Distributed Projections onto a Simplex

Yongzheng Dai and Chen Chen

ISE, The Ohio State University, Columbus, OH, USA

April 25, 2022

Abstract

Projecting a vector onto a simplex is a well-studied problem that arises in a wide range of optimization problems. Numerous algorithms have been proposed for determining the projection; however, all but one of these algorithms are serial. We address this gap by developing a method that preprocesses the input vector by decomposing and distributing it across multiple processors for local projection. Our method is especially effective when the projection is highly sparse; which is the case, for instance, in large-scale problems with i.i.d. entries. Moreover, the method can be adapted to work with a broad range of serial algorithms from the literature. We fill in theoretical gaps in serial algorithm analysis, and develop similar results for our parallel analogues. Numerical experiments conducted on a wide range of large-scale instances demonstrate the practical effectiveness of the method.

1 Introduction

Given a vector $d \in \mathbb{R}^n$, we consider the following projection of d:

$$\text{proj}_{\Delta_b}(d) := \text{argmin}_{v \in \Delta_b} \|v - d\|_2, \tag{1}$$

where Δ_b is a scaled standard simplex parameterized by some scaling factor $b > 0$,

$$\Delta_b := \{v \in \mathbb{R}^n \mid \sum_{i=1}^{n} v_i = b, v \geq 0\}.$$

1.1 Applications

Projection onto a simplex can be leveraged as a subroutine to determine projections onto more complex polyhedra. Such projections arise in numerous settings such as: image processing, e.g. labeling [27], or multispectral unmixing [6]; portfolio optimization [10]; and machine learning [7]. As a particular example, projection onto a simplex can be used to project onto the parity polytope (see e.g. Wasson et al. [36]):

$$\mathrm{proj}_{\mathbb{PP}_n}(d) := \mathrm{argmin}_{v \in \mathbb{PP}_n} \|v - d\|_2, \tag{2}$$

where $\mathbb{PP}_n$ is a n-dimensional parity polytope:

$$\mathbb{PP}_n := \mathrm{conv}(\{v \in \{0,1\}^n \mid \sum_{i=1}^{n} v_i = 0 \ (\mathrm{mod} 2)\}).$$

Projection onto the parity polytope arises in linear programming (LP) decoding [2, 28, 37–39], which is used for signal processing.

Another example is projection onto a ℓ_1 ball:

$$\mathcal{B}_b := \{v \in \mathbb{R}^n \mid \sum_{i=1}^{n} |v_i| \leq b\}. \tag{3}$$

Duchi et al. [19] demonstrate that the solution to this problem can be easily recovered from projection onto a simplex (see Section 4 for details). Furthermore, projection onto a ℓ_1 ball can, in turn, be used as a subroutine in gradient-projection methods (see e.g. van den Berg [4]) for a variety of machine learning problems that use ℓ_1 penalty, such as: Lasso [34]; basis-pursuit denoising [4, 5, 14]; sparse representation in dictionaries [18]; variable selection [35]; and classification [1].

Finally, we note that methods for projection onto the scaled standard simplex and ℓ_1 ball can be extended to projection onto the weighted simplex and weighted ℓ_1 ball [31] (see Section 4.3). Projection onto the weighted simplex can, in turn, be used to solve the continuous quadratic knapsack problem [33]. Moreover, ℓ_p regularization can be handled by iteratively solving weighted ℓ_1 projections [11, 12, 15].

1.2 Contributions

Most of the algorithms for projection onto a simplex are serial. Indeed, to our knowledge, there is only one published parallel method for the projection problem (1), proposed by Wasson et al. [36], which parallelizes a basic sort and scan (specifically prefix sum) approach. In this paper we propose a way to decompose the projection problem across processors in order to determine (some) zero-valued entries in the original projection. The key insight to our approach is captured by Proposition 3.4: the projection of any subvector of d onto a simplex (in the corresponding space with scale factor b) will have zero-valued entries only if the full-dimension projection has corresponding zero-valued entries. Our method can be interpreted as a sparsity-exploiting distributed preprocessing method, and thus it can be adapted to parallelize a broad range serial projection algorithms. Furthermore, we develop a modest but practically significant enhancement to the parallel approach of Wasson et al. We provide thorough theoretical analyses of all aforementioned algorithms, and also conduct computational experiments. Our computational results demonstrate significant speedups with our distributed method on a standard multicore CPU over a wide range of large-scale problems. The remainder of the paper is organized as follows. Section 2 describes serial algorithms from the literature and develops new complexity results to fill in gaps in the literature. Section 3 develops parallel analogues of the aforementioned algorithms using our novel distributed method. Section 4 extends these parallelized algorithms to various applications of projection onto a simplex. Section 5 describes computational experiments. Section 6 concludes.

2 Background and Serial Algorithms

This section begins with a presentation of some fundamental results regarding projection onto a simplex, followed by analysis of serial algorithms for the problem, filling in various gaps in the literature. The final subsection, Section 2.6, provides a summary. Note that, for the purposes of average case analysis, we assume uniformly distributed inputs: $d_1, \ldots, d_n$ are i.i.d $\sim U[l, u]$.

2.1 Properties of Simplex Projection

KKT conditions yield a useful characterization of the unique optimal solution v^* to problem (1):

Proposition 2.1 (Held, Wolfe, and Crowder [22])**.** *For a vector $d \in \mathbb{R}^n$ and a scaled standard simplex $\Delta_b \in \mathbb{R}^n$, there exists a unique $\tau \in \mathbb{R}$ such that*

$$v_i^* = \max\{d_i - \tau, 0\}, \ \forall i = 1, \cdots, n, \tag{4}$$

where $v^* := \text{proj}_{\Delta_b}(d) \in \mathbb{R}^n$.

Hence, problem (1) can be reduced to a univariate search problem for the optimal *pivot* τ. Note that the nonzero (positive) entries of v^* correspond to entries where $d_i > \tau$. So for a given value $t \in \mathbb{R}$ and the index set $\mathcal{I} := \{1, ..., n\}$, we denote the *active index set* $\mathcal{I}_t \subseteq \mathcal{I}$ as the set containing all indices of *active elements* where $d_i > t$. Now consider the following function, which will be used to provide an alternate characterization of τ:

$$f(t) := \begin{cases} \frac{\sum_{i \in \mathcal{I}_t} d_i - b}{|\mathcal{I}_t|} - t, & t < \max_i\{d_i\} \\ -b, & t \geq \max_i\{d_i\} \end{cases} \tag{5}$$

Corollary 2.2. *For any $t_1, t_2 \in \mathbb{R}$ such that $t_1 < \tau < t_2$, we have*

$$f(t_1) > f(\tau) = 0 > f(t_2).$$

Proof. By definition,

$$f(t) = \frac{\sum_{i \in \mathcal{I}_t} d_i - b}{|\mathcal{I}_t|} - t,$$

$$= \frac{\sum_{i \in \mathcal{I}_t}(d_i - t) - b}{|\mathcal{I}_t|},$$
$$= \frac{\sum_{i=1}^n \max\{d_i - t, 0\} - b}{|\mathcal{I}_t|}.$$

Observe that $g(t) := \sum_{i=1}^n \max\{d_i - t, 0\} - b$ is a strictly decreasing function for $t \leq \max_i\{d_i\}$, and $g(t) = -b$ for $t \geq \max_i\{d_i\}$. Furthermore, from Proposition 2.1, we have that τ is the unique value such that $g(\tau) = 0$; moreover, since $b > 0$ then $\tau < \max_i\{d_i\}$. Thus $g(t_1) > g(\tau) = 0 > g(t_2)$, which implies $f(t_1) > 0, f(\tau) = 0$, and $f(t_2) < 0$. □ □

Thus the sign of f only changes once, and τ is its unique root. These results can be leveraged to develop search algorithms for τ, which are presented next. This corollary and the use of f is our own contribution, as we have found it a convenient organizing principle for various simplex projection algorithms from the literature.

