Artificial Intelligence 306 (2022) 103683

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Efficient projection algorithms onto the weighted ℓ_1 ball

Guillaume Perez [a,*], Sebastian Ament [b], Carla Gomes [b], Michel Barlaud [c]

[a] Ugmented S.A.S. Sophia Antipolis, France
[b] Cornell University, Ithaca, NY, 14850, United States
[c] Université Côte d'Azur, CNRS, Sophia Antipolis, 06900, France

ARTICLE INFO

Article history:
Received 2 August 2021
Received in revised form 19 January 2022
Accepted 17 February 2022
Available online 23 February 2022

Keywords:
Optimization
Gradient-based methods
Variable selection

ABSTRACT

Projected gradient descent has been proved efficient in many optimization and machine learning problems. The weighted ℓ_1 ball has been shown effective in sparse system identification and features selection. In this paper we propose three new efficient algorithms for projecting any vector of finite length onto the weighted ℓ_1 ball. The first two algorithms have a linear worst case complexity. The third one has a highly competitive performances in practice but the worst case has a quadratic complexity. These new algorithms are efficient tools for machine learning methods based on projected gradient descent such as compressed sensing, feature selection. We illustrate this effectiveness by adapting an efficient compressed sensing algorithm to weighted projections. We demonstrate the efficiency of our new algorithms on benchmarks using very large vectors. For instance, it requires only 8 ms, on an Intel I7 3rd generation, for projecting vectors of size 10^7.

© 2022 Elsevier B.V. All rights reserved.

1. Introduction

Looking for sparsity appears in many machine learning applications, such as biomarker identification in biology [1,2], or the recovery of sparse signals in compressed sensing [3–5]. For example consider the problem of minimizing a common ℓ_2 reconstruction loss function. In addition consider constraining the number of non-zero components (ℓ_0 norm) of the learned vector to be lower than a given sparsity value:

$$\underset{x\in\mathbb{R}^d}{\text{minimize}}\|Ax-b\|_2 \quad \text{subject to} \quad \|x\|_0 < \epsilon$$

A solution of this generic problem will use only a subset of size lower than ϵ of the components of x leading to the best reconstruction error. This implies that solving this problem and varying the ϵ value allows us to manage sparsity with a fine grain. Unfortunately, this problem is generally strictly nonconvex and very difficult to solve [6]. Hence a common solution is to constrain the ℓ_1 norm of the vector instead [7,8,5,4], or one of its modified versions [9–13]. More specifically, the weighted ℓ_1 ball has been proven to have strong properties for finding or recovering sparse vectors [14,9]. To this purpose, it is crucially important to have simple algorithms to work with the weighted ℓ_1 ball during the optimization process. Several methods in machine learning are based on projected gradient descent for their optimization part, for example by iterating between gradient updates and ℓ_1 ball projections [15]. The goal of this paper is to help the application of

[*] Corresponding author.
E-mail addresses: guillaume.perez06@gmail.com (G. Perez), barlaud@i3s.unice.fr (M. Barlaud).

https://doi.org/10.1016/j.artint.2022.103683
0004-3702/© 2022 Elsevier B.V. All rights reserved.

projected gradient descent methods for solving sparse machine learning problems by working with the weighted ℓ_1 ball. Unfortunately, efficiently applying projected gradient descent requires an efficient projection algorithm. For the basic ℓ_1 ball, many projection algorithms have been proposed [16–20], but in the context of the weighted ℓ_1 ball, only few works have been done [9].

In this paper, we propose three efficient projection algorithms by generalizing works made for the basic (i.e. non weighted) ℓ_1 ball to its generalization, the weighted ℓ_1 ball. We start by giving a proof of existence of the threshold value ($\lambda *$) allowing efficient projection algorithms as for the basic ℓ_1 ball [21]. Using this threshold, we propose three efficient algorithms. First, $w\text{-}pivot^F$ (weighted pivot Filtered) iteratively approximates $\lambda *$ using a pivot-based algorithm. It splits the vector in two sub-vectors at each iteration and has quadratic worst case complexity, but is near linear in practice. Second, $w\text{-}bucket$ and $w\text{-}bucket^F$, two algorithms based on a bucket decomposition of the vector that efficiently detects components that will be zero in the projection. These two ones have linear worst case complexity. We propose an experimental protocol using randomly generated high dimensional ($> 10^5$) vectors to show the performances of our algorithms. We then adapt the sparse vector recovery framework from [22] to the proposed projection algorithms and show the efficiency and simplicity of the model. The paper starts with the definitions of the projection onto the ℓ_1 and weighted ℓ_1 balls, and highlights the need of finding the $\lambda *$ value. In the projection section, the algorithm $w\text{-}pivot^F$ is first defined and the algorithms $w\text{-}bucket$ and $w\text{-}bucket^F$ are later defined. In the experiments section, an evaluation of the time performances shows that the proposed algorithms are order of magnitude faster than current projection methods, and then that they can be used to adapt existing machine-learning frameworks. Finally, the proofs required for the proposed algorithms are given and followed by the conclusion.

2. Definitions of the projections

ℓ_1 ball. Given a vector $y = (y_1, y_2, \dots, y_d) \in \mathbb{R}^d$ and a real $a > 0$, we aim at computing its projection $P_{\mathcal{B}_a}(y)$ onto the ℓ_1 ball $\mathcal{B}_a$ of radius a:

$$\mathcal{B}_a = \left\{ x \in \mathbb{R}^d | \ \|x\|_1 \leq a \right\}, \tag{1}$$

where $\|x\|_1 = \sum_{i=1}^d |x_i|$. The projection $P_{\mathcal{B}_a}(y)$ is defined by

$$P_{\mathcal{B}_a}(y) = \arg\min_{x \in \mathcal{B}_a} \|x - y\|_2 \tag{2}$$

where $\|x\|_2$ is the Euclidean norm. As shown in [15] and revisited in [19], the projection onto the ℓ_1 ball can be derived from the projection onto the simplex Δ_a:

$$\Delta_a = \left\{ x \in \mathbb{R}^d \,|\, \sum_{i=1}^d x_i = a \text{ and } x_i \geq 0, \ \forall i = 1, \dots, d \right\}. \tag{3}$$

Let the sign function $\text{sign}(v)$ defined as $\text{sign}(v) = 1$ if $v > 0$, $\text{sign}(v) = -1$ if $v < 0$ and $\text{sign}(v) = 0$ otherwise, for any real value $v \in \mathbb{R}$. The projection of y onto the ℓ_1 ball is given by the following formula:

$$P_{\mathcal{B}_a}(y) = \begin{cases} y & \text{if } y \in \mathcal{B}_a, \\ (\text{sign}(y_1)x_1, \dots, \text{sign}(y_d)x_d) & \text{otherwise,} \end{cases} \tag{4}$$

where $x = P_{\Delta_a}(|y|)$ with $|y| = (|y_1|, |y_2| \dots, |y_d|)$, with $|y_i|$ the absolute value of y_i, is the projection of $|y|$ onto Δ_a. An important property has been established to compute this projection. Let $y = (y_1, y_2, \dots, y_d) \in \mathbb{R}^d_{\geq 0}$ It was shown [21] that there exists a unique $\tau = \tau_y \in \mathbb{R}$ such that

$$x_i = \max\{y_i - \tau, 0\}, \forall i = 1, \dots, d. \tag{5}$$

The projection is almost equivalent to a thresholding operation. The main difficulty is to compute quickly the threshold τ_y for any vector y. Let $y_{\downarrow()}$ be the decreasing sorted order. Let $y_{\downarrow(i)}$ be the ith largest value of y such that $y_{\downarrow(1)} \geq y_{\downarrow(2)} \geq \dots \geq y_{\downarrow(d)}$. It is interesting to note that (5) involves that $\sum_{i=i}^d \max\{y_i - \tau, 0\} = a$. Let S^* be the support of x, i.e., $S^* = \{i | x_i > 0\}$. Then,

$$a = \sum_{i=1}^d x_i = \sum_{i \in S^*} x_i = \sum_{i \in S^*} (y_i - \tau).$$

It follows that $\tau_y = (\sum_{i \in S^*} y_i - a)/|S^*|$ where $|S^*|$ is the number of elements of S^*. The following property allows us to compute the threshold τ_y. Let

$$\varrho_j(y) = \left(\sum_{i=1}^j y_{\downarrow(i)} - a \right) / j \tag{6}$$

for any $j = 1, \ldots, d$. Then, it was shown that $\tau_y = \varrho_{K_y}(y)$ where

$$K_y = \max\{k \in \{1, \ldots, d\} \mid \varrho_k(y) < y_{\downarrow(k)}\}. \tag{7}$$

Looking for K_y, or equivalently $y_{\downarrow(K_y)}$, allows us to find immediately the threshold τ_y. The most famous algorithm to compute the projection, which has been presented in [21], is based on (7). It consists in sorting the values and then finding the maximum index i satisfying (7). A possible implementation is given in Algorithm 1. The worst case complexity of this algorithm is $O(d \log d)$. Several other methods have been proposed [16–20], outperforming this simple approach.

