. By rephrasing this last equation, we also get the following property. For  $\delta \in [0, 1]$ , if the number  $T$  of pulls is set so that at each iteration  $t$ ,

$$T \geq \left(\log \log_2 \frac{d}{\delta} + \log \frac{K}{\delta} + \log \frac{3}{\delta}\right) 8\bar{H} \log_2 d$$

then, when using the *successive halving* algorithm for retrieving the extremum gradient component, an inexact OMP or FW algorithm behaves like the exact OMP or FW with probability  $1 - \delta$ . This last property is another emphasis on the strength of our inexact gradient method compared to stochastic gradient



descent approaches as it shows that with high probability, all the theoretical properties of OMP or FW (*e.g.* convergence, exact recovery of sparse signal) apply.

## V. NUMERICAL EXPERIMENTS

In this section, we describe the experimental studies we have carried out for illustrating the computational benefits of using inexact gradient for sparsity-constraint optimization problems.

### A. Experimental setting

In order to illustrate the benefit of using inexact gradient for sparse learning or sparse approximation, we have set up a simple sparse approximation problem which focuses on the computational gain, and for which a sparse signal has to be recovered by the Frank-Wolfe, OMP or CoSaMP algorithm.

Note that sparse approximations are mostly used for approximation problems on overcomplete dictionary. This is the case in our experiments, where the dimension  $d$  of the learning problem is in most cases larger than the number  $n$  of samples. We believe that if the signal or the image at hand to be approximated can be fairly approximated by representations for which fast transforms are available, then it is better (and faster) to indeed use this representation and the fast transform. Sparse approximation problems as considered in the sequel, mostly occur in overcomplete dictionary learning problems. In such a situation, as the dictionary is data-driven, we believe that the approach we propose is relevant.

The target sparse signals are built as follows. For a given value of the dictionary size  $d$  and a number  $k$  of active elements in the dictionary, the true coefficient vector  $\mathbf{w}^*$  is obtained as follows. The  $k$  non-zero positions are chosen randomly, and their values are drawn from a zero-mean unit variance Gaussian distribution, to which we added  $\pm 0.1$  according to the sign of the values. The columns of the regression design matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$  are drawn uniformly from the surface of a unit hypersphere of dimension  $n$ . Finally, the target vector is obtained as  $\mathbf{y} = \mathbf{X}\mathbf{w}^* + \mathbf{e}$ , where  $\mathbf{e}$  is a random noise vector drawn from a Gaussian distribution with zero-mean and variance  $\sigma_e^2$  determined from a given signal-to-noise as  $\sigma_e^2 = \frac{1}{n} \|\mathbf{X}\mathbf{w}^*\|^2 \cdot 10^{-SNR/10}$ . Unless specified, the SNR ratio has been set to 3. For each setting, the results are averaged over 20 trials, and  $\mathbf{X}$ ,  $\mathbf{w}^*$  and  $\mathbf{e}$  are resampled at each trial.

The criteria used for evaluating and comparing the proposed approaches are the running time of the algorithms and their ability to recover the true non-zero elements of  $\mathbf{w}^*$ . The latter is computed through the F-measure between the support of the true coefficient vector  $\mathbf{w}^*$  and the estimated one  $\hat{\mathbf{w}}$ :

$$\text{F-meas} = 2 \frac{|\text{supp}_\gamma(\mathbf{w}^*) \cup \text{supp}_\gamma(\hat{\mathbf{w}})|}{|\text{supp}_\gamma(\mathbf{w}^*)| + |\text{supp}_\gamma(\hat{\mathbf{w}})|}$$

where,  $\text{supp}_\gamma(\mathbf{w}) = \{j : |w_j| > \gamma\}$  is the support of vector  $\mathbf{w}$  and  $\gamma$  is a threshold used to neglect some non-zero coefficients that may be obliterated by the noise. In all our experiments, we have set  $\gamma = 0.001$  which is small compared to the minimal absolute value of a non-zero coefficient (0.1).

All the algorithms (Frank-Wolfe, OMP and CoSaMP) exact and inexact gradient codes have been written in Matlab except for the *successive reject* bandit which as been written in C and transformed in a mex file. All computations have been run on each single core of an Intel Xeon E5-2630 processor clocked at 2.4 GHz in a Linux machine with 144 Gb of memory.

### B. Sparse learning using a Frank-Wolfe algorithm

For this experiment, the constraint set  $C$  is the  $\ell_1$  unit-ball and the loss function is  $L(\mathbf{w}) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$ . Our objectives are two-folded:

- 1) analyze the capability of inexact gradient approaches to recover the true support and compare them to the FW algorithm,
- 2) compare the two stopping criteria for computing the inexact gradient : the *stability condition* and the *error bound condition*.

In the latter, while this *error bound condition* provides an adaptive condition for stopping — recall that the parameter  $\epsilon$  in Equation (17) is determined automatically through the data and the related curvature of the loss function —, the *stability condition* needs a user-defined parameter  $N_s$  for stopping the accumulation of the partial gradient. In the same spirit, we use a fixed pre-defined budget of pulls in the best-arm identification problem. This budget is given as a ratio of  $n \times d$ . The exact gradient is computed using the accumulation strategy as given in Equation (7) so as to make all running times comparable. The maximum number of iterations for FW is set to 5000.