2.2 Sort and Scan

Observe that only the greatest $|\mathcal{I}_\tau|$ terms of d are indexed in $\mathcal{I}_\tau$. Now suppose we sort d in decreasing order:

$$d_{\pi_1} \geq d_{\pi_2} \geq ... \geq d_{\pi_n}.$$

We can sequentially test these values in descending order, $f(d_{\pi_1}), f(d_{\pi_2})$, etc. to determine $|\mathcal{I}_\tau|$. In particular, from Corollary 2.2 we know there exists some $\kappa := |\mathcal{I}_\tau|$ such that $f(d_{\pi_\kappa}) < 0 \leq f(d_{\pi_{\kappa+1}})$. Thus the projection must have κ active elements, and since $f(\tau) = 0$, we have $\tau = \frac{\sum_{i=1}^\kappa d_{\pi_i} - b}{\kappa}$. We also note that, rather than recalculating f at each iteration, one can keep a running cumulative/prefix sum or *scan* of $\sum_{i=1}^j d_{\pi_i}$ as j increments.

Algorithm 1: Sort and Scan (Held, Wolfe, and Crowder [22])

```
  Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
  Output: projection $v^*$.
1 Sort $d$ as $d_{\pi_1} \geq \cdots \geq d_{\pi_n}$;
2 Set $\kappa := \max_{1 \leq j \leq n}\{j : \frac{\sum_{i=1}^j d_{\pi_i} - b}{j} < d_{\pi_j}\}$;
3 Set $\tau := \frac{\sum_{i=1}^\kappa d_{\pi_i} - b}{\kappa}$;
4 Set $v_i^* := \max\{d_i - \tau, 0\}$ for all $1 \leq i \leq n$;
5 return $v^* = (v_1^*, \cdots, v_n^*)$.
```

The bottleneck is sorting, as all other operations are linear time; for instance, `QuickSort` executes the sort with average complexity $O(n \log n)$ and worst-case $O(n^2)$, while `MergeSort` has worst-case $O(n \log n)$ (see, e.g. [3]). Moreover, non-comparison sorting methods can achieve $O(n)$ (see, e.g. [29]), albeit typically with a high constant factor as well as dependence on the bit-size of d. Indeed, the use of buckets in radix sort is shared by the linear-time Bucket Method [32] for simplex projection, described in Section 2.5.

2.3 Pivot and Partition

Sort and Scan begins by sorting all elements of d, yet only the greatest $|\mathcal{I}_\tau|$ terms are needed to calculate τ. Pivot and Partition, proposed by Duchi et al. [19], can be interpreted as a hybrid sort-and-scan that attempts to avoid sorting all elements. We present as Algorithm 2 a variant of this method approach given by Condat [16].

```
Algorithm 2: Pivot and Partition
   Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
   Output: projection $v^*$.
1  Set $\mathcal{I} := \{1, ..., n\}$, $\mathcal{I}_\tau := \emptyset$, $\mathcal{I}_p := \emptyset$;
2  while $\mathcal{I} \neq \emptyset$ do
3      Select a pivot $p \in [\min_{i\in\mathcal{I}}\{d_i\}, \max_{i\in\mathcal{I}}\{d_i\}]$;
4      Set $\mathcal{I}_p := \{i \mid d_i > p, i \in \mathcal{I}\}$;
5      if $(\sum_{i\in\mathcal{I}_p\cup\mathcal{I}_\tau} d_i - b)/(|\mathcal{I}_p| + |\mathcal{I}_\tau|) > p$ then
6          Set $\mathcal{I} := \mathcal{I}_p$;
7      else
8          Set $\mathcal{I}_\tau := \mathcal{I}_\tau \cup \mathcal{I}_p$, $\mathcal{I} := \mathcal{I} \setminus \mathcal{I}_p$;
9      end
10 end
11 Set $\tau := \frac{\sum_{i\in\mathcal{I}_\tau} d_i - b}{|\mathcal{I}_\tau|}$;
12 Set $v_i^* := \max\{d_i - \tau, 0\}$ for all $1 \leq i \leq n$;
13 return $v^* := (v_1^*, \cdots, v_n^*)$.
```

We select a pivot $p \in [\min_i\{d_i\}, \max_i\{d_i\}]$, which is intended as a candidate value for τ, and calculate $f(p)$. From Corollary 2.2, if $f(p) > 0$, then $p < \tau$ and so $\mathcal{I}_p \supset \mathcal{I}_\tau$; consequently, a new pivot is chosen in the (tighter) interval $[\min_{i\in\mathcal{I}_p}\{d_i\}, \max_{i\in\mathcal{I}_p}\{d_i\}]$, which is known to contain τ. Otherwise, if $f(p) \leq 0$, then $p \geq \tau$, and so we can find a new pivot $p \in [\min_{i\in\bar{\mathcal{I}}_p}\{d_i\}, \max_{i\in\bar{\mathcal{I}}_p}\{d_i\}]$, where $\bar{\mathcal{I}}_p := \{1, ..., n\}\backslash\mathcal{I}_p$ is the complement set. Repeatedly selecting new pivots and creating partitions in this manner results in a binary search to determine the correct active set $\mathcal{I}_\tau$, and consequently τ.

Several strategies have been proposed for selecting a pivot within a given interval. Duchi et al. [19] choose a random value in the interval, while Kiwiel [23] uses the median value. The classical approach of Michelot [30] can be interpreted as a special case that sets the initial pivot as $p^{(1)} = (\sum_{i\in\mathcal{I}} d_i - b)/|\mathcal{I}|$, and subsequently $p^{(i+1)} = f(p^{(i)}) + p^{(i)}$. This ensures that $p^{(i)} \leq \tau$ which avoids extraneous re-evaluation of sums in the **if** condition. Michelot's algorithm is presented separately as Algorithm 3.

Algorithm 3: Michelot's method

```
    Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
    Output: projection $v^*$.
1 Set $\mathcal{I}_p := \{1, ..., n\}$, $\mathcal{I} := \emptyset$;
2 do
3     Set $\mathcal{I} := \mathcal{I}_p$;
4     Set $p := (\sum_{i\in\mathcal{I}} d_i - b)/|\mathcal{I}|$;
5     Set $\mathcal{I}_p := \{i \in \mathcal{I} \mid d_i > p\}$;
6 while $|\mathcal{I}| > |\mathcal{I}_p|$;
7 Set $v_i^* := \max\{d_i - p, 0\}$ for all $1 \le i \le n$;
8 return $v^* := (v_1^*, \cdots, v_n^*)$.
```

Condat [16] provided worst-case runtimes for each of the pivot rules, as well as average case complexity (over the uniform distribution) for the random pivot rule (see Table 2). We fill in the gaps here and establish $O(n)$ runtimes for the median rule as well as Michelot's method. We note that the median pivot method is a linear-time algorithm, but relies on a median-of-medians subroutine [8], which has a high constant factor.

Lemma 2.3.
$$E[\frac{\sum_{i=1}^n d_i - b}{n}] \to \frac{l+u}{2} \text{ as } n \to \infty$$
with a sublinear convergence rate.