Algorithm 1: Sort based algorithm [21].

```
Data: $y, a$
$u \leftarrow sort_{\downarrow}(y)$
$K \leftarrow \max_{1 \le k \le d}\{k | (\sum_{r=1}^{k} u_r - a)/k < u_k\}$
$\tau \leftarrow (\sum_{r=1}^{K} u_r - a)/K$
for $i \in \{1..d\}$ do
    $x_i \leftarrow \max(y_i - \tau, 0)$
```

Weighted ℓ_1 ball. Given a vector $y = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d$, a vector $w = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$, $w_i > 0$[1] for all i, and a real $a > 0$, we aim at computing its projection $P_{\mathcal{B}_{w,a}}(y)$ onto the w weighted ℓ_1 ball $\mathcal{B}_{w,a}$ of radius a:

$$\mathcal{B}_{w,a} = \left\{ x \in \mathbb{R}^d | \sum_{i=1}^{d} w_i |x_i| \le a \right\}, \tag{8}$$

The projection $P_{\mathcal{B}_{w,a}}(y)$ is defined by

$$P_{\mathcal{B}_{w,a}}(y) = \arg \min_{x \in \mathcal{B}_{w,a}} \|x - y\|_2 \tag{9}$$

where $\|x\|_2$ is the Euclidean norm.

As for the classical ℓ_1 ball, the weighted projection operator can be derived from the weighted simplex $\Delta_{w,a}$ projection [9]:

$$\Delta_{w,a} = \left\{ x \in \mathbb{R}^d \mid \sum_{i=1}^{d} w_i x_i = a \text{ and } x_i \ge 0, \forall i = 1, \ldots, d \right\}. \tag{10}$$

The projection of y onto the weighted ℓ_1 ball is given by the following formula:

$$P_{\mathcal{B}_{w,a}}(y) = \begin{cases} y & \text{if } y \in \mathcal{B}_{w,a}, \\ (\text{sign}(y_1)x_1, \ldots, \text{sign}(y_d)x_d) & \text{otherwise,} \end{cases} \tag{11}$$

where $x = P_{\Delta_{w,a}}(|y|)$ with $|y| = (|y_1|, |y_2|..., |y_d|)$ is the projection of $|y|$ onto $\Delta_{w,a}$. Once again, the fast computation of the projection $x = P_{\Delta_{w,a}}(y)$ for any vector y is of utmost importance.

Let the vector $y = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d$, the vector $w = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$, $w_i \ge 0$ for all i, and a real $a > 0$, if $y \notin \Delta_{w,a}$, then there exists a unique $\lambda = \lambda_y \in \mathbb{R}$ such that

$$x_i = \max\{y_i - w_i \lambda, 0\}, \forall i = 1, \ldots, d. \tag{12}$$

The proof is given in the proof section of this paper, and is used to derive three projection algorithms. These algorithms are generalizations of the current state-of-the-art algorithms for ℓ_1 ball projection [19,20]. The main difficulty is to compute quickly the threshold λ_y for any vector y. Let $z \in \mathbb{R}^d$ be the vector such that

$$z_i = \frac{y_i}{w_i}, \forall i = 1, \ldots, d. \tag{13}$$

Let $z_{\uparrow()}$, $y_{\uparrow()}$ and $w_{\uparrow()}$ be the increasing sorted order with respect to z. Let $z_{\uparrow(i)}$ be the ith value of z such that $z_{\uparrow(1)} \le z_{\uparrow(2)} \le \ldots \le z_{\uparrow(d)}$. Let $y_{\uparrow(i)}$ (resp. $w_{\uparrow(i)}$) be the jth entry of y such that $z_{\uparrow(i)} = y_j/w_j$. $y_{\uparrow()}$ is a permutation of y with respect to the order of $z_{\uparrow()}$. It is interesting to note that equation (12) implies that $\sum_{i=i}^{d} \max\{y_i - w_i\lambda, 0\} = a$. Let S^* be the support of x, i.e., $S^* = \{i | x_i > 0\}$. Then,

[1] We can consider $w_i > 0$ instead of $w_i \ge 0$ without loss of generality since the associated entries of y will be present in the projection, and do not influence the processing of the rest of the projection.

$$a = \sum_{i=1}^{d} w_i x_i = \sum_{i \in S^*} w_i x_i = \sum_{i \in S^*} w_i (y_i - w_i \lambda).$$

It follows that

$$\lambda = \frac{\sum_{i \in S^*} w_i y_i - a}{\sum_{i \in S^*} w_i^2} \tag{14}$$

The following property computes the threshold λ_j.

$$\varrho_j(w, y) = \frac{\sum_{i=j}^{d} w_{\uparrow(i)} y_{\uparrow(i)} - a}{\sum_{i=j}^{d} w_{\uparrow(i)}^2} \tag{15}$$

for any $j = 1, \ldots, d$. Then, we have shown in the proof section that $\lambda_y = \varrho_{K_y}(w, y)$ where

$$K_y = \max\{k \in \{1, \ldots, d\} \mid \varrho_k(w, y) < z_{\uparrow(k)}\}. \tag{16}$$

Looking for K_y, or equivalently $y_{\uparrow(K_y)}$, gives us immediately the threshold λ.

A direct algorithm to compute this projection, which is a generalization of [21], is based on (16), and is given in Algorithm 2. This algorithm starts by sorting the values according to z, and then searches for the K_y index. Note that once z sorted, finding K_y is easily done by starting from the largest value. That is why Algorithm 2 does not need more steps as in [9,10].

Algorithm 2: Weighted generalization of the sort based algorithm.

```
Data: $y, w, a$
Output: $x = P_{\mathcal{B}_{w,a}}(y)$
$z^u \leftarrow \{ \frac{y_i}{w_i} | \forall i \in \{1..d\}\}$
$\uparrow() \leftarrow$ Permutation$\uparrow$ $(z^u)$
$z \leftarrow \{z^u_{\uparrow(i)} | \forall i \in \{1..d\}\}$
$J \leftarrow \max_{1 \le J \le d}\{\arg\max j : \frac{-a + \sum_{i=j+1}^{d} w_{\uparrow(i)} y_{\uparrow(i)}}{\sum_{i=j+1}^{d} w_{\uparrow(i)}^2} > z_j\}$
$\lambda^* \leftarrow \frac{-a + \sum_{j=J+1}^{d} w_{\uparrow(j)} y_{\uparrow(j)}}{\sum_{j=J+1}^{d} w_{\uparrow(j)}^2}$
for $i \in \{1..d\}$ do
    $x_i \leftarrow \mathrm{sign}(y_i) \max(y_i - w_i \lambda^*, 0)$
```

The ordered weighted ℓ_1 norm [12] is another type of norm with some interesting statistical properties such as clustering. Such norm is outside the scope of the paper since is consider w to be ordered. The main goal of this paper is to avoid any sorting algorithms to be applied to the vector. If the data are sorted, finding J becomes trivial.

3. Efficient projection

Many works have focused on designing efficient algorithms for finding $P_{\mathcal{B}_a}(y)$ given $y \in \mathbb{R}^d$ and $a \in \mathbb{R}$ [16,18,17,19,20]. In this section, given y, w, and a, we propose to generalize some of these efficient algorithms to find the projection onto the weighted ℓ_1 ball $\mathcal{B}_{a,w}$. Specifically, we generalize the methods from [19,20] that we respectively name $pivot^F$ and $bucket^F$. Both of these methods are looking for the τ value such that the projection $P_{\mathcal{B}_a}(y) = x$ is defined by equation (5).