Figure 1 presents the results obtained for  $n = 2000$  samples,  $d = 4000$  dictionary elements and  $k = 50$  active atoms. We depict the running time and recovery abilities of the Frank-Wolfe algorithm with an exact gradient (**exact**), a greedy deterministic gradient sampling computation with a stability stopping criterion (**deterministic**) and an error bound stopping criterion (**grad upb**), the randomized approach with an uniform sampling (**uniform**), and with a best probability sampling as given in Equation (14) (**best**), the successive reject bandit (**succ**), the *non-iid* successive halving approach with losses computed as given in Equation (15) (**SuccHalvSame**) with a random uniform sampling and the non-stochastic successive halving approach with losses computed as given in Equation (16) (**SuccHalvNonStoch**) using a permutation  $\tau$  that defines a decreasing ordering of the  $\|\mathbf{x}_i\| \|r_i\|$ .

The figures depict the performances with respect to the stopping condition parameter  $N_s$  of the stability criterion (the first value in the bracket) and the sampling budget of the bandit approach ( $\frac{n \times d}{10} z$  where  $z$  is the second value in the bracket). First, we can note that the deterministic approach used with any stopping criterion and the non-stochastic successive halving approaches are able to perfectly recover the exact support of the true vector  $\mathbf{w}^*$ , regardless of the considered stopping criterion's value. Randomized approaches with uniform and best probability sampling nearly achieve perfect recovery with an average F-measure of 0.975 with the stability criterion  $N_s$  equal to 5 for the uniform approach. When  $N_s$  increases, the

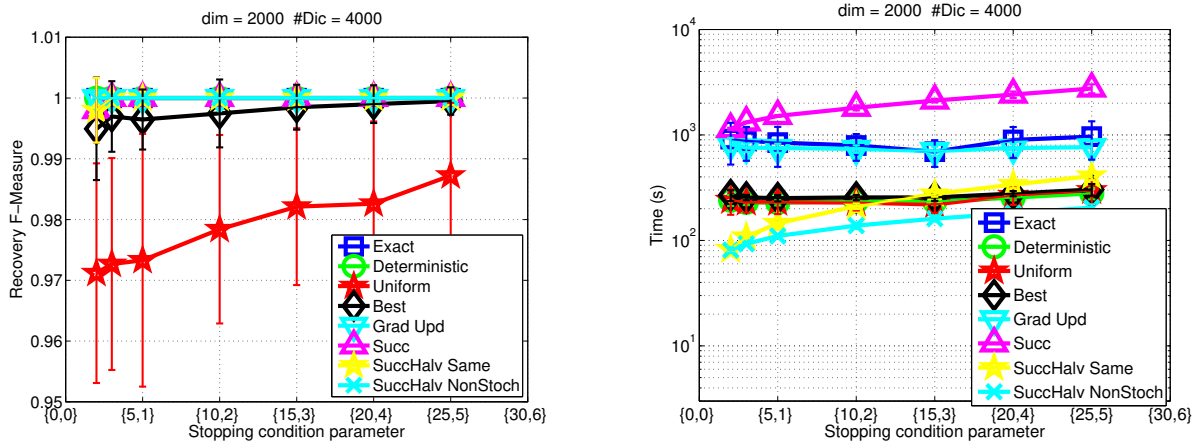


Fig. 1. Comparing vanilla Frank-Wolfe and inexact FW algorithms with different ways for computing the inexact gradient with  $n = 2000$ ,  $d = 4000$  and  $k = 50$ . Performances are compared with increasing precision on the inexact computations. We report the exact recovery performance and the running time of algorithms.

performances of these two approaches also increase but still fail to achieve perfect recovery.

From a running time point of view, the proposed approaches based on greedy deterministic and randomized sampling strategies with stability criterion and the successive halving strategies are faster than the exact FW approach, the plain *successive reject* method acting on single entry of  $\{x_{j,i}r_j\}$  and the deterministic method with the *error bound condition*. For instance, the greedy deterministic approach (green curve) achieves a gain in running time of a factor 2 with respect to the exact Frank-Wolfe algorithm. Interestingly for the greedy deterministic approach and the successive halving approaches, this gain is achieved without compromise on the recovery performance. For the randomized strategies, increasing the stability parameter  $N_s$  leads to a very slight increase of running time, hence for these methods, a trade-off can eventually be found. When comparing bandit approaches, one can note the substantial gain in performances that can be obtained by the halving strategy, the *non-iid* and *non-stochastic* strategies compared to *successive reject*. We conjecture that this higher computational running time of the *successive reject* algorithm is essentially due to computational overhead needed for accessing each single matrix entry  $\mathbf{X}_{i,j}$  in memory while all other methods use slices of this matrix (through the samples  $\mathbf{x}_i$ ) and thus they can leverage on the chunk of memory access. Best performances jointly in recovery and running times are achieved by the greedy deterministic and the non-stochastic successive halving approaches.

When comparing the *stability* and the *error bound* stopping criteria, the latter one is rather inefficient. While grounded on theoretical analysis, this bound is loose enough to be non-informative. Indeed, a careful inspection shows that the *error bound* criterion accumulates about 5 times more elements  $r_i\mathbf{x}_i$  than the stability one before triggering. In addition, other computational overheads necessary for the bound estimation, make the approach just as efficient as the exact Frank-Wolfe algorithm.

In summary, from this experiment, we can conclude that the non-stochastic bandit approach is the most efficient one. It can

achieve a gain in computation of about an order of magnitude (the left most point in the Figure 1's right panel) without compromising accuracy. The greedy deterministic approach with stability criterion performs also very well but it is slightly less efficient. We can remark that these two best methods both use the same strategy of gradient accumulation based on decreasing ordering of  $\|\mathbf{x}_i\|/|r_i|$ .

### C. Sparse Approximation with OMP

Here, we evaluate the usefulness of using inexact gradient in a greedy framework like OMP. The toy problem is similar to the one used above except that we analyze the performance of the algorithm for an increasing number  $k$  of active atoms and two sizes of dictionary matrix  $\mathbf{X}$  have been considered.