Proof. Observe that $E[d_i] = \frac{l+u}{2}$. Since $d_1, ...d_n$ are i.i.d, then we have
$$E[\frac{\sum_{i=1}^n d_i - b}{n}] = \frac{l+u}{2} - \frac{b}{n}.$$
Thus $E[\frac{\sum_{i=1}^n d_i - b}{n}]$ converges to $\frac{l+u}{2}$ at the rate of $O(\frac{1}{n})$ as n approaches infinity. □ □

Proposition 2.4. *Michelot's method has an average runtime of $O(n)$.*

Proof. Let δ_i be the number of elements that Algorithm 3 removes from the (candidate) active set $\mathcal{I}_p^i$ in iteration i, and let T be the total number of iterations. Then
$$\sum_{i=1}^T \delta_i = n - |\mathcal{I}_\tau|.$$
From Lemma 2.3 and Proposition A.3, the algorithm will (recursively) remove half the entries in expectation from $\mathcal{I}_p$ in each iteration as n approaches infinity, so (asymptotically) $E[\delta_i] = n/2^i$. Furthermore, in a given iteration j the algorithm scans $|\mathcal{I}_p^{(j)}|$ terms. So
$$E[|\mathcal{I}_p^{(j)}|] = E[n - \sum_{i=1}^{j-1} \delta_i]$$

$$
\begin{aligned}
&= n - \sum_{i=1}^{j-1} \frac{n}{2^i} \\
&= \frac{n}{2^{j-1}} \\
&= 2E[\delta_j];
\end{aligned}
$$

thus, $E[\sum_{j=1}^{T} |\mathcal{I}_p^{(j)}|] = 2E[\sum_{j=1}^{T} \delta_j] = 2(n - E[|\mathcal{I}_\tau|])$; and so $O(n)$ operations are used for scanning. All other operations, i.e. assigning $\mathcal{I}$ and $\mathcal{I}_p$, are within a constant factor of the scanning operations. □ □

The same argument holds for the median pivot rule, as half the terms are guaranteed to be removed each iteration, and the operations per loop are within a constant factor of Michelot's; we omit a formal proof of its $O(n)$ average runtime for brevity.

2.4 Condat's Method

Condat's method [16], presented as Algorithm 5, can be seen as a modification of Michelot's method in two ways. First, Condat replaces the initial scan with a Filter to find an initial pivot, presented as Algorithm 4. Lemma 2.5 shows that the Filter provides a greater (or equal) initial starting pivot compared to a scan; since Michelot approaches τ from below, this results in fewer iterations (see proof of Proposition 2.9). Second, Condat's method dynamically updates the pivot value whenever an inactive entry is removed from $\mathcal{I}_p$; in contrast, Michelot's method updates the pivot every iteration by summing over all entries.

Condat [16] supplies a worst-case complexity of $O(n^2)$. We supplement this with average-case analysis. The following technical lemmas are needed to develop the final result.

Lemma 2.5. *Filter provides a pivot p such that $\tau \geq p \geq \frac{\sum_{i \in \mathcal{I}} d_i - b}{|\mathcal{I}|}$.*

Proof. The upper bound $p \leq \tau$ is given by construction of p (see Condat [16, Section 3, Paragraph 2]).

We can establish the lower bound on p by considering the sequence $p^{(1)} \leq \ldots \leq p^{(n)} \in \mathbb{R}$, which represents the initial as well as subsequent (intermediate) values of p from the first outer for-loop on line 2 of presented as Algorithm 4, and their corresponding index sets $\mathcal{I}_p^{(1)} \subseteq \ldots \subseteq \mathcal{I}_p^{(n)}$. Filter initializes with $p^{(1)} := d_1 - b$ and $\mathcal{I}_p^{(1)} := \{1\}$. For $i = 2, \ldots, n$, if $d_i > p^{(i-1)}$,

$$
p^{(i)} := p^{(i-1)} + (d_i - p^{(i-1)})/(|\mathcal{I}_p^{(i-1)}| + 1),\ \mathcal{I}_p^{(i)} := \mathcal{I}_p^{(i-1)} \cup \{i\};
$$

otherwise $p^{(i)} := p^{(i-1)}$, and $\mathcal{I}_p^{(i)} := \mathcal{I}_p^{(i-1)}$. Then it can be shown that $p^{(i)} = (\sum_{j \in \mathcal{I}_p^{(i)}} d_j - b)/|\mathcal{I}_p^{(i)}|$ (see Condat [16, Section 4, Paragraph 2]), and $p \geq p^{(n)}$

Algorithm 4: Filter

```
    Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
    Output: $\mathcal{I}_t$.
1  Set $\mathcal{I}_p := \{1\}$, $\mathcal{I}_w := \emptyset$, $p := d_1 - b$;
2  for $i = 2 : n$ do
3      if $d_i > p$ then
4          Set $p := p + \frac{d_i - p}{|\mathcal{I}_p| + 1}$;
5          if $p > d_i - b$ then
6              Set $\mathcal{I}_p := \mathcal{I}_p \cup \{i\}$;
7          else
8              Set $\mathcal{I}_w := \mathcal{I}_w \cup \mathcal{I}_p$, $\mathcal{I}_p := \{i\}$, $p := d_i - b$;
9          end
10     end
11 end
12 for $i \in \mathcal{I}_w$ do
13     if $d_i > p$ then
14         Set $\mathcal{I}_p := \mathcal{I}_p \cup \{i\}$, $p := p + \frac{d_i - p}{|\mathcal{I}_p|}$;
15     end
16 end
17 return $\mathcal{I}_p$.
```

(see Condat [16, Section 3, Paragraph 5]). Now in terms of $p^{(n)}$ we may write

$$\begin{aligned}\frac{\sum_{i\in\mathcal{I}} d_i - b}{|\mathcal{I}|} =& \frac{\sum_{i\in\mathcal{I}_p^{(n)}} d_i - b + \sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}} d_i}{|\mathcal{I}|} \\ =& \frac{(\sum_{i\in\mathcal{I}_p^{(n)}} d_i - b)|\mathcal{I}_p^{(n)}|}{|\mathcal{I}||\mathcal{I}_p^{(n)}|} + \frac{\sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}} d_i}{|\mathcal{I}|} \\ =& \frac{|\mathcal{I}_p^{(n)}|}{|\mathcal{I}|} p^{(n)} + \frac{\sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}} d_i}{|\mathcal{I}|} \\ =& p^{(n)} + \frac{\sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}} d_i - (|\mathcal{I}| - |\mathcal{I}_p^{(n)}|)p^{(n)}}{|\mathcal{I}|} \\ =& p^{(n)} + \frac{\sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}} (d_i - p^{(n)})}{|\mathcal{I}|}.\end{aligned}$$

For any $i \in \mathcal{I}\setminus\mathcal{I}_p^{(n)}$, since $\mathcal{I}_p^{(i)} \subseteq \mathcal{I}_p^{(n)}$ then $i \notin \mathcal{I}_p^{(i)}$. By construction of $p^{(i-1)}$ and $\mathcal{I}_p^{(i)}$, we have $d_i \leq p^{(i-1)} \leq p^{(n)}$. Thus, $\sum_{i\in\mathcal{I}\setminus\mathcal{I}_p^{(n)}}(d_i - p^{(n)}) \leq 0$, and $p^{(n)} \geq (\sum_{i\in\mathcal{I}} d_i - b)/|\mathcal{I}|$. So $p \geq p^{(n)} \geq (\sum_{i\in\mathcal{I}} d_i - b)/|\mathcal{I}|$. □ □