For the weighted ball, we are looking for the λ value such that the projection $P_{\mathcal{B}_{a,w}}(y) = x$ is given by equations (11) and (12). As for the basic ℓ_1 ball, the weighted ℓ_1 ball algorithms are looking for K and $y_{\uparrow(K)}$ from equation (16). Let $z = \frac{y}{w}$. If z is sorted in increasing order, then finding λ can be easily done by iteratively processing $\varrho_i(w, y)$, with $i = d, \ldots, 1$ until $\varrho_i(w, y) > (z_i)$, as shown in Algorithm 2. A projection algorithm using as a first iteration a sort has already been proposed [9]. But most of the time z is not sorted, and sorting z is the exact operation that we want to avoid because of its time complexity, and that often, we are looking for sparse solutions, thus only a subset of the values of y will remain relevant. This section is split into two parts. The first one is a generalization of the algorithm $pivot^F$ [19]. The second one is a generalization of the algorithm $bucket^F$ [20].

3.1. w-$pivot^F$ Algorithm

In this section we propose a generalization of the $pivot^F$ algorithm. The proposed algorithm is composed of three points that we name pivot, lower bound extraction and online filtering, and are described in the next paragraphs. The idea of the algorithm is the following; at each iteration, the vector is split into two sub-vectors, using a pivot value, and determine which sub-vector contains $y_{\uparrow(K)}$ (pivot part). In the mean time, a fine grain lower bound is defined as a pivot for an efficient splitting (lower bound part). Finally, the algorithm discards *on the fly* values that are provably not part of S^* (online filtering part).

Pivot It is often considered that for regular amount of data, the *quicksort* algorithm is the fastest sort [23]. The *quicksort* algorithm splits the data into two partitions by using a pivot value. Values smaller than the pivot go into the first partition, others in the second. The process is then applied recursively to both partitions. In the context of the basic simplex projection, using a *pivot like* algorithm led to some of the most efficient algorithms [18,15,16,19]. From equation (14), we can notice that the λ value only requires the knowledge about elements of the set $S*$, but not that this set is ordered. Such a remark gives a hint in why partitioning instead of sorting could be beneficial. Consider $p \in [z_{\uparrow(1)}, z_{\uparrow(d)}]$, note that to get $z_{\uparrow(1)}$ and $z_{\uparrow(d)}$ only one pass over the vector z is required. One can partition z into z_{low} and z_{high} by putting the elements smaller (resp larger) than p from z. Consider that the size of z_{high} is j (i.e. it contains j entries), then we can easily compute $\varrho_j(w, y)$. If $\varrho_j(w, y) \geq p$, then $\lambda \geq p$. If this condition is true, this implies that we can stop the processing of z_{low} because $z_{\uparrow(K_y)} \in z_{high}$. If the condition is false, if $\varrho_j(w, y) \leq p$, then $\lambda \leq p$, then we know that $\{i | z_i \in z_{high}\} \subseteq S*$. Using this knowledge, we can continue the processing of z_{low}.

Lower bound as pivot The choice for the pivot is of utmost importance for the global running time, we can easily show that the worst case complexity is $O(d^2)$. As for the basic simplex, one could seek for the median pivot [18]. From a complexity point of view, this could be an improvement, but our goal is not to partition equitably the data, but to efficiently find λ. Instead, we will seek for a pivot which is a lower bound of λ.

Let V be any sub-sequence of the component indices of vector y. Let p_V be a pivot value defined with respect to V.

$$p_V = \frac{-a + \sum_{i \in V} w_i y_i}{\sum_{i \in V} w_i^2} \tag{17}$$

Proposition 1. *Let V be any sub-sequence of the component indices of vector y. Using p_V as a pivot allows to directly discard elements of z_{low}.*

Proof. If we use p_V as a threshold value, and by definition of a lower bound, we have:

$$a \leq \sum_{i \in \{1..d\}} w_i \max(y_i - w_i p_V, 0) \tag{18}$$

The proof can be found at the end of this paper.

If $z_i \in z_{low}$, it implies that $\max(y_i - w_i p_V, 0) = 0$, which also implies that $\max(y_i - w_i \lambda, 0) = 0$. Consider the algorithm iterating between the following step: Step 1) Set $p = p_{z_{high}}$. Step 2) remove elements of z_{high} which are smaller than p. This algorithm will converge to a state where no element can be removed anymore. This state implies that the resulting vector z_{high} is $S*$.

Online Filtering Another optimization of this algorithm is the following, the pivot value does not need to wait until the end of step 2) before being updated, but can be updated after every element of z_{high} is read. Such an interactive update requires us to divide the algorithm in two parts.

The first part is the first pass over y, where we do not have any knowledge about y. Let initialize $V = \{1\}$ and the pivot by $p_V = \frac{w_1 y_1 - a}{w_1^2}$. Consider that we are processing the jth element of y, which implies that we have already processed all the elements in $[1, j-1]$. From these elements, we have built a sub-sequence V, and an associated pivot p_V. If we have $z_i \leq p_V$, then, as before, we can discard this element. Otherwise, if $z_i > p_V$, then we can add i to V because z_i is potentially larger than λ. Once V is updated, we can update p_V incrementally without waiting for the pass to be over, and then process the $(j+1)$th element of y.

For the second part, we have hopefully already discarded several elements of y, and more importantly, we have a set V containing elements that are potentially larger than λ. In addition, we have the associated pivot p_V, which is processed with respect to all the elements in V. Consider that we pass over the elements of $z_i \in V$, if $z_i \leq p_V$, then we can remove z_i from V, and update p_V accordingly. Note that just like for the first part, this can be done incrementally. When no more element can be removed from V, the algorithm finished and $V = S*$. A possible implementation is given in Algorithm 3.

In the same fashion as [19], we have incorporated a refinement. During the first iteration, while processing the ith element, when the current pivot value is smaller than the one defined by the current value $p' = \frac{w_i y_i - a}{w_i^2}$, then p' will become the new basis. To do that, an additional set v' is required and a cleanup step which ensures that V contains all the elements z_i larger than p_V before the second part of the algorithm. Note that the proposed algorithm works on the permuted space of z without using z for the calculation of J and λ, which is a major difference with the non-weighted version.

3.2. *w-bucket^F Algorithm*

In this section, we present the *w-bucket^F* algorithm, which is a generalization of the linear time simplex projection *bucket^F* [20]. *w-bucket^F* fundamental idea is to recursively split vector z into $B \geq 2$ ordered sub-vectors (say buckets) $\tilde{z}_b^k$

Algorithm 3: $w\text{-}pivot^F$.

```
Data: $y, w, a$
$v \leftarrow \{y_1\}$ # a set containing only $y_1$
$\tilde{v} \leftarrow \emptyset$ # an empty set
$\lambda' \leftarrow \frac{w_1 y_1 - a}{w_1^2}$
for $n \in \{2..d\}$ do
    if $\frac{y_n}{w_n} > \lambda'$ then
        $\lambda' \leftarrow \frac{w_n y_n - a + \sum_{i \in v} w_{\uparrow(i)} y_{\uparrow(i)}}{w_n^2 + \sum_{i \in v} w_{\uparrow(i)}^2}$
        if $\frac{w_n y_n - a}{w_n^2} < \lambda'$ then
            Insert $n$ in set $v$
        else
            Insert $v$ in set $\tilde{v}$
            $v \leftarrow \{y_n\}$
            $\lambda' \leftarrow \frac{w_n y_n - a}{w_n^2}$
if $\tilde{v} \neq \emptyset$ then
    for $n \in \tilde{v}$ do
        if $\frac{y_n}{w_n} > \lambda'$ then
            Insert $n$ in set $v$
            $\lambda' \leftarrow \frac{-a + \sum_{i \in v} w_{\uparrow(i)} y_{\uparrow(i)}}{\sum_{i \in v} w_{\uparrow(i)}^2}$
while $|v|$ changes do
    for $n \in v$ do
        if $\frac{y_n}{w_n} < \lambda'$ then
            Remove $n$ from set $v$
            $\lambda' \leftarrow \frac{-a + \sum_{i \in v} w_{\uparrow(i)} y_{\uparrow(i)}}{\sum_{i \in v} w_{\uparrow(i)}^2}$
$\lambda* \leftarrow \lambda'$
for $i \in \{1..|y|\}$ do
    $x_i \leftarrow \max(y_i - w_i \lambda*, 0)$
```

with $b = 1, \ldots, B$ and $k = 1, \ldots, \bar{k}$, while looking for $z_{\uparrow(K)}$. We say that the sub-vectors are ordered in the sense that all elements of $\tilde{z}_b^k$ are smaller than the ones of $\tilde{z}_{b+1}^k$ for all $b = 1, \ldots, B-1$. The depth, or number of recursive splitting is $\bar{k}$. In the description of $w\text{-}pivot^F$, from the two sub-vectors z_{low} and z_{high}, only one of them was re-used at the next iteration. In the $w\text{-}bucket^F$ algorithm, only one of the B buckets will be re-used at the next iteration. Such a fine grain gives us three possible states for the buckets, $< z_{\uparrow(K)}$, $> z_{\uparrow(K)}$, $?z_{\uparrow(K)}$. Only one bucket will be in the uncertainty state ($?z_{\uparrow(K)}$), this is the bucket that will be re-used. Fig. 1 shows a toy example of the application of this process.