The same ways for computing the inexact gradient are evaluated and compared in terms of efficiency and correctness to the true gradient in an OMP algorithm. For all sampling approaches, the stopping criterion for gradient accumulation is based on the stability criterion with the parameter  $N_s$  adaptively set at 2% of the number  $n$  of samples. For the successive reject bandit approach, the sampling budget has been limited to 20% of the number of entries (which is  $n \cdot d$  in the matrix  $\mathbf{X}$ ). In all cases, the stopping criterion for the OMP algorithm is based on a fixed number of iterations and this number is the desired sparsity  $k$ .

Results are reported in Figure 2. They globally follow the same trend as those obtained for the Frank-Wolfe algorithm. First, note that in terms of support recovery, when the number of active atoms is small, the greedy deterministic approach performs better than the randomized sampling strategies. Bandit approaches perform similar to the greedy deterministic method. As the number of active atoms increases, the bandit approaches succeed better in recovering the extreme component of the gradient while the deterministic approach is slightly less accurate. Note that for any value of  $k$ , the randomized strategies suffer more than the other strategies for recovering the true vector  $\mathbf{w}^*$  support. From a running time point of view, again, we note that the deterministic and *non-iid* successive halving bandit approaches seem to be the most

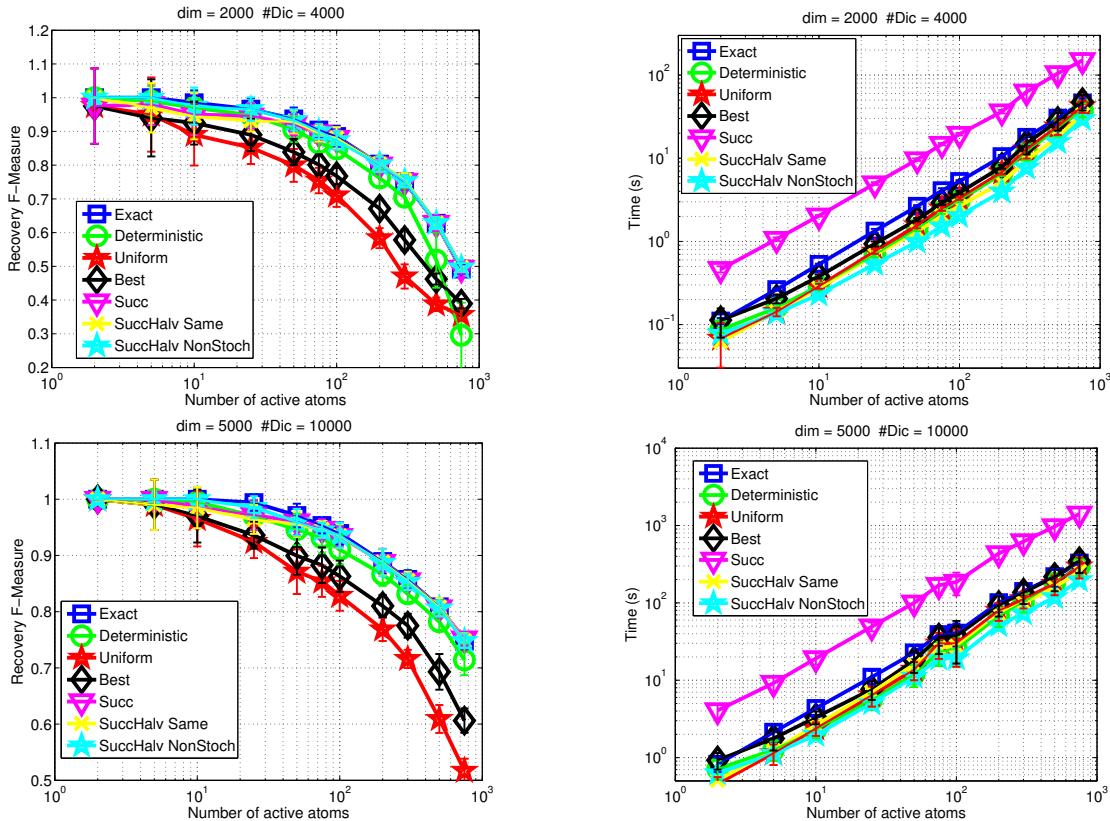


Fig. 2. Comparing OMP algorithm with different ways for computing the inexact gradient. The comparison holds for dictionary size with (top)  $n = 2000$ ,  $d = 4000$ . (bottom)  $n = 5000$ ,  $d = 10000$  and for an exact recovery criterion (left) and running time (right).

efficient methods. The gain in running time compared to the exact gradient OMP is slight but significant while it is larger when comparing with the *successive reject* algorithm.

#### D. Sparse Approximation with CoSaMP

To the best of our knowledge, there are very few greedy algorithms that are able to leverage from stochastic gradient. One of these algorithms has been introduced in [13]. In this experiment, we want to evaluate the efficiency gain achieved by our inexact gradient approach compared to this stochastic greedy algorithm. Our objective is to show that the approach we propose is empirically significantly faster than a pure stochastic gradient approach. For the different versions of the CoSaMP algorithm, we have set the stopping criterion as follows. For the CoSaMP with exact gradient approach, which serves only as a baseline for computing the exact solution, the number of iteration is set to the level of sparsity  $k$  of the target signal. A tolerance of the residual norm is also used as a stopping criterion which should be below  $10^{-3}$ . Next, for the stochastic and the inexact gradient CoSaMP versions, the algorithms were stopped when the norm of the residual  $(\mathbf{y} - \mathbf{X}\mathbf{w})$  became smaller than 1.001 times the one obtained by the exact CoSaMP or when a maximal number of iterations. Regarding gradient accumulation, the stopping criterion we choose is based on the stability condition with the parameter  $N_S$  set dynamically at 2% of the number of samples. For the bandit approaches, we have fixed the budget of pulls at

$0.2n \times d$ .