As $(\sum_{i\in\mathcal{I}} d_i - b)/|\mathcal{I}|$ is the initial pivot value used by Michelot's method (see Algorithm 3 line 4), Lemma 2.5 implies that Filter provides a (weakly)

Algorithm 5: Condat's method

```
   Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
   Output: projection $v^*$.
1  Set $\mathcal{I}_p :=$ Filter$(d, b)$, $p := \frac{\sum_{i\in\mathcal{I}_p} d_i - b}{|\mathcal{I}_p|}$, $\mathcal{I} := \emptyset$;
2  do
3      Set $\mathcal{I} := \mathcal{I}_p$;
4      for $i \in \mathcal{I} : d_i \le p$ do
5          Set $\mathcal{I}_p := \mathcal{I}_p \backslash \{i\}$;
6          Set $p := p + \frac{p - d_i}{|\mathcal{I}_p|}$;
7      end
8  while $|\mathcal{I}| > |\mathcal{I}_p|$;
9  Set $v_i^* := \max\{d_i - p, 0\}$ for all $1 \le i \le n$;
10 return $v^* = (v_1^*, \cdots, v_n^*)$.
```

better starting pivot. Now we introduce some notation in order to compare subsequent iterations of Condat's method with iterations of Michelot's method. Let $t_C \le n$ and $t_M \le n$ be the total number of iterations taken by Condat's method and Michelot's method (respectively) on a given instance. Let $\mathcal{I}_0^C, ..., \mathcal{I}_n^C$ be the active index sets per iteration for Condat's method with corresponding pivots $p_0^C, ..., p_n^C$. Likewise, we denote the index sets and pivots of Michelot's method as $\mathcal{I}_0^M, ..., \mathcal{I}_n^M$ and $p_0^M, ..., p_n^M$, respectively. If $t_C < n$ then we set $p_{t_C}^C = p_{t_C+1}^C = ... = p_n^C = \tau$ and $\mathcal{I}_{t_C}^C = \mathcal{I}_{t_C+1}^C = ... = \mathcal{I}_n^C = \mathcal{I}_\tau$; likewise for Michelot's algorithm.

Lemma 2.6. *$\mathcal{I}_i^C \subseteq \mathcal{I}_i^M$, and $p_i^C \ge p_i^M$ for $i = 0, ..., n$.*

Proof. We will prove this by induction. For the base case, $\mathcal{I}_0^C$ is obtained by Filter. So $\mathcal{I}_0^C \subseteq \mathcal{I} = \mathcal{I}_0^M$. Moreover, from Lemma 2.5, $p_0^C \ge p_0^M$.

Now for any iteration $i \ge 1$, suppose $\mathcal{I}_i^C \subseteq \mathcal{I}_i^M$, and $p_i^C \ge p_i^M$. From line 5 in Algorithm 3, $\mathcal{I}_{i+1}^M := \{j \in \mathcal{I}_i^M : d_j > p_i^M\}$. From Condat [16, Section 3, Paragraph 3], Condat's method uses a dynamic pivot between p_i^C to p_{i+1}^C to remove inactive entries that would otherwise remain in Michelot's method. Therefore, $\mathcal{I}_{i+1}^C \subseteq \{j \in \mathcal{I}_i^C : d_j > p_i^C\} \subseteq \mathcal{I}_{i+1}^M$, and moreover for any $j \in \mathcal{I}_{i+1}^M \backslash \mathcal{I}_{i+1}^C$, we have that $d_j \le p_{i+1}^C$. Now observe that

$$\begin{aligned}
&p_{i+1}^C - p_{i+1}^M \\
=&\frac{\sum_{j\in\mathcal{I}_{i+1}^C} d_j - b}{|\mathcal{I}_{i+1}^C|} - \frac{\sum_{j\in\mathcal{I}_{i+1}^M} d_j - b}{|\mathcal{I}_{i+1}^M|} \\
=&\frac{\sum_{j\in\mathcal{I}_{i+1}^C} d_j - b}{|\mathcal{I}_{i+1}^C|} - \frac{\sum_{j\in\mathcal{I}_{i+1}^C} d_j + \sum_{j\in\mathcal{I}_{i+1}^M\backslash\mathcal{I}_{i+1}^C} d_j - b}{|\mathcal{I}_{i+1}^C| + |\mathcal{I}_{i+1}^M\backslash\mathcal{I}_{i+1}^C|} \\
=&\frac{\sum_{j\in\mathcal{I}_{i+1}^M\backslash\mathcal{I}_{i+1}^C} (p_{i+1}^C - d_j)}{|\mathcal{I}_{i+1}^C| + |\mathcal{I}_{i+1}^M\backslash\mathcal{I}_{i+1}^C|} \ge 0,
\end{aligned}$$

and so $p_{i+1}^C \geq p_{i+1}^M$. □ □

Corollary 2.7. $\sum_{i=1}^{t_C} |\mathcal{I}_i^C| \leq \sum_{i=1}^{t_M} |\mathcal{I}_i^M|$.

Proof. Observe that both algorithms remove elements (without replacement) from their candidate active sets $\mathcal{I}_i^C, \mathcal{I}_i^M$ at every iteration; moreover, they terminate with the pivot value τ and so $\mathcal{I}_{t_C}^C = \mathcal{I}_{t_M}^M = \mathcal{I}_\tau$. So, together with Lemma 2.6, we have for $i = 0, ..., n$ that $\mathcal{I}_\tau \subseteq \mathcal{I}_i^C \subseteq \mathcal{I}_i^M$. So $\mathcal{I}_{t_M}^M = \mathcal{I}_\tau$ implies $\mathcal{I}_{t_M}^C = \mathcal{I}_\tau$, and so $t_C \leq t_M$. Therefore $\sum_{i=1}^{t_C} |\mathcal{I}_i^C| \leq \sum_{i=1}^{t_M} |\mathcal{I}_i^M|$. □ □

Lemma 2.8. *The worst-case runtime of Filter is $O(n)$.*

Proof. Since $\mathcal{I}_w \subseteq \mathcal{I}$ at any iteration, Filter will scan at most $2|\mathcal{I}|$ entries; including $O(1)$ operations to update p. □

Proposition 2.9. *Condat's method has an average runtime of $O(n)$.*

Proof. Filter takes $O(n)$ operations from Lemma 2.8. From Corollary 2.7, the total operations spent on scanning in Condat's method is less than (or equal to) the average $O(n)$ operations for Michelot's method (established in Proposition 2.4); hence Condat's average runtime is $O(n)$. □ □

2.5 Bucket Method

Pivot and Partition selects one pivot in each iteration to partition d and applies this recursively in order to create sub-partitions in the manner of a binary search. The Bucket Method, developed by Perez et al. [32], can be interpreted as a modification that uses multiple pivots and partitions (buckets) per iteration.

The algorithm, presented as Algorithm 6 is initialized with tuning parameters T, the maximum number of iterations, and c, the number of buckets with which to subdivide the data. In each iteration the algorithm partitions the problem into the buckets $\mathcal{I}_j$ with the inner for loop of line 4, and then calculates corresponding pivot values in the inner for loop of line 10.

The tuning parameters can be determined as follows. Suppose we want the algorithm to find a (final) pivot $\bar{\tau}$ within some absolute numerical tolerance D of the true pivot τ, i.e. such that $|\bar{\tau} - \tau| \leq D$. This can be ensured (see [32]) by setting

$$T = \log_c \frac{R}{D},$$

where $R := \max_{i \in \mathcal{I}}\{d_i\} - \min_{i \in \mathcal{I}}\{d_i\}$ denotes the range of d. Perez et al. [32] prove the worst-case complexity is $O((n + c) \log_c(R/D))$.