We define here the different components of the $w\text{-}bucket^F$ algorithm. For any level $k+1 \geq 1$, consider the interval I^{k+1} defined by

$$I^{k+1} = [\min \tilde{z}_{b_k}^k, \max \tilde{z}_{b_k}^k] \tag{19}$$

with $\min \tilde{z}_b^k$ (resp. $\max \tilde{z}_b^k$) the minimum (resp. maximum) element of sub-vector $\tilde{z}_b^k$.

Consider a partition of I^{k+1} into B ordered sub-intervals I_1^{k+1}, ldots , I_B^{k+1}. Let $h^{k+1} : I^{k+1} \mapsto \{1, \ldots, B\}$ be the bucketing function such that $h^{k+1}(v) = b$ when the real value v belongs to I_b^{k+1}. The bucket $\tilde{z}_{b_k}^k$ is split into B ordered sub-vectors $\tilde{z}_b^{k+1}$ such that

1. $S_b^{k+1} = \{i \in S_{b_k}^k \, : \, h^{k+1}(z_i) = b\}$,
2. $\tilde{z}_b^{k+1} = (z_i)_{i \in S_b^{k+1}}$,
3. $\max \tilde{z}_b^{k+1} < \min \tilde{z}_{b+1}^{k+1}$ for all $b = 1, \ldots, B-1$,

with the convention $S_{b_0}^0 = \{1, \ldots, d\}$. We get $\left|S_B^k\right| \geq 1$ at any level $k \geq 1$ because of the definition of I^{k+1}. The fact that $\max \tilde{z}_b^{k+1} < \min \tilde{z}_{b+1}^{k+1}$ follows from the fact that equal values of z necessarily belong to the same bucket.

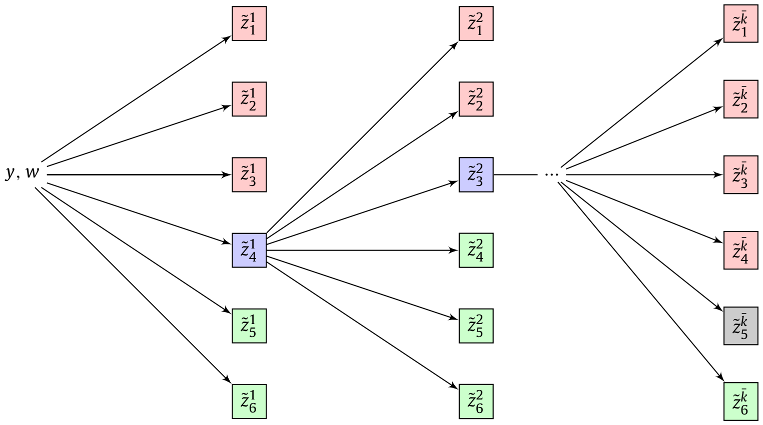

Fig. 1. Principle of “bucket filtering”. The vector y is split into 6 buckets. The buckets in green must be kept to compute the threshold λ_y. The buckets in red are not involved in the computation of λ_y. The bucket in blue must be explored in order to identify its elements which participate to λ_y. Hence, the splitting process is repeated recursively until all the values of y involved in the computation of λ_y are identified. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

For any $k > 0$,

$$C_b^k = \sum_{k'=1}^{k-1} \sum_{b'>b_{k'}} \sum_{i\in S_{b'}^{k'}} w_i y_i + \sum_{b'\geq b} \sum_{i\in S_{b'}^{k}} w_i y_i \tag{20}$$

$$W_b^k = \sum_{k'=1}^{k-1} \sum_{b'>b_{k'}} \sum_{i\in S_{b'}^{k'}} w_i^2 + \sum_{b'\geq b} \sum_{i\in S_{b'}^{k}} w_i^2 \tag{21}$$

$$N_b^k = \sum_{k'=1}^{k-1} \sum_{b'>b_{k'}} \sum_{i\in S_{b'}^{k'}} 1 + \sum_{b'\geq b} \sum_{i\in S_{b'}^{k}} 1 \tag{22}$$

are cumulative sums of all the buckets discarded because we know they belong to $S*$ from previous iterations, and all the larger or equal buckets of the iteration k. Both of these values will be used for incrementally processing $\varrho_j(w, y)$, equation (15). More precisely, we define:

$$\varrho_{N_b^k}(w, y) = (C_b^k - a)/W_b^k. \tag{23}$$

Let b be the largest value such that $\tilde{z}_b^k$ is not empty. If $\varrho_{N_b^k}(w, y) \geq \min \tilde{z}_b^k$, then $b_k = b$ and we can discard all the buckets with $b' < b$ and continue to the next iteration, since $\lambda \geq \varrho_{N_b^k}(w, y)$. Otherwise, if $\varrho_{N_b^k}(w, y) < \min \tilde{z}_b^k$, then $S_b^k \subseteq S^*$ and we can continue processing the other buckets. Let b_k be the largest value such that $\tilde{z}_{b_k}^k$ is not empty and $\varrho_{N_{b_k}^k}(w, y) \geq \min \tilde{z}_{b_k}^k$. We know that for all $b > b_k$, $S_b^k \subseteq S^*$, and that for all $b < b_k$, $\forall v \in \tilde{z}_b^k$, $v < \varrho_{N_{b_k}^k}(w, y) \leq \lambda$. Thus we can safely go to the next iteration, considering only $\tilde{z}_{b_k}^k$. Note that if $y \in \Delta_a$, then such a b_k value does not exist, and at the first iteration we can stop.

From the definition of I^k (19), we can show that the size of the bucket $\tilde{z}_{b_k}^k$ is strictly decreasing. Let $\bar{k}$ be the iteration where $z_{\uparrow(K)}$ is the minimum value of a bucket $\tilde{z}_{b_{\bar{k}}}^{\bar{k}}$. Let b' be the largest value, strictly lower than $b_{\bar{k}}$, such that $\tilde{z}_{b'}^{\bar{k}}$ is not empty. We have:

$$\max \tilde{z}_{b'}^{\bar{k}} < \varrho_{N_{b_{\bar{k}}}^{\bar{k}}}(w, y) \wedge \min \tilde{z}_{b_{\bar{k}}}^{\bar{k}} \geq \varrho_{N_{b_{\bar{k}}}^{\bar{k}}}(w, y). \tag{24}$$

Such a condition, from equation (16), implies that $\varrho_{N_{b_{\bar{k}}}^{\bar{k}}}(w, y) = \lambda$. The complexity of the *w-bucket*^{F} algorithm is highly dependent of the bucketing function h^k, and using equation (19), we can easily show that the worst case complexity is bound by $O(d^2)$. Following the same idea as [20], we can use a bucketing function based on the numbers encoding in nowadays computers. Such a function, at each iteration, choose to partition the numbers with respect to their kth byte, in the same fashion as the Radix sort [23]. Using such a bucketing function loose the property of equation (19), but the advantage is that the complexity becomes linear $O(d + B)$. An implementation is given in Algorithm 4.

Algorithm 4: *w-bucket*.

```
  Data: $y, w, a$
  $\tilde{y}^0_{b_0} \leftarrow y$
  $C^0_{b_0} \leftarrow -a$
  $W^0_{b_0} \leftarrow 0$
1 for $k \in \{1..\lceil \log_b(D) \rceil\}$ do
      for $b \in \{1..B\}$ do
          $S^k_b \leftarrow \{i \in S^{k-1}_{b_{k-1}} \mid h^{k-1}(y_i) = b\}$
          $\tilde{y}^k_b \leftarrow (y_i)_{i \in S^k_b}$
2     for $b \in \{B..1\}$ do
          $b_k \leftarrow b$
          if $\varrho_{N^k_{b+1}}(w, y) > max(\tilde{y}^k_b)$ then
              break loop 1
          if $\varrho_{N^k_b}(w, y) \geq min(\tilde{y}^k_b)$ then
              break loop 2
  $\lambda \leftarrow \varrho_{N^k_{b_k}}(y)$
  for $i \in \{1..|y|\}$ do
      $x_i \leftarrow \max(y_i - w_i\lambda, 0)$
```

Filtering The advantage of algorithm $w\text{-}pivot^F$ is to discard values that are known to be already dominated by an incrementally updated lower bound of λ. In $w\text{-}bucket^F$, we can easily use the exact same lower bound, by keeping its process in parallel of the processing of the buckets. Moreover, thanks to the bucketization, we can also have another lower bound. When we are processing bucket b at iteration k, then $\varrho_{N^k_{b+1}}(w, y)$ is another, pretty good lower bound of λ. Note that this value should be directly available, because it was required to process the previous bucket. Finally, the filtering consists in removing values that are lower than one of our lower bounds.