Note that for the CoSaMP algorithm, we do not look for the top entry of the gradient vector but for the top  $2k$  entries as such, we have thus straightforwardly adapted the *successive halving* algorithm to handle such a situation.

Figure 3 presents the observed results. Regarding support recovery, we remark that all approaches achieve performances similar to the exact CoSaMP. When few active atoms are in play, we can note that sometimes, the stochastic approach of [13] fails to recover the support of  $\mathbf{w}^*$ . This occurs seldom but it happens regardless of the dictionary size we have experimented with.

From a running time perspective, the results show that the proposed approaches are highly more efficient than the exact gradient approach and interestingly, they are faster than a pure gradient stochastic approach. One or two orders of magnitude can be gained depending on the level of sparsity of the signal to be recovered. This observation clearly depicts the trade-off that occurs in sparsity-constrained optimization problems in which the gradient computation and an approximation problem on a limited number of atoms are the major computational burdens (Lines 3 and 5 of Algo 1). Indeed in a stochastic gradient approach, inexact gradient computations are very cheap but more approximation problems to be solved may be needed for achieving a desired accuracy. In the approaches we propose, the inexact gradient computation is slightly more expensive but we somehow “ensure” that it provides the correct information

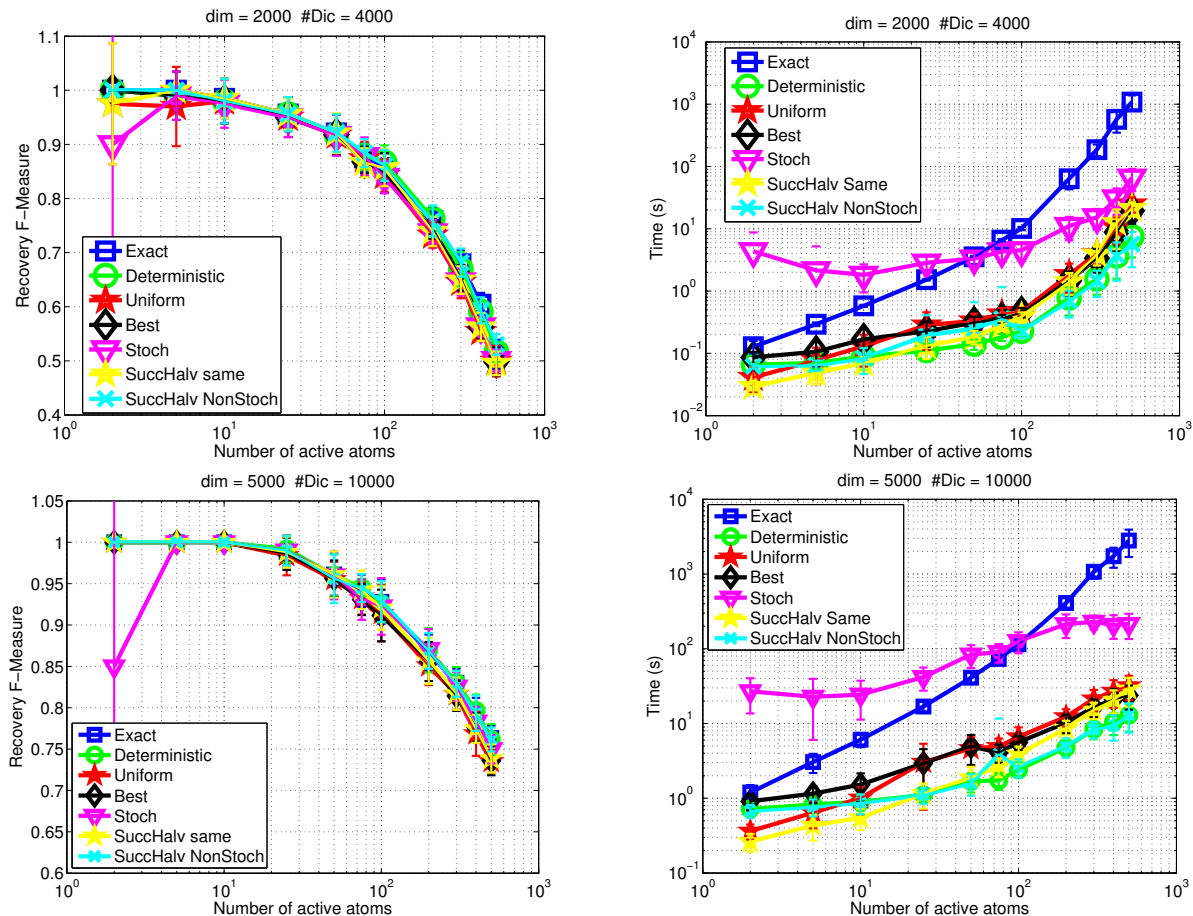


Fig. 3. Comparing CosAMP and CosAMP algorithms with different ways for computing the inexact gradient. The comparison holds for different dictionary size  $n = 2000$ ,  $d = 4000$  and  $n = 5000$ ,  $d = 10000$ . We report exact recovery performance and running time.

needed by the CoSaMP algorithm. Hence, our approaches need less approximation problems to be solved, making them more efficient than a stochastic gradient approach.

When comparing the efficiency of the proposed algorithms, the approach based on non-stochastic successive halving and the greedy deterministic approach are the most efficient especially as the number of active atoms grows.