Proposition 2.10. *The Bucket method has an average runtime of $O(cn)$.*

Proof. Let $\mathcal{I}^{(t)}$ denote the index set $\mathcal{I}$ at the start of iteration t in the outer **for** loop (line 2), and $\mathcal{I}_j^{(t)}$ denote the index set of the jth bucket, $\mathcal{I}_j$, at the end of the first inner **for** loop (line 4).

```
Algorithm 6: Bucket method
Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, bucket number $c$,
       maximum number of iterations $T$.
Output: projection $v^*$.
1  Set $\mathcal{I} = \{1, ..., n\}$, $\mathcal{I}_\tau = \emptyset$;
2  for $t = 1 : T$ do
3      Set $\mathcal{I}_1, ..., \mathcal{I}_c := \emptyset$;
4      for $j = 1 : c$ do
5          Set $p_j := (\max_{i\in\mathcal{I}}\{d_i\} - \min_{i\in\mathcal{I}}\{d_i\}) \cdot (c-j)/c + \min_{i\in\mathcal{I}}\{d_i\}$;
6          for $i \in \mathcal{I} : d_i \geq p_j$ do
7              Set $\mathcal{I}_j := \mathcal{I}_j \cup \{i\}$, $\mathcal{I} := \mathcal{I}\backslash\{i\}$
8          end
9      end
10     for $j = 1 : c$ do
11         Set $p = \frac{\sum_{i\in\mathcal{I}_\tau} d_i + \sum_{i\in\mathcal{I}_j} d_i - b}{|\mathcal{I}_\tau| + |\mathcal{I}_j|}$;
12         if $p \geq p_j$ then
13             Set $\mathcal{I} := \mathcal{I}_j$;
14             Break the inner loop;
15         else if $j < c$ & $p > \max_{i\in\mathcal{I}_{j+1}}\{d_i\}$ then
16             Set $\mathcal{I}_\tau := \mathcal{I}_\tau \cup \mathcal{I}_j$;
17             Break the outer loop;
18         else
19             $\mathcal{I}_\tau := \mathcal{I}_\tau \cup \mathcal{I}_j$;
20         end
21     end
22 end
23 Set $\tau := \frac{\sum_{i\in\mathcal{I}_\tau} d_i - b}{|\mathcal{I}_\tau|}$;
24 Set $v_i^* := \max\{d_i - \tau, 0\}$ for all $1 \leq i \leq n$;
25 return $v^* := (v_1^*, \cdots, v_n^*)$.
```

For a given outer **for** loop iteration t (line 2), the first inner **for** loop (line 4) uses $O(c|\mathcal{I}^{(t)}|)$ operations. Note that the max and min on line 5 can be reused in each iteration, and the nested **for** loop on line 6 has $|\mathcal{I}^{(t)}|$ iterations. The second inner **for** loop (line 10) also uses $O(c|\mathcal{I}^{(t)}|)$ operations. In line 11, the first sum $\sum_{i\in\mathcal{I}_\tau} d_i$ can be updated dynamically (in the manner of a scan) as a cumulative sum as τ is updated in line 16 or 19, thus requiring a constant number of operations per iteration j. The second sum $\sum_{i\in\mathcal{I}_j} d_i$ is bounded above by $O(|\mathcal{I}^{(t)}|)$ since $\mathcal{I}_j \subseteq \mathcal{I}^{(t)}$. Thus each iteration j of the outer **for** loop uses $O(c|\mathcal{I}^{(t)}|)$ operations.

Since $d_1, ..., d_n$ are i.i.d $\sim U[l, u]$, then from Proposition A.3, the terms from each sub-partition are also i.i.d uniform. So for any $t = 1, ..., T$ and $j = 1, ..., c$, $E[|\mathcal{I}_j^{(t)}|] = E[|\mathcal{I}^{(t)}|]/c$. From line 13, $E[|\mathcal{I}^{(t+1)}|] = E[|\mathcal{I}^{(t)}|]/c$. Since

$E[|\mathcal{I}^{(1)}|] = n$ then $E[|\mathcal{I}^{(t)}|] = n/c^{t-1}$; thus

$$E[\sum_{t=1}^{T} |\mathcal{I}^{(t)}|] = \sum_{t=1}^{\log_c(R/D)} \frac{n}{c^{t-1}} = \frac{c}{c-1} n(1 - \frac{D}{R}).$$

Therefore, $E[\sum_{t=1}^{T} c \cdot |\mathcal{I}^{(t)}|] \in O(cn)$. □ □

2.6 Summary of Results

Pivot Rule	Worst Case	Average Case
(Quick)Sort and Scan	$O(n^2)$	$O(n)$
Michelot's method	$O(n^2)$	$\mathbf{O(n)}$
Pivot and Partition (Median)	$O(n)$	$\mathbf{O(n)}$
Pivot and Partition (Random)	$O(n^2)$	$O(n)$
Condat's method	$O(n^2)$	$\mathbf{O(n)}$
Bucket method	$O(cn)$	$\mathbf{O(cn)}$

Table 1: Complexity results for serial algorithms for projection onto a simplex. New results are bolded.

Table 1 shows that all presented algorithms attain $O(n)$ performance on average given uniformly i.i.d. inputs. The methods are ordered by publication date, starting from the oldest result. As described in Section 2.2, Sort and Scan can be implemented with non-comparison sorting to achieve $O(n)$ worst-case performance. However, as with the linear-time median pivot rule, and the Bucket Method, there are tradeoffs: increased memory, overhead, dependence on factors such as input bit-size, etc. Both sorting and scanning are (separately) well-studied in parallel algorithm design, so the Sort and Scan idea lends itself to a natural decomposition for parallelism (discussed in Section 3.1). The other methods integrate sorting and scanning in each iterate, and it is no longer clear how best to exploit parallelism. We develop in the next section a distributed preprocessing scheme that can greatly reduce candidate index set if the projection is sparse.

3 Parallel Algorithms

In Section 3.1 we consider the parallel method proposed by Wasson et al. [36] and propose a modification. In Section 3.2 we develop a novel distributed scheme that can be used to preprocess the input vector d and remove some elements of the projection v^* that should have the value of zero. The remainder of this section analyzes how our method can be used to enhance Pivot and Partition, as well as Condat's method via parallelization of the Filter method. Results are summarized in Section 3.5.

3.1 Parallel Sort and Parallel Scan

Wasson et al. [36] parallelize Sort and Scan in a natural way: first applying a parallel merge sort (see e.g. [17, p. 797]) and then a parallel scan [25] on the input vector. However, their scan calculates $\sum_{i=1}^{j} d_{\pi_i}$ for all $j \in \mathcal{I}$, but only $\sum_{i=1}^{\kappa} d_{\pi_\kappa}$ is needed to calculate τ. We modify the algorithm accordingly, presented as Algorithm 7: checks are added (lines 7 and 14) in the for-loops to allow for possible early termination of scans. As we are adding constant operations per loop, Algorithm 7 has the same complexity as the original Parallel Sort and Scan. We combine this with parallel mergesort in Algorithm 8 and empirically benchmark this method with the original (parallel) version in Section 5.