4. Experimental evaluation

In these experiments, we reproduced the experiments from [20] by defining random vectors of size varying between 10^5 and 10^7, using either uniform or Gaussian distributions. We generated 500 vectors for each experiment and ran each algorithm independently, and extracted their mean times. We show here the performances of the proposed algorithms, $w\text{-}pivot^F$, and *w-bucket* and $w\text{-}bucket^F$ against the existing algorithm *w-sort* [9] Algorithm 1, implemented using an efficient quick-sort procedure. The difference between *w-bucket* and $w\text{-}bucket^F$ is the use or not of the filtering improvement. All the algorithms are implemented in C, and run on a I7 3rd generation. All source codes are available *online*.[2]

Uniform We start our experiments with Fig. 4 and Fig. 5 which is a filtered plot containing only the $w\text{-}bucket^F$ and $w\text{-}pivot^F$ algorithms. This figure shows that when a uniform random distribution is used for vector y, the time needed for projecting the vector grows linearly for all methods as a function of the vector size. Moreover, the proposed algorithms seem to perform order of magnitude faster than *w-sort*, which is already a faster algorithm than the state of the art. The second plot of Fig. 4 shows the impact of the radius for the projection time. It is interesting to note that while all proposed methods outperform the sorting scheme, when the radius becomes too large, the cost of filtering become larger than the gain it can bring, because less values are discarded.

Gaussian Fig. 7 (left) shows that the time seems to grow linearly with d, the size of vector y. Moreover, the $w\text{-}bucket^F$ and $w\text{-}pivot^F$ algorithms perform better as shown in Fig. 8. Fig. 7 (right) shows that first, when the radius is small or unit, the filtering algorithms are the most efficient, but the more the radius grows, the less they are, and the classical bucket become the best one. Such a result may imply that in function of the radius size, one should choose to use the filtering or not in the *w-bucket* algorithms. Fig. 6 shows the different results for the filtered algorithms only, since their running times are order of magnitude faster. As we can see, they behave similarly in most of the cases, which may imply that the larger cuts in the search are induced by the filtering scheme, rather than the splitting scheme.

Discussion Differences in terms of running time between $w\text{-}bucket^F$ and $w\text{-}pivot^F$ are relatively small in practice. From a theoretical point of view, both of these algorithms take advantage of the filtering. We made the choice of designing these two algorithms because they represent some of the current best state of the art algorithms for the ℓ_1 ball. From our point of view, only the $w\text{-}bucket^F$ algorithm should be implemented because of its time efficiency and its linear worst-case complexity compared to the quadratic worst-case complexity of $w\text{-}pivot^F$.

[2] https://github.com/memo-p/weighted_projection.

5. Variables selection

In signal reconstruction and variables selection, the non-smooth ℓ_p $(0 < p < 1)$ regularization has proven efficient in finding sparse solutions. Consider the problem of minimizing a quadratic reconstruction error subject to the ℓ_p regularization, $\|x\|_p = (\sum_{i=1}^d |x_i|^p)^{\frac{1}{p}}$.

$$\underset{x\in\mathbb{R}^d}{\text{minimize}} \quad \frac{1}{2}\|Ax-b\|_2^2 + \lambda\|x\|_p \tag{25}$$

where $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^m$, and $\lambda \in_{\geq 0}$ is the penalty parameter.

A popular method to solve this problem is to use the iteratively reweighted ℓ_1 minimization (IRL1) [14,24–26]. An application of IRL1 to problem (25) can be

$$x^{k+1} \in \underset{x\in\mathbb{R}^d}{\arg\min} \quad \frac{1}{2}\|Ax-b\|_2^2 \quad \text{subject to} \quad \|W^k x\|_1 < r^k \tag{26}$$

with the weight $W^k = \text{diag}(w^k)$, and w^k is defined by the previous iterates by

$$w_i^k = \frac{1}{(|x_i^k| + \epsilon)^{1-p}}, \qquad i = 1, \dots, m. \tag{27}$$

where $\epsilon \in \mathbb{R}_{>0}$.

Using the weighted ℓ_1 ball projection algorithms proposed in this paper, solving (26) can be done easily. We propose to use the following algorithm, based on [22], we start with $x^0 \in \mathbb{R}^m$ randomly defined. We set $p = 1$, which is equivalent to solving the LASSO problem. Then, we smoothly decrease p, and iteratively solve an IRL1 problem such as (26) with a fixed p. A possible implementation of this algorithm is in Algorithm 5. We call this algorithm Smooth Iterative Reweighted ℓ_1 ball Projections (SIRL1).

Algorithm 5: Smooth Iterative Reweighted ℓ_1 ball Projections.

```
Data: $x^0 \in \mathbb{R}^m$, $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^m$
$p \leftarrow 1$
while $p > 0$ do
    while IRLS-p hasn't converged do
        $w_i^k = \frac{1}{(|x_i^k|+\epsilon)^{1-p}}, \quad i = 1, \dots, n.$
        $x^{k+1} \in \underset{x\in\mathbb{R}^d}{\text{minimize}} \ \frac{1}{2}\|Ax-b\|_2^2$
                     subject to $\|W^k x\|_1 < r^k$
    Decrease smoothly $p$
```

Reconstruction results. We show in the section the reconstruction efficiency of the SIRL1, which is a direct application of having an efficient projection onto the weighted L1 ball. We reproduce here part of the experimental protocol of [14]. We compare the reconstruction error and the sparsity against the state of the art LASSO algorithm and the projection based (PC) version of [14]. In this experiment, n is the number of rows, m the number of columns and k the real number of non-zero components of the solution. First, Table 1 shows some results on the reconstruction of sparse vectors with sparsity of 5, 15, 30 non-zero values over 100 values. As we can see, the reconstruction accuracy and the sparsity is rapidly found with even a small number of different p. Moreover, because of the smoothness induced by p, even if the number of iterations is larger in (a=5, $\#p = 5$) compared to (a=5, $\#p = 3$), the running time is slower.

A look at the impact of the smoothness between the values of p is given in Table 2. We can see that less than 3 iterations over p may be too small, and that more than 5 in this example seem to be useless. But one must be careful with the smoothness, we tried to push the smoothness to 100 iterations over p, the results are in Fig. 2. As we can see, the number of iterations between $p = 0.2$ and $p = 0.8$ is high, while the L0 norm is not impacted. We were able to see this kind of results in different settings we set, with larger n, m and k. Finaly, the radius is one of the most important parameter of this algorithm, it's impact can be seen on Table 3. As it is expected, a radius smaller than the number of non-zero components has a bad impact on the reconstruction, but it is interesting to see that less iterations seem to be required to solve these problems than for larger values of the radius. Finally, the results we obtained are coherent with the results obtained in [22], which validates the inverted model we used.

Table 1
Reconstruction and Sparsity for various State of the art algorithms and two instances of SIRL1 noted $\#p = 3$ (for three iterations of p) and $\#p = 5$ (for five iterations of p). (n=100, m=256).

Algorithm	ℓ_0	ℓ_1	Reconstruction	Time (s)	# iterations	a
Lasso	5	5.00	1.00	0.0029	11	5.00
PC	5	4.86	4.82	1.1295	3880	5.00
$\#p = 3$	5	5.22	0.00	0.2542	2107	5.00
$\#p = 5$	5	5.22	0.00	0.1844	2176	5.00
Lasso	151	15.00	0.01	0.0163	159	15.00
PC	256	14.80	0.00	0.9570	307	15.00
$\#p = 3$	15	14.80	0.01	0.3580	4909	15.00
$\#p = 5$	15	14.80	0.00	0.3790	5201	15.00
Lasso	146	30.00	0.05	0.0223	240	30.00
PC	256	27.73	0.00	0.0477	352	30.00
$\#p = 3$	30	27.73	0.00	0.1563	2053	30.00
$\#p = 5$	30	27.73	0.00	0.5103	7131	30.00

Table 2
Impact of the smoothness of Q (i.e. $\#p$) on the reconstruction sparsity using $a = 15$ and $a = 5$ (n=100, m=256).