In the second experiment, for the *successive halving* algorithms, we analyze the effect of the pull budget on the running time and on the recovery performance. We consider the following setting of the problem:  $n = 5000$ ,  $d = 10000$  and  $k = 50$ . Results are given in Figure 4. We note that a budget ratio varying from 0.05 and 0.7 allows a good compromise between ability of recovering the true vector and gain in computational time, particularly for the non-stochastic successive halving method. As the budget of pulls decreases, both algorithms fail more frequently in recovery and in addition, the computational gain substantially reduces. This experiment suggests that one should not be too greedy and should allow sufficient amount of pulls. A budget ratio of 0.2 or 0.3 for the bandit algorithms seems to be a good rule of thumb according to our experience.

Our last experiment with CoSaMP demonstrates how the

running time and the support recovery performance behave with an increasing number  $n$  of samples and afterwards with an increasing number of dictionary elements  $d$ . We have restricted our comparison to the exact and stochastic CoSaMP, and the CoSaMP variants based on the successive halving bandit algorithms and the greedy deterministic one (which are the most efficient among those we propose). The experimental setup, the stopping criterion for the CoSaMP algorithm as well as the stopping criterion for the gradient accumulation and pull budget are the same as above. Results are depicted in Figure 5. As a sanity check, we note that recovery performances are almost similar for all algorithms with slightly worse performances for the stochastic CoSaMP and the *non-iid* bandit algorithm based CoSaMP.

The computational time results show that all algorithms globally follow the same trend as the number of dictionary atoms or the number of samples increase. Recall that the computational complexity for the gradient computation is  $O(nd)$ . For the bandit approaches, we use a fixed budget of pulls dependent on  $nd$  to compute the inexact gradient. Similarly, for the greedy deterministic approach, the number of accumulation (and the stability criterion) is proportional to the number  $n$  of samples and thus the gradient computation is a constant factor of  $nd$ . Hence, our findings, illustrated on

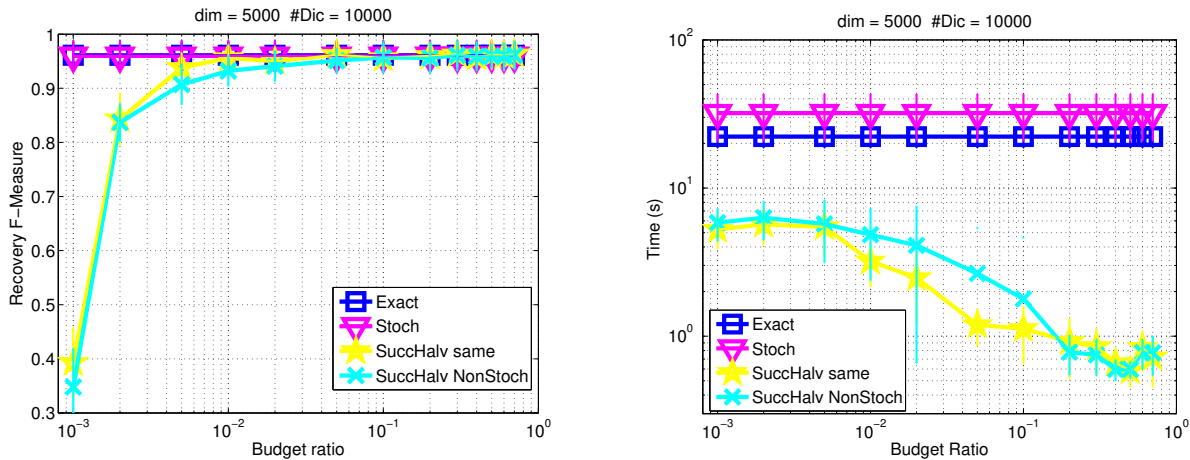


Fig. 4. Analyzing the effect of the pull budget on the successive halving algorithm (left) recovery F-measure and (right) computational running time. The pull budget is defined as the budget ratio times  $n \cdot d$ . Here  $n = 5000$ ,  $d = 10000$  and  $k = 50$ .

Figure 5, are somewhat natural since the main differences of running time essentially come from a constant factor. This factor is highly dependent on the problem but according to our numerical experiments, a ten-fold factor computational gain can be expected in many cases.

#### E. Application to audio data

We have compared the efficiency of the approaches we propose on a real signal processing application. The audio dataset we use is the one considered by Yaghoobi et al. [34]. This dataset is composed of an audio sample recorded from a BBC radio session which plays classical music. From that audio sample, 8192 pieces of signal have been extracted, each being composed of 1024 time samples. Details about the dataset can be found in [34]. From this dataset, we have learned 2048 dictionary atoms using the approach described in [35]. Our objective is to perform sparse approximation of each of the 8192 audio pieces over the 2048 dictionary atoms using CoSaMP and we want to evaluate the running time and the approximation quality of a CoSaMP algorithm using an exact gradient computation (**Exact**), a stochastic gradient CoSaMP algorithm (**Stoch**  $k$ ) and the CoSaMP variants with inexact gradient computations as we propose. The approximation error is measured as  $\frac{\|\mathbf{y} - \hat{\mathbf{y}}\|_2}{\|\mathbf{y}\|_2}$  where  $\mathbf{y}$  and  $\hat{\mathbf{y}}$  are respectively the true audio piece and its CoSaMP-based approximation. We thus want to validate that our approaches achieve similar approximation performance than CoSaMP while being faster. For all algorithms, the number of CoSaMP iterations is fixed to the sparsity pattern, here fixed to  $k = 10$ . Note that for the stochastic gradient approach, we have also considered a version with more iterations (**Stoch**  $3k$ ). Results are gathered in Table I and they are obtained as the averaged performance when approximating all the 8192 pieces of audio signal in the dataset. We can see that the inexact approaches we introduce lead to the best compromise between approximation error and running time. For instance, our *successive halving* algorithms achieve similar approximation errors than the exact CoSaMP but they are 3 times faster. At the contrary, the stochastic

TABLE I  
APPROXIMATION PERFORMANCE RESULTS AND RUNNING TIME FOR CoSaMP AND VARIANTS.  $\mathbf{y}$  AND  $\hat{\mathbf{y}}$  RESPECTIVELY DEPICTS THE SIGNAL AND ITS RESULTING APPROXIMATION. RESULTS ARE AVERAGED OVER THE APPROXIMATION OF 4500 SIGNALS. *Stoch*  $3k$  DENOTES THE STOCHASTIC GRADIENT ALGORITHM THAT USED  $3k$  ITERATIONS.