Algorithm 7: Parallel Partial Scan

```
   Input: sorted vector $d_{\pi_1}, \cdots, d_{\pi_n}$, scaling factor $b$
   Output: $\tau$
 1 Set $T := \lceil \log_2 n \rceil$, $s[1], ..., s[n] = d_{\pi_1}, ..., d_{\pi_n}$;
 2 for $j = 1 : T$ do
 3     for $i = 2^j : 2^j : \min(n, 2^T)$ do Parallel
 4         Set $s[i] := s[i] + s[i - 2^{j-1}]$;
 5     end
 6     Set $\kappa := \min(n, 2^j)$;
 7     if $\frac{s[\kappa]-a}{\kappa} \geq d_{\pi_\kappa}$ then
 8         break loop;
 9     end
10 end
11 Set $p := 2^{j-1}$;
12 for $i = j - 1 : -1 : 1$ do
13     Set $\kappa := \min(p + 2^{i-1}, n)$, $s[\kappa] := s[\kappa] + s[p]$;
14     if $\frac{s[\kappa]-a}{\kappa} < d_{\pi_\kappa}$ then
15         break loop;
16     end
17 end
18 Set $\tau := \frac{s[\kappa]-b}{\kappa}$;
19 return $\tau$.
```

Algorithm 8: Parallel Mergesort and Partial Scan

```
   Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$.
   Output: projection $v^*$.
 1 Parallel mergesort $d$ so that $d_{\pi_1} \geq \cdots \geq d_{\pi_n}$;
 2 Set $\tau = \texttt{PPScan}(\{d_{\pi_i}\}_{1 \leq i \leq n}, b)$;
 3 Parallel set $v_i^* := \max\{d_i - \tau, 0\}$ for all $1 \leq i \leq n$;
 4 return $v^* = (v_1^*, \cdots, v_n^*)$.
```

3.2 Sparsity-Exploiting Distributed Projections

Our main idea is motivated by the following two theorems that establish that projections with i.i.d. inputs become increasingly sparse as the problem size n increases. The first theorem considers i.i.d. uniform inputs.

Theorem 3.1. $E[|\mathcal{I}_\tau|] < \sqrt{\frac{2b(n+1)}{u-l} + \frac{1}{4}} + \frac{1}{2}$.

Proof. Sort d such that $d_{\pi_1} \geq d_{\pi_2} \geq ... \geq d_{\pi_n}$. Thus for a given order statistic, (see e.g. [21, p. 63]),

$$E[d_{\pi_i}] = u - \frac{i}{n+1}(u-l).$$

Define $N := |\mathcal{I}_\tau|$ for ease of presentation. From Corollary 2.2,

$$\frac{\sum_{i=1}^{N} d_{\pi_i} - b}{N} = \tau;$$

and, together with $\tau < d_{\pi_N}$ (by definition of $\mathcal{I}_t$), we have

$$\sum_{i=1}^{N} d_{\pi_i} - b < N \cdot d_{\pi_N},$$

$$\implies E[\sum_{i=1}^{N} d_{\pi_i} - b] < E[N \cdot d_{\pi_N}]. \tag{6}$$

Furthermore, from the Law of Total Expectation,

$$E[N \cdot d_{\pi_N}] = uE[N] - E[N^2]\frac{u-l}{n+1},$$

$$E[\sum_{i=1}^{N} d_{\pi_i} - b] = uE[N] - E[N(N+1)]\frac{u-l}{2(n+1)} - b.$$

Substituting into (6) yields

$$E[N^2] - E[N] < \frac{2b(n+1)}{n-l},$$

and since $E^2[N] \leq E[N^2]$, then

$$E^2[N] - E[N] \leq E[N^2] - E[N] < \frac{2b(n+1)}{n-l},$$

$$\implies E[|\mathcal{I}_\tau|] = E[N] < \sqrt{\frac{2b(n+1)}{u-l} + \frac{1}{4}} + \frac{1}{2}, \tag{7}$$

which is in $O(\sqrt{n})$. □ □

The second theorem considers arbitrary input distributions. The following lemma is needed to prove it.

Lemma 3.2. *Suppose* $d_1, ..., d_n$ *are* i.i.d. *from an arbitrary distribution* X*, with PDF* f_X *and CDF* F_X*. Let* $\epsilon > 0$ *be some positive number, and* $t \in \mathbb{R}$ *be such that* $1 - F_X(t) = \epsilon$*. Then i)* $|\mathcal{I}_t| \to \infty$ *as* $n \to \infty$ *and ii)* $P(\frac{|\mathcal{I}_t|}{n} \leq \epsilon) = 1$ *as* $n \to \infty$*.*

Proof. For $i = 1, ..., n$, define the indicator variable

$$\delta_i := \begin{cases} 1, & \text{if } d_i > t \\ 0, & \text{otherwise} \end{cases}$$

and let $S_n := \sum_{i=1}^{n} \delta_i$. So $S_n = |\mathcal{I}_t|$, and we can show that it is binomially distributed.

Claim. $S_n \sim B(n, \epsilon)$.

Proof: Observe that $P(\delta_i = 1) = P(d_i > t) = \epsilon$, and $P(\delta_i = 0) = P(d_i \leq t) = 1 - \epsilon$; thus $\delta_i \sim \text{Bernoulli}(\epsilon)$. Moreover, $d_1, ..., d_n$ are independent, and so $\delta_1, ..., \delta_n$ are i.i.d. Bernoulli(ϵ); consequently, $S_n := \sum_{i=1}^{n} \delta_i \sim B(n, \epsilon)$. ■

So, as $n \to \infty$, we can apply the Central Limit Theorem (see e.g. [20]):

$$S_n^* := \frac{S_n - n\epsilon}{\sqrt{n\epsilon(1-\epsilon)}} \sim N(0, 1). \tag{8}$$

Denote $\Phi(q) = \int_{-\infty}^{q} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx$ to be the CDF of the standard normal distribution. Consider (for any $q \in \mathbb{R}$) the right-tail probability

$$P(\frac{S_n - n\epsilon}{\sqrt{n\epsilon(1-\epsilon)}} \geq -q) = 1 - \Phi(-q) = \Phi(q), \tag{9}$$

$$P(S_n \geq n\epsilon - q\sqrt{n\epsilon(1-\epsilon)}) = \Phi(q),$$

$$\implies P(|\mathcal{I}_t| \geq \lfloor -q\sqrt{n\epsilon(1-\epsilon)} + n\epsilon \rfloor) \geq \Phi(q), \tag{10}$$

where $\lfloor . \rfloor$ is the floor function.

Setting $q = \sqrt{2 \log n}$ yields

$$P(|\mathcal{I}_t| \geq \lfloor -\sqrt{2(n \log n)\epsilon(1-\epsilon)} + n\epsilon \rfloor) \geq \Phi(\sqrt{2 \log n}).$$

Consider the right-hand side, $\lim_{n\to\infty} \Phi(\sqrt{2 \log n})$. Since Φ is a CDF, it is monotonically increasing, continuous, and converges to 1; thus $\lim_{n\to\infty} \Phi(\sqrt{2 \log n}) = 1$. So as n approaches infinity,

$$P(|\mathcal{I}_t| \geq \lfloor -\sqrt{2(n \log n)\epsilon(1-\epsilon)} + n\epsilon \rfloor) = 1,$$

which establishes condition (i).