ℓ_0	ℓ_1	Reconstruction	Time (s)	# iterations	a	$\#p$
156	15.00	0.00	0.0580	655	15.00	1
15	14.76	0.15	0.0600	796	15.00	2
15	14.80	0.01	0.3429	4909	15.00	3
15	14.80	0.00	0.3770	5201	15.00	5
15	14.80	0.00	0.4386	5929	15.00	6
15	14.80	0.00	1.1664	11197	15.00	10
5	5.00	1.00	0.0050	14	5.00	1
5	5.03	0.91	0.6068	3032	5.00	2
5	5.22	0.00	0.1810	2107	5.00	3
5	5.22	0.00	0.1766	2176	5.00	5
5	5.22	0.00	0.2325	2227	5.00	6
5	5.22	0.00	0.1939	2446	5.00	10

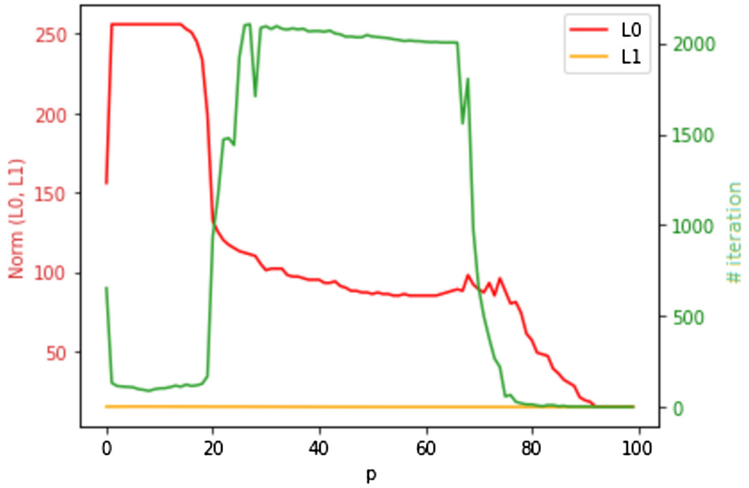

Fig. 2. Relationship between the number of iterations (*#iteration*), p value, and the norms (n=100, m=256, k=15).

6. Conclusion

Data and feature sizes are ever increasing in nowadays problems. In this paper we proposed 3 efficient projection algorithms with different complexities, including linear time, for working with very large problems.

Differences in terms of running time between $w\text{-}bucket^F$ and $w\text{-}pivot^F$ are relatively small in practice. From a theoretical point of view, both of these algorithms take advantage of the filtering.

We made the choice of designing these two algorithms because they represent some of the current best state of the art algorithms for the projection on the ℓ_1 ball.

From our point of view, only the $w\text{-}bucket^F$ algorithm should be implemented because of its time efficiency and its linear worst-case complexity compared to the quadratic worst-case complexity of the $w\text{-}pivot^F$ algorithm.

Table 3
Impact of the radius on the reconstruction sparsity. (n=100, m=256, k=15, $\#p = 4$).

ℓ_0	ℓ_0	Reconstruction	Time (s)	# iterations	Radius
8	11.58	13.18	0.0328	316	7.50
9	12.05	11.69	0.0265	247	8.62
10	13.13	8.73	0.1676	1072	9.75
11	13.33	7.61	0.0452	459	10.88
12	13.82	3.34	0.2793	2251	12.00
14	14.50	1.95	0.0318	306	13.12
15	14.69	1.02	0.0427	528	14.25
15	14.80	0.00	0.3668	5411	15.38
17	14.80	0.00	0.4542	5607	16.50
19	14.80	0.00	0.4216	5829	17.62
20	14.80	0.00	0.4015	5601	18.75
20	14.80	0.00	0.4011	5754	19.88

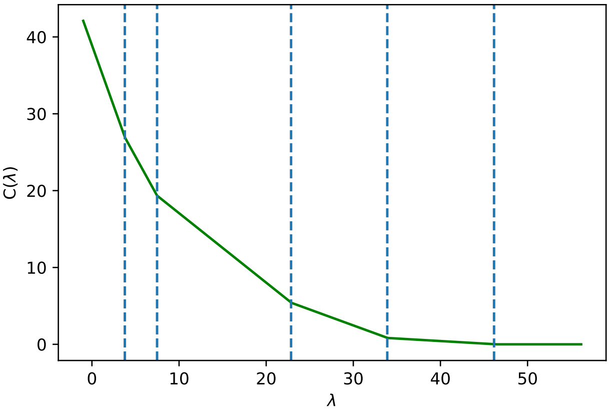

Fig. 3. Example of C function. Looking for a projection onto $a = 10$ is equivalent to looking for $C(\lambda) = 10$. Each blue vertical line is the position of a z_i value.

Experiments show that the proposed algorithms project very large non-sparse vectors in a small amount of time, such as 8 ms for vectors of size 10^7, and seem to be robust to the randomness of the vector. We showed how to directly use them in basic sparse-vector reconstruction frameworks and obtain state of the art results. Finally, we empirically proved that they should be used as a basis for projected gradient descent frameworks working with ℓ_1 balls, weighted or not.

7. Proofs

Notation Let $y = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d$ be a vector. Let a real $a > 0$ be a radius. $\Delta_{w,a}$ is the weighted simplex. Let $x = P_{\mathcal{B}_a}(y)$ be the projection of vector y onto the ℓ_1 ball $\mathcal{B}_a$ of radius a. z is a vector where each element $z_i = \frac{y_i}{w_i}$. $\lambda*$ is the threshold used to compute the projection onto the weighted ball $x_i \leftarrow \operatorname{sign}(y_i)\max(y_i - w_i\lambda^*, 0)$

Proof of existence of λ^***.** The projection onto the weighted simplex can be formulated as:

$$\begin{aligned}
\arg\min_x \quad & \frac{1}{2}\|x - y\|_2^2 \\
\text{subject to} \quad & w^\intercal x = a \\
& x_i \geq 0, \forall i \in \{0..d\}
\end{aligned}$$

Whose Lagrange dual is: $\arg\min_x \frac{1}{2}\|x - y\|_2^2 + \lambda(w^\intercal x - a) - \mu^\intercal x$ Using the Kuhn-Tucker theorem [27], we can show that necessary and sufficient conditions for x to be a optimum are $x_i - y_i = \mu_i - w_i\lambda$, $\mu_i \geq 0$, and $\mu_i x_i = 0$ and $w^\intercal x = a$. If we define x and μ to be respectively $x_i = \max(y_i - w_i\lambda, 0)$, $\mu_i = \max(w_i\lambda - y_i, 0)$ then, the conditions are respected. From this definition, it follows that $x_i \geq 0$ for all i. Let $C(\lambda) = \sum_i w_i \max(y_i - w_i\lambda, 0) = \sum_i w_i x_i$. Our goal is to find the value of λ^* such that $C(\lambda^*) = a$ (see Fig. 3).

We know that the C function cannot be negative and can reach any positive value, that C is piece-wise linear, and that the pieces of linearity are delimited by the values of $z = \{\frac{y_i}{w_i} | \forall i \in \{0..d\}\}$. Moreover, C is non-increasing.