Approches	$\frac{\ \mathbf{y} - \hat{\mathbf{y}}\ _2}{\ \mathbf{y}\ _2}$	Time (s)
exact	$0.376 \pm 0.22$	$0.164 \pm 0.01$
stoch $k$	$0.670 \pm 0.13$	$0.026 \pm 0.00$
stoch $3k$	$0.570 \pm 0.17$	$0.076 \pm 0.01$
uniform	$0.351 \pm 0.21$	$0.133 \pm 0.02$
deterministic	$0.361 \pm 0.22$	$0.187 \pm 0.02$
SuccHalvSame	$0.371 \pm 0.22$	$0.059 \pm 0.01$
SuccHalvNonStoch	$0.374 \pm 0.22$	$0.064 \pm 0.01$

gradient CoSaMP approaches are efficient but lack in properly approximating target audio pieces.

#### F. Benchmark classification problems

We have also benchmarked our algorithms on real-world high-dimensional learning classification problems. These datasets are frequently used for evaluating sparse learning problems [36], [4] and more details about them can be found in these papers. Here, we considered CoSaMP as a learning algorithm and our objective is to validate the fact that the approaches we propose for computing approximate gradient are able to speed up computation time while achieving the same level of accuracy as the exact gradient. For the approximate gradient computations, we have considered the stability criterion with  $N_S = \frac{n_{train}}{20}$  ( $n_{train}$  being the number of training examples) for the deterministic and randomized approaches, and we have set the budget as  $0.2n_{train} \times d$  for the bandit approaches.

The protocol we have set up is the following. Training and test sets are obtained by randomly splitting the dataset in a 80% – 20% fold. For model selection, the training set is further split in two sets of equal size. The parameter we have cross-validated is the number of non-zero elements  $k$  in  $\mathbf{w}$ . It has been selected among 10, 50, 100, 250 so as to maximize the accuracy on the validation set. This value of  $k$

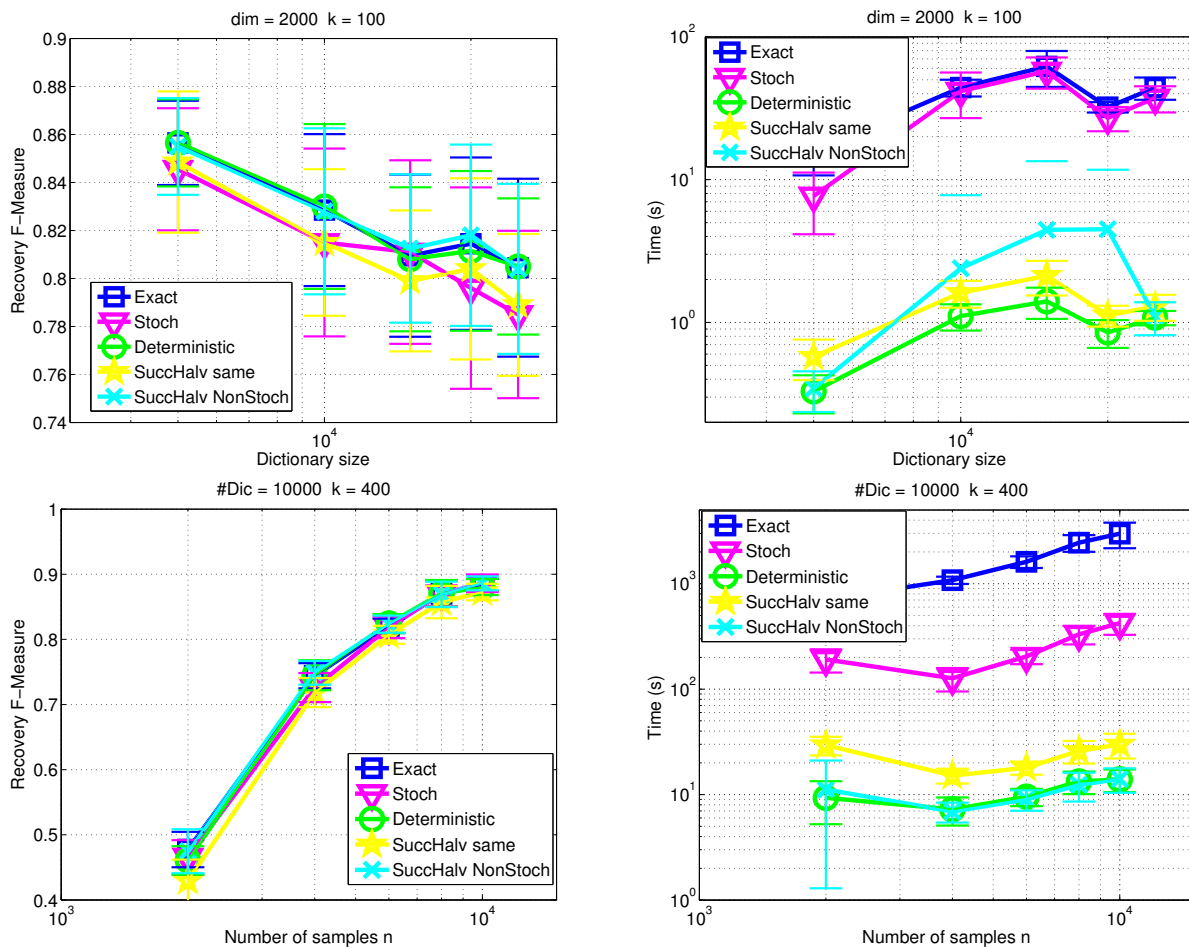


Fig. 5. (top left and top right) Evaluating how the recovery capability behaves and how the computation time scales with the number of dictionary elements (with  $n = 2000$  and  $k = 100$ ). (bottom left and bottom right) Evaluation of the same criteria with respect to the number of samples (with  $d = 10000$  and  $k = 400$ ).