Now consider the left-tail probability,

$$P(\frac{S_n - n\epsilon}{\sqrt{n\epsilon(1-\epsilon)}} \leq q) = \Phi(q)$$

$$\implies P(|\mathcal{I}_t| \leq \lceil q\sqrt{n\epsilon(1-\epsilon)} + n\epsilon\rceil) \geq \Phi(q).$$

Again setting $q = \sqrt{2\log n}$, we have that $\lim_{n\to\infty}\lceil q\sqrt{n\epsilon(1-\epsilon)} + n\epsilon\rceil/n = \epsilon$. So as n approaches infinity,

$$P(\frac{|\mathcal{I}_t|}{n} \leq \epsilon) = 1,$$

which establishes condition (ii). □ □

Theorem 3.3. *Suppose* $d_1, ..., d_n$ *are* i.i.d. *from an arbitrary distribution* X*, with PDF* f_X *and CDF* F_X*. Then, for any* $\epsilon > 0$*,* $P(\frac{|\mathcal{I}_\tau|}{n} \leq \epsilon) = 1$ *as* $n \to \infty$*.*

Proof. Let $t \in \mathbb{R}$ be such that $1 - F_X(t) = \epsilon$. We shall first establish that $P(\tau > t) = 1$ as $n \to \infty$.

From Corollary 2.2, $\tau > t \iff f(t) > 0$, and so

$$\begin{aligned}
P(\tau > t) =& P(f(t) > 0)\\
=& P(\frac{\sum_{i\in\mathcal{I}_t} d_i - b}{|\mathcal{I}_t|} > t)\\
=& 1 - P(\sum_{i\in\mathcal{I}_t} d_i \leq |\mathcal{I}_t|\cdot t + b)\\
=& 1 - P(\sum_{i\in\mathcal{I}_t} d_i - E[\sum_{i\in\mathcal{I}_t} d_i] \leq |\mathcal{I}_t|\cdot t + b - E[\sum_{i\in\mathcal{I}_t} d_i]). \qquad (11)
\end{aligned}$$

Now observe that, for any $i \in \mathcal{I}_t$, d_i can be treated as a conditional variable: $d_i|_{d_i>t}$. Since all such d_i are i.i.d, we may denote the (shared) expected value $\mu := E[d_i|d_i > t]$ and variance as $\sigma^2 := \text{Var}[d_i|d_i > t]$. moreover, by definition of $\mathcal{I}_t$ we have

$$E[d_i|i \in \mathcal{I}_t] = \mu > t. \qquad (12)$$

Together with condition (i) of Lemma 3.2, this implies that the right-hand side of the probability in (11) is negative as $n \to \infty$:

$$|\mathcal{I}_t|\cdot t + b - E[\sum_{i\in\mathcal{I}_t} d_i] = |\mathcal{I}_t|\cdot(t-\mu) + b < 0.$$

It follows, continuing from Equation (11), that

$$\begin{aligned}
P(f(t) > 0) &\geq 1 - P(|\sum_{i\in\mathcal{I}_t} d_i - E[\sum_{i\in\mathcal{I}_t} d_i]| \geq E[\sum_{i\in\mathcal{I}_t} d_i] - |\mathcal{I}_t|\cdot t - b)\\
&\geq 1 - \frac{\texttt{Var}(\sum_{i\in\mathcal{I}_t} d_i)}{(E[\sum_{i\in\mathcal{I}_t} d_i] - |\mathcal{I}_t|\cdot t - b)^2} \quad \text{(Chebyshev's inequality)}\\
&= 1 - \frac{\sigma^2|\mathcal{I}_t|}{(\mu|\mathcal{I}_t| - t|\mathcal{I}_t| - b)^2}.
\end{aligned}$$

Thus we have the desired result:

$$P(\tau > t) \geq 1 - \frac{\sigma^2|\mathcal{I}_t|}{(\mu|\mathcal{I}_t| - t|\mathcal{I}_t| - b)^2} \to 1 \text{ as } n \to \infty. \qquad (13)$$

Now observe that $\tau > t$ implies $\mathcal{I}_\tau \subseteq \mathcal{I}_t$, and subsequently $|\mathcal{I}_\tau| \leq \mathcal{I}_t$. Together with condition (ii) from Lemma 3.2 we have

$$P(\frac{|\mathcal{I}_\tau|}{n} \leq \epsilon) \geq P(\frac{|\mathcal{I}_t|}{n} \leq \epsilon) = 1, \text{ as } n \to \infty.$$

□ □

We apply Theorem (3.3) to example distributions in Appendix B, and test it empirically in Section 5.1.

Proposition 3.4. *Let $\hat{d}$ be a subvector of d with $m \leq n$ entries; moreover, without loss of generality suppose the subvector contains the first m entries. Let $\hat{v}^*$ be the projection of $\hat{d}$ onto the simplex $\hat{\Delta} := \{v \in \mathbb{R}^m \mid \sum_{i=1}^m v_i = b, v \geq 0\}$, and $\hat{\tau}$ be the corresponding pivot value. Then, $\tau \geq \hat{\tau}$. Consequently, for $1 \leq i \leq m$ we have that $\hat{v}_i^* = 0 \implies v_i^* = 0$.*

Proof. Define two index sets,

$$\mathcal{I}_{\hat{\tau}} := \{i = 1, ..., n \mid d_i > \hat{\tau}, d_i \in d\};$$

$$\hat{\mathcal{I}}_{\hat{\tau}} := \{i = 1, ..., m \mid d_i > \hat{\tau}, d_i \in \hat{d}\}.$$

As $\hat{d}$ is a subvector of d, we have $\hat{\mathcal{I}}_{\hat{\tau}} \subseteq \mathcal{I}_{\hat{\tau}}$; thus,

$$\sum_{i \in \mathcal{I}_{\hat{\tau}}} (d_i - \hat{\tau}) \geq \sum_{i \in \hat{\mathcal{I}}_{\hat{\tau}}} (d_i - \hat{\tau}) = b,$$

$$\implies \frac{\sum_{i \in \mathcal{I}_{\hat{\tau}}} d_i - b}{|\mathcal{I}_{\hat{\tau}}|} \geq \hat{\tau}.$$

From Corollary 2.2, $\tau \geq \hat{\tau}$; from Proposition 2.1 it thus follows that $\mathcal{I}_{\hat{\tau}} \supset \mathcal{I}_\tau$. □ □

Theorem 3.1 establishes that, for i.i.d. uniformly distributed inputs, the projection has $O(\sqrt{n})$ active entries in expectation and thus has considerable sparsity as n grows; we also show this in the computational experiments of Section 5. Theorem 3.3 establishes arbitrarily sparse projections over arbitrary i.i.d. distributions. Proposition 3.4 tells us that if we project a subvector of some length $m \leq n$ onto the same b-scaled simplex in the corresponding $\mathbb{R}^m$ space, the zero entries in the projected subvector must also be zero entries in the projected full vector.

Our idea is to partition and distribute the vector d across cores (broadcast); have each core find the projection of its subvector (local projection); and combine the nonzero entries from all local projections to form a vector $\hat{v}$ (reduce), and apply a final (global) projection to $\hat{v}$. The method is outlined in Figure 1. Provided the projection v^* is sufficiently sparse, which (for instance) we have established is the case for i.i.d. distributed large-scale problems, we can expect $\hat{v}$ to have $<< n$ entries. We demonstrate the practical advantages of this procedure with various computational experiments in Section 5.