Suppose z sorted in increasing order, thus $z_1 \leq z_2 \leq \ldots \leq z_d$, let $\uparrow()$ denote the permutation of y and w to z such that $\frac{y_{\uparrow(1)}}{w_{\uparrow(1)}} \leq \frac{y_{\uparrow(2)}}{w_{\uparrow(2)}} \leq \ldots \leq \frac{y_{\uparrow(d)}}{w_{\uparrow(d)}}$. The values at the vertices of C are

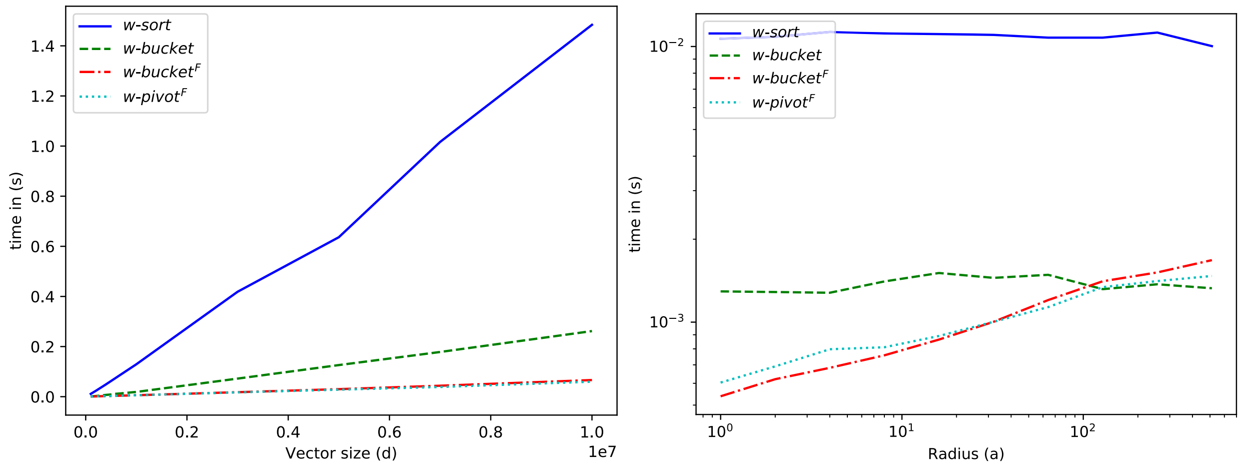

Fig. 4. Uniform distribution - Left: Projection time comparison, while the d value (size of the vector y to project) changes from 10^5 to 10^7, with $a = 4$. Right: Projection time comparison, while the radius a changes from 1 to 512, with $d = 10^5$.

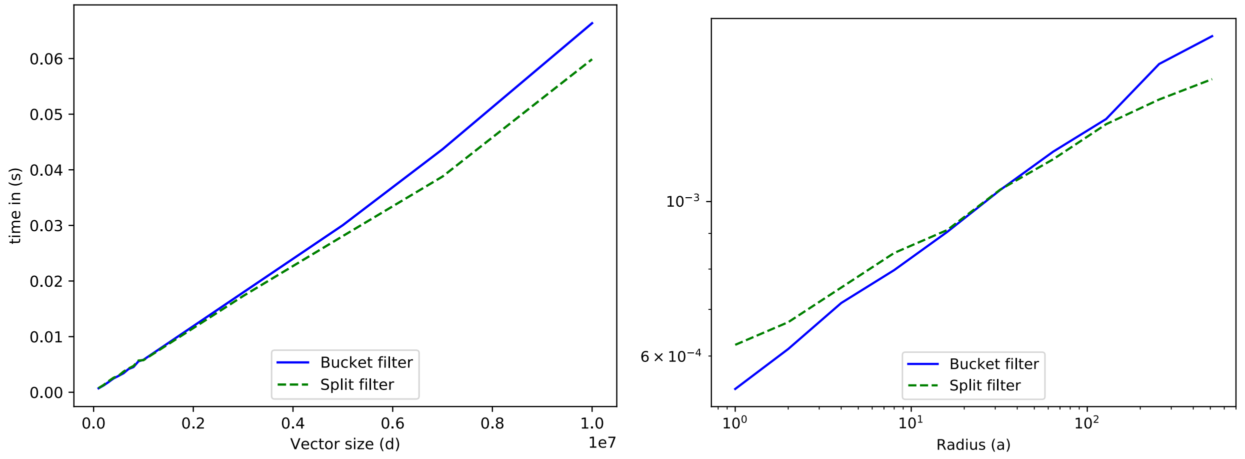

Fig. 5. Uniform distribution Zoom in - Left: Projection time comparison, while the d value (size of the vector y to project) changes from 10^5 to 10^7, with $a = 4$. Right: Projection time comparison, while the radius a changes from 1 to 512, with $d = 10^5$.

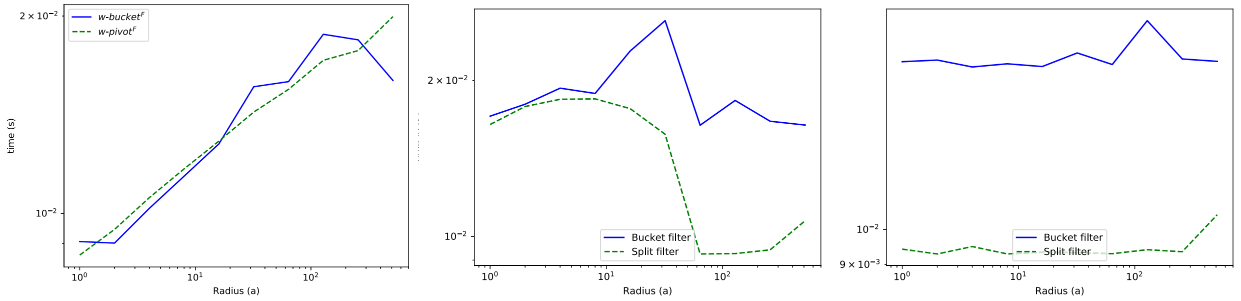

Fig. 6. Gaussian law, impact of std-dev. All three experiments have $d = 10^6$. Left: std-dev=10^{-1}. Middle: std-dev=10^{-2}. Right: std-dev=10^{-3}.

$$C(z_i) = \sum_{j=1}^{d} w_j \max(y_j - \frac{w_j y_{\uparrow(i)}}{w_{\uparrow(i)}}, 0)$$

$$C(z_i) = \sum_{j=i+1}^{d} w_{\uparrow(j)}(y_{\uparrow(j)} - \frac{w_{\uparrow(j)} y_{\uparrow(i)}}{w_{\uparrow(i)}})$$

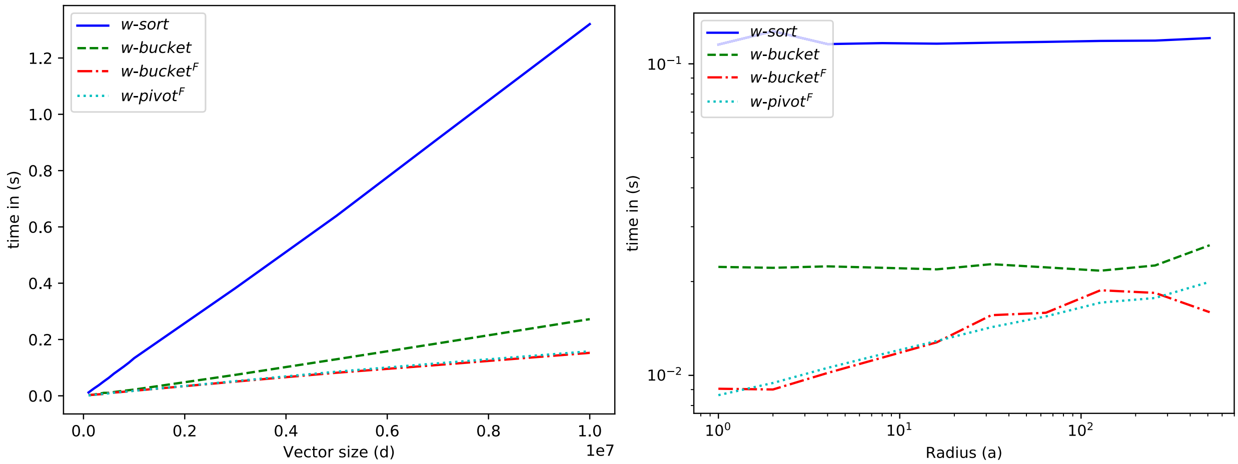

Fig. 7. Gaussian law: Left: Projection time comparison, while the d value (size of the vector y to project) changes from 10^5 to 10^7, with $a = 4$. Right: Projection time comparison, while the radius a changes from 1 to 512, with $d = 10^5$.