has also been used as the maximal number of iterations for all the algorithms except for the stochastic ones. For these, we have reported accuracies and running times for a number of maximal iterations of  $k$  and  $3k$ .

Results averaged over 20 replicas of the training and test sets are reported in Table II. Stochastic approaches fail in learning a relevant decision function. We can note that our deterministic and randomized approaches are more efficient than the exact CoSaMP but are less accurate. On the other hand, our bandit approaches achieve nearly similar accuracy to CoSaMP while being at least 30 times faster.

## VI. CONCLUSIONS

The methodologies proposed in this paper aim at accelerating sparsity-constrained optimization algorithms. This is made possible thanks to the key observation that, at each iteration, only the component of the gradient with smallest or largest entry is needed, instead of the full gradient. By exploiting this insight, we proposed greedy algorithms, randomized approaches and bandit-based best arm identification methods for estimating efficiently this top entry. Our experimental results show that the bandit and the greedy approaches seem to be the most efficient methods for this estimation. Interestingly, the

bandit approaches come with guarantees that, given a sufficient number of draws, this top entry can be retrieved with high-probability.

Future works will be geared towards gaining further theoretical understandings on the good behaviour of the greedy approach, linking the number of iterations needed for the Frank-Wolfe algorithm to converge, with the quality of the gradient approximation in the greedy and randomized approaches, analyzing the role of the importance sampling in the randomized methods. In addition, we plan to explore how this work can be extended to an online and/or distributed computation setting.

## REFERENCES

- [1] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society*, vol. 58, no. 1, pp. 267–288, 1996.
- [2] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression (with discussion)," *Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.
- [3] M. Jaggi, "Revisiting frank-wolfe : Projection free sparse convex optimization," in *Proceedings of the International Conference on Machine Learning*, 2013.
- [4] Alain Rakotomamonjy, Remi Flamary, and Gilles Gasso, "Dc proximal newton for nonconvex optimization problems," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 1, no. 1, pp. 1–13, 2015.

TABLE II

COMPARING PERFORMANCES OF CoSAMP AND ITS VARIANTS WITH APPROXIMATE GRADIENTS ON REAL-WORLD HIGH-DIMENSIONAL CLASSIFICATION PROBLEMS. (TOP) STATISTIC SUMMARY OF DATASETS. (MIDDLE) ACCURACY OF THE DECISION FUNCTION. (BOTTOM) RUNNING TIME (IN SECONDS) OF THE LEARNING ALGORITHMS.

		Datasets				
Information	ohscal	classic	la2	hitech	sports	
n	11162	7094	3075	2301	8580	
d	11465	41681	31472	10080	14866	

		Datasets				
Algorithms	ohscal	classic	la2	hitech	sports	
CoSAMP	82.93 ± 1.9	83.46 ± 1.8	81.20 ± 2.5	76.29 ± 3.7	93.07 ± 1.3	
Stoch	56.16 ± 19.3	63.90 ± 22.6	70.24 ± 5.1	11.63 ± 11.1	40.74 ± 22.8	
Stoch 3k	36.92 ± 26.2	52.58 ± 26.8	69.76 ± 5.5	8.42 ± 3.1	32.54 ± 20.4	
Determ.	83.20 ± 1.6	82.35 ± 1.3	77.29 ± 2.3	75.98 ± 2.9	92.30 ± 1.4	
Uniform	77.50 ± 1.7	77.17 ± 1.2	78.98 ± 2.9	68.88 ± 3.6	92.33 ± 1.4	
HalvingSame	81.39 ± 1.5	82.57 ± 1.5	81.54 ± 2.6	75.16 ± 2.1	93.23 ± 1.2	
HalvingNonStoch	81.90 ± 1.5	82.97 ± 1.7	79.91 ± 2.6	76.49 ± 3.8	93.21 ± 1.0	

		Datasets				
Algorithms	ohscal	classic	la2	hitech	sports	
CoSAMP	1257.89 ± 306.3	563.51 ± 208.5	401.36 ± 222.1	56.42 ± 19.4	998.63 ± 385.3	
Stoch	3.12 ± 0.8	1.05 ± 0.7	2.98 ± 2.0	0.69 ± 0.3	4.13 ± 1.8	
Stoch 3k	7.93 ± 1.8	2.73 ± 1.6	8.50 ± 5.4	1.90 ± 0.8	12.08 ± 4.8	
Determ.	323.65 ± 76.6	142.06 ± 62.3	209.70 ± 130.5	41.24 ± 16.3	353.04 ± 129.9	
Uniform	416.74 ± 102.2	189.14 ± 74.4	150.28 ± 89.2	21.00 ± 7.7	348.30 ± 117.0	
HalvingSame	17.50 ± 5.2	13.20 ± 12.5	14.68 ± 15.2	1.48 ± 0.6	11.94 ± 6.3	
HalvingNonStoch	17.61 ± 5.4	11.74 ± 11.9	13.54 ± 15.9	1.37 ± 0.5	10.57 ± 6.5	

- [5] L. Laporte, R. Flamary, S. Canu, S. Dejean, and J. Mothe, "Nonconvex regularizations for feature selection in ranking with sparse svm," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 25, no. 6, pp. 1118–1130, 2014.
- [6] S. Mallat and Z. Zhang, "Matching pursuit with time-frequency dictionaries," *IEEE Trans Signal Processing*, vol. 41, no. 12, pp. 3397–3415, 1993.
- [7] Y.C. Pati, R. Rezaifar, and P. Krishnaprasad, "Orthogonal matching pursuit : Recursive function approximation with applications to wavelet decomposition," in *Proc. of the 27th Annual Asilomar Conference on Signals, Systems and Computers*, 1993.
- [8] D. Merhej, C. Diab, M. Khalil, and R. Prost, "Embedding prior knowledge within compressed sensing by neural networks," *Neural Networks, IEEE Transactions on*, vol. 22, no. 10, pp. 1638–1649, Oct 2011.
- [9] Tong Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, pp. 116–123.
- [10] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos : Primal estimated subgradient solver for svm," in *Proceedings of the International Conference on Machine Learning*, 2007, pp. 807–814.
- [11] Shai Shalev-Shwartz and Ambuj Tewari, "Stochastic methods for  $l_1$ -regularized loss minimization," *The Journal of Machine Learning Research*, vol. 12, pp. 1865–1892, 2011.
- [12] Rie Johnson and Tong Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, 2013, pp. 315–323.
- [13] N. Nguyen, Deanna Needell, and T. Woolf, "Linear convergence of stochastic iterative greedy algorithms with sparse constraints," <http://arxiv.org/abs/1407.0088>, 2014.
- [14] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz, "Pure exploration in multi-armed bandits problems," in *Algorithmic Learning Theory*. Springer, 2009, pp. 23–37.
- [15] Alain Rakotomamonjy, Sokol Koço, and Liva Ralaivola, "More efficient sparsity-inducing algorithms using inexact gradient," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*. IEEE, 2015, pp. 709–713.
- [16] Thomas Blumensath and Michael E Davies, "Gradient pursuits," *Signal Processing, IEEE Transactions on*, vol. 56, no. 6, pp. 2370–2382, 2008.
- [17] Aleksandr Aravkin, Aurélie Lozano, Ronny Luss, and Prabhajan Kambadur, "Orthogonal matching pursuit for sparse quantile regression," in *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 11–19.
- [18] Aurélie C Lozano, Grzegorz Swirszcz, and Naoki Abe, "Group orthogonal matching pursuit for logistic regression," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 452–460.
- [19] D. Needell and J. Tropp, "Cosamp: Iterative signal recovery from incomplete and inaccurate samples," *Applied and Computational Harmonic Analysis*, vol. 26, no. 3, pp. 301–321, 2009.
- [20] Sohail Bahmani, Bhiksha Raj, and Petros T Boufounos, "Greedy sparsity-constrained optimization," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 807–841, 2013.
- [21] J. Guélat and P. Marcotte, "Some comments on wolfe's away step," *Mathematical Programming*, vol. 35, no. 1, 1986.
- [22] P. Drineas, R. Kannan, and M. Mahoney, "Fast monte carlo algorithms for matrices i: Approximating matrix multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.
- [23] Jean-Yves Audibert and Sébastien Bubeck, "Best arm identification in multi-armed bandits," in *COLT-23th Conference on Learning Theory-2010*, 2010, pp. 13–20.
- [24] Zohar Karnin, Tomer Koren, and Oren Somekh, "Almost optimal exploration in multi-armed bandits," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1238–1246.
- [25] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Workshop on Artificial Intelligence and Statistics*, 2016.
- [26] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.
- [27] J. Tropp, A. Gilbert, and M. Strauss, "Algorithms for simultaneous sparse approximation. part I: Greedy pursuit," *Signal Processing*, vol. 86, pp. 572–588, 2006.
- [28] S. Shalev-Shwartz, N. Srebro, and T. Zhang, "Trading accuracy for sparsity in optimization problems with sparsity constraints," *SIAM Journal on Optimization*, vol. 20, 2010.
- [29] R.-B. Chen, C.-H. Chu, T.-Y. Lai, and Y. Wu, "Stochastic matching pursuit for bayesian variable selection," *Statistics and Computing*, vol. 21, no. 2, pp. 247–259, 2011.
- [30] Thomas Peel, Valentin Emiya, Liva Ralaivola, and Sandrine Anthoine, "Matching pursuit with stochastic selection," in *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*. IEEE, 2012, pp. 879–883.
- [31] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher, "Block-

- coordinate frank-wolfe optimization for structural svms,” in *Proceedings of the International Conference on Machine Learning*, 2013.
- [32] Hua Ouyang and Alexander G Gray, “Fast stochastic frank-wolfe algorithms for nonlinear svms.,” in *SDM*. SIAM, 2010, pp. 245–256.
  - [33] Zhao P and Zhang T, “Stochastic optimization with importance sampling,” <http://arxiv.org/abs/1401.2753>, 2014.
  - [34] M. Yaghoobi, T. Blumensath, and M. Davies, “Dictionary learning for sparse approximations with the majorization method,” *IEEE Transaction on Signal Processing*, vol. 57, no. 6, pp. 2178–2191, 2009.
  - [35] A. Rakotomamonjy, “Direct optimization of the dictionary learning problem,” *IEEE Trans. on Signal Processing*, vol. 61, no. 12, pp. 5495–5506, 2013.
  - [36] P. Gong, C. Zhang, Z. Lu, J. Huang, and J. Ye, “A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems,” in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, Georgia, Jun. 2013, pp. 37–45.