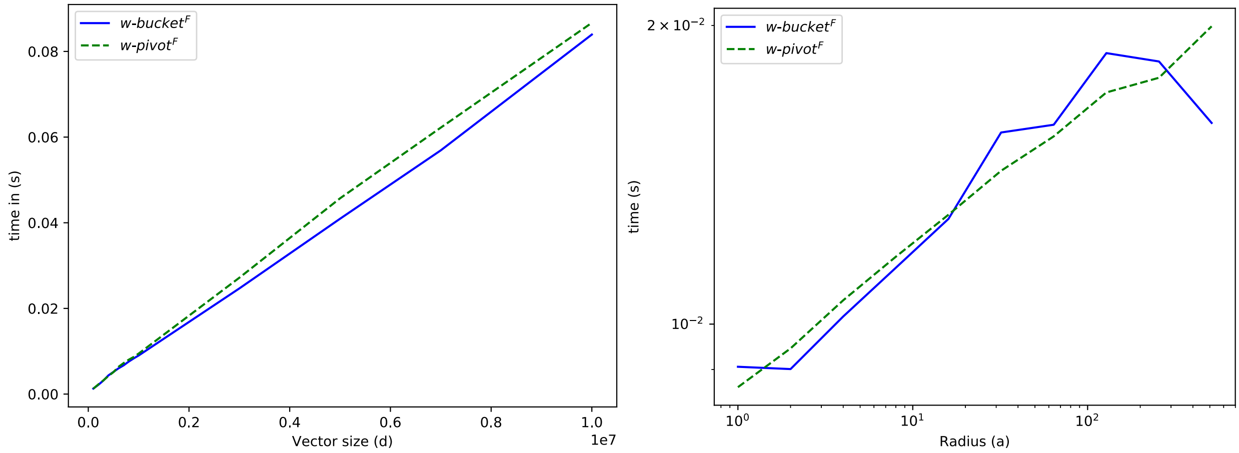

Fig. 8. Gaussian law comparison of $w\text{-}bucket^F$ and $w\text{-}pivot^F$. left: Projection time comparison, while the d value (size of the vector y to project) changes from 10^5 to 10^7, with $a = 4$. Right: Projection time comparison, while the radius a changes from 1 to 512, with $d = 10^5$.

$$C(z_i) = \sum_{j=i+1}^{d} w_{\uparrow(j)} y_{\uparrow(j)} - \sum_{j=i+1}^{d} \frac{w_{\uparrow(j)}^2 y_{\uparrow(i)}}{w_{\uparrow(i)}}$$

Since $C(\lambda^*) = a$, then for any $z_i < \lambda^*$, $x_i = 0$ and $C(z_i) > a$. Thus $x_i = 0$ for $i \in \{1..J\}$, with:

$$J := \max\left\{ j \middle| \frac{-a + \sum_{i=j+1}^{d} w_{\uparrow(i)} y_{\uparrow(i)}}{\sum_{i=j+1}^{d} w_{\uparrow(i)}^2} > z_{\uparrow(j)} \right\} \tag{28}$$

Using J, we can finally get λ^*.

$$C(\lambda^*) = \sum_{j=J+1}^{d} w_{\uparrow(j)} y_{\uparrow(j)} - \sum_{j=J+1}^{d} w_{\uparrow(j)}^2 \lambda^* = a$$

$$-\lambda^* \sum_{j=J+1}^{d} w_{\uparrow(j)}^2 = - \sum_{j=J+1}^{d} w_{\uparrow(j)} y_{\uparrow(j)} + a$$

$$\lambda^* = \frac{-a + \sum_{j=J+1}^{d} w_{\uparrow(j)} y_{\uparrow(j)}}{\sum_{j=J+1}^{d} w_{\uparrow(j)}^2}$$

This proof shows that once z is sorted, finding J and λ^* can be done in worst case linear time, as for the non-weighted version. Only one iteration gives J.

Proof that each subset is a lower-bound pivot. Consider V to be any sub-sequence of y. We can compute the following pivot:

$$p_V = \frac{-a + \sum_{i \in V} w_i y_i}{\sum_{i \in V} w_i^2}$$

Then we have:

$$a = -p_V \sum_{i \in V} w_i^2 + \sum_{i \in V} w_i y_i$$

$$a = \sum_{i \in V} w_i (y_i - w_i p_V)$$

$$a \leq \sum_{i \in \{1..d\}} w_i \max(y_i - w_i p_V, 0)$$

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Abeel, T. Helleputte, Y. Van de Peer, P. Dupont, Y. Saeys, Robust biomarker identification for cancer diagnosis with ensemble feature selection methods, Bioinformatics 26 (3) (2009) 392–398.
[2] Z. He, W. Yu, Stable feature selection for biomarker discovery, Comput. Biol. Chem. 34 (4) (2010) 215–225.
[3] D.L. Donoho, et al., Compressed sensing, IEEE Trans. Inf. Theory 52 (4) (2006) 1289–1306.
[4] S.J. Wright, R.D. Nowak, M.A. Figueiredo, Sparse reconstruction by separable approximation, IEEE Trans. Signal Process. 57 (7) (2009) 2479–2493.
[5] M.A. Figueiredo, R.D. Nowak, S.J. Wright, Gradient projection for sparse reconstruction: application to compressed sensing and other inverse problems, IEEE J. Sel. Top. Signal Process. 1 (4) (2007) 586–597.
[6] B.K. Natarajan, Sparse approximate solutions to linear systems, SIAM J. Comput. 24 (2) (1995) 227–234.
[7] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc., Ser. B, Methodol. (1996) 267–288.
[8] D.L. Donoho, M. Elad, Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization, Proc. Natl. Acad. Sci. 100 (5) (2003) 2197–2202.
[9] K. Slavakis, Y. Kopsinis, S. Theodoridis, Adaptive algorithm for sparse system identification using projections onto weighted ℓ_1 balls, in: Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on, IEEE, 2010, pp. 3742–3745.
[10] Y. Kopsinis, K. Slavakis, S. Theodoridis, Online sparse system identification and signal reconstruction using projections onto weighted ℓ_1 balls, IEEE Trans. Signal Process. 59 (3) (2010) 936–952.
[11] M. Bogdan, E. Van Den Berg, C. Sabatti, W. Su, E.J. Candès, Slope—adaptive variable selection via convex optimization, Ann. Appl. Stat. 9 (3) (2015) 1103.
[12] D. Davis, An o(n log(n)) algorithm for projecting onto the ordered weighted ℓ_1 norm ball, arXiv preprint, arXiv:1505.00870.
[13] W. Wang, An o (nlogn) projection operator for weighted ℓ_1-norm regularization with sum constraint, CoRR.
[14] E.J. Candes, M.B. Wakin, S.P. Boyd, Enhancing sparsity by reweighted ℓ_1 minimization, J. Fourier Anal. Appl. 14 (5–6) (2008) 877–905.
[15] J. Duchi, S. Shalev-Shwartz, Y. Singer, T. Chandra, Efficient projections onto the ℓ_1-ball for learning in high dimensions, in: Proceedings of the 25th International Conference on Machine Learning, ACM, 2008, pp. 272–279.
[16] C. Michelot, A finite algorithm for finding the projection of a point onto the canonical simplex of n, J. Optim. Theory Appl. 50 (1) (1986) 195–200.
[17] E. Van Den Berg, M.P. Friedlander, Probing the Pareto frontier for basis pursuit solutions, SIAM J. Sci. Comput. 31 (2) (2008) 890–912.
[18] K.C. Kiwiel, Breakpoint searching algorithms for the continuous quadratic knapsack problem, Math. Program. 112 (2) (2008) 473–491.
[19] L. Condat, Fast projection onto the simplex and the ℓ_1 ball, Math. Program., Ser. A 158 (1) (2016) 575–585.
[20] G. Perez, M. Barlaud, L. Fillatre, J.-C. Régin, A filtered bucket-clustering method for projection onto the simplex and the ℓ_1 ball, Math. Program. (2020).
[21] M. Held, P. Wolfe, H.P. Crowder, Validation of subgradient optimization, Math. Program. 6 (1) (1974) 62–88.
[22] J. Trzasko, A. Manduca, Highly undersampled magnetic resonance image reconstruction via homotopic l0-minimization, IEEE Trans. Med. Imaging 28 (1) (2008) 106–121.
[23] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, vol. 6, MIT Press, Cambridge, 2001.
[24] R. Chartrand, W. Yin, Iteratively reweighted algorithms for compressive sensing, in: 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2008, pp. 3869–3872.
[25] X. Chen, W. Zhou, Convergence of reweighted ℓ_1 minimization algorithms and unique solution of truncated lp minimization, Department of Applied Mathematics, The Hong Kong Polytechnic University.
[26] X. Chen, W. Zhou, Convergence of the reweighted ℓ_1 minimization algorithm for l2-lp minimization, Comput. Optim. Appl. 59 (1–2) (2014) 47–61.
[27] M.A. Hanson, On sufficiency of the Kuhn-Tucker conditions, J. Math. Anal. Appl. 80 (2) (1981) 545–550.