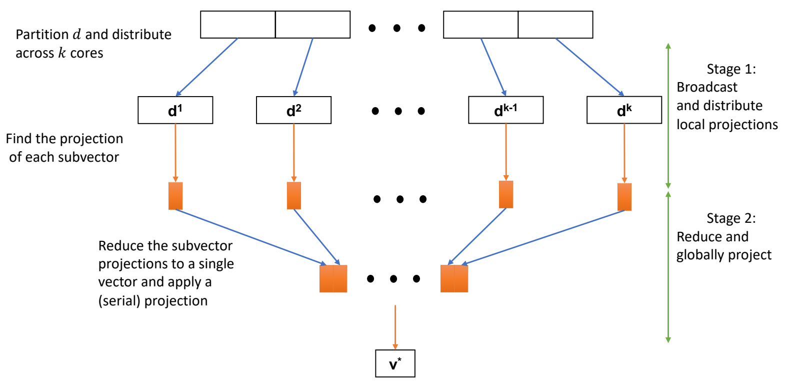
Partition d and distribute across k cores
Find the projection of each subvector
Reduce the subvector projections to a single vector and apply a (serial) projection
d^1
d^2
d^{k-1}
d^k
v^*
Stage 1: Broadcast and distribute local projections
Stage 2: Reduce and globally project

Figure 1: Distributed Projection Algorithm

3.3 Parallel Pivot and Partition

The distributed method outlined in Figure 1 can be applied directly to Pivot and Partition, as described in Algorithm 9. Note that, as presented, v^* is a sparse vector: entries not processed in the final Pivot and Project call are set to zero (recall Proposition 3.4.

Algorithm 9: Parallel Pivot and Partition

```
  **Input:** vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, cores number $k$.
  **Output:** projection $v^*$.
1 Partition $d$ into subvectors $d^1, ..., d^k$ of dimension $\leq \frac{n}{k}$ ;
2 Set $v^i :=$ Pivot_Project($d^i, b$)   (distributed across cores $i = 1, ..., k$);
3 Set $\hat{v} := \cup_{i=1}^k \{v_j^i \mid v_j^i > 0\}$;
4 Set $v^* :=$ Pivot_Project($\hat{v}, b$);
5 **return** $v^*$.
```

Proposition 3.5. *Parallel Pivot and Partition with either the median, random, or Michelot's pivot rule, has an average runtime of* $O(\frac{n}{k} + \sqrt{kn})$.

Proof. The algorithm starts by distributing projections. Pivot and Partition has linear runtime on average with any of the stated pivot rules, and so for the ith core with input d^i, whose size is $O(\frac{n}{k})$, we have an average runtime of $O(\frac{n}{k})$. Now from Theorem 3.1, each core returns in expectation at most $O(\sqrt{\frac{n}{k}})$ active terms. So the reduced input $\hat{v}$ (line 3 of Algorithm 9) will have at most $O(\sqrt{kn})$

entries on average; thus the final projection will incur an expected number of operations in $O(\sqrt{kn})$. $\square$ $\square$

In the worst case we may assume the distributed projections are ineffective, $\dim(\hat{v}) \in O(n)$, and so the final projection is bounded above by $O(n^2)$ with random pivots and Michelot's method, and $O(n)$ with the median pivot rule.

3.4 Parallel Condat's Method

We could apply the distributed sparsity idea as a preprocessing step for Condat's method. However, due to Proposition 3.6 (and confirmed via computational experiments) we have found that Filter tends to discard many non-active elements. Therefore, we propose instead to apply our distributed method to parallelize the Filter itself. Our Distributed Filter is presented in Algorithm 10: we partition d and broadcast it to the cores, and in each core we apply (serial) Filter on its subvector. Condat's method with the distributed Filter is presented as Algorithm 11.

Algorithm 10: Distributed Filter (Dfilter)

```
   Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, $k$ cores.
   Output: Index set $\mathcal{I}$ of Stage 1.
1  Partition $\mathcal{I}$ into index sets $\{\mathcal{I}_1, \cdots, \mathcal{I}_k\}$ such that $\mathcal{I}_i \leq \frac{n}{k}, i = 1,,,.k$;
2  for $i = 1 : k$ do parallel
3      Update $\mathcal{I}_i$ with (serial) Filter($d_{\mathcal{I}_i}, b$);
4      Set $p^i := \frac{\sum_{j \in \mathcal{I}_i} d_j - b}{|\mathcal{I}_i|}$;
5      for $j \in \mathcal{I}_i$ do
6          if $d_j \leq p^i$ then
7              Set $p^i := p^i + \frac{p^i - d_j}{|\mathcal{I}_i| - 1}$, $\mathcal{I}_i := \mathcal{I}_i \backslash \{j\}$
8          end
9      end
10 end
11 return $\mathcal{I} := \cup_{i=1}^k \mathcal{I}_i$.
```

Proposition 3.6. *Let $\mathcal{I}_p$ be the output of* Filter(d, b)*. Then $E[|\mathcal{I}_p|] \in O(n^{\frac{2}{3}})$.*

Proof. We assume that the **if** in line 5 from Algorithm 4 does not trigger; this is a conservative assumption as otherwise more elements would be removed from $\mathcal{I}_p$, reducing the number of iterations.

Let $p^{(j)}$ be the jth pivot with $p^{(1)} := d_1 - b$ (from line 1), and subsequent pivots corresponding to the for-loop iterations of line 2. Whenever Filter finds $d_i > p^{(j)}$, it updates p as follows:

$$p^{(j+1)} := p^{(j)} + \frac{d_i - p^{(j)}}{j+1}. \tag{14}$$

Algorithm 11: Parallel Condat's method

```
    Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, $k$ cores.
    Output: projection $v^*$.
1 Set $\mathcal{I}_p := \texttt{Dfilter}(d, b, k)$, $\mathcal{I} := \emptyset$, $p := \frac{\sum_{i\in\mathcal{I}_p} d_i - b}{|\mathcal{I}_p|}$;
2 do
3     Set $\mathcal{I} := \mathcal{I}_p$;
4     for $i \in \mathcal{I} : d_i \leq p$ do
5         Set $\mathcal{I}_p := \mathcal{I}_p \backslash \{i\}$, $p := p + \frac{p-d_i}{|\mathcal{I}_p|}$;
6     end
7 while $|\mathcal{I}| > |\mathcal{I}_p|$;
8 Set $v_i^* = \max\{d_i - p, 0\}$ for all $1 \leq i \leq n$;
9 return $v^* = (v_1^*, \cdots, v_n^*)$.
```

Moreover, when $d_i > p^{(j)}$ is found, $d_i \sim U[p^{(j)}, u]$; thus, from the Law of Total Expectation, $E[d_i] = E[E[d_i|p^{(j)}]] = (u + E[p^{(j)}])/2$. Together with (14),

$$\begin{aligned}
E[p^{(j+1)}] =& E[p^{(j)} + \frac{d_i - p^{(j)}}{j+1}] \\
=& E[p^{(j)}] + \frac{E[d_i] - E[p^{(j)}]}{j+1} \\
=& E[p^{(j)}] + \frac{(u + E[p^{(j)}])/2 - E[p^{(j)}]}{j+1} \\
=& \frac{(2j+1)E[p^{(j)}] + u}{2j+2}; \\
\implies E[p^{(j+1)}] - u =& (E[p^{(j)}] - u)\frac{2j+1}{2j+2}.
\end{aligned}$$

Using the initial value $E[p^{(1)}] = E[E[p^{(1)}|d_1]] = E[d_1] - b = \frac{u+l}{2} - b$, we can obtain a closed-form representation for the recursive formula:

$$E[p^{(j)}] - u = -2(b + \frac{u-l}{2}) \prod_{i=0}^{j-1} \frac{2i+1}{2i+2}. \tag{15}$$

Now let L_j denote the number of terms Filter scans after calculating $p^{(j)}$ and before finding some $d_i > p^{(j)}$. Since Filter scans n terms in total (from initialize and $n-1$ calls to line 4), then

$$\begin{aligned}
&\sum_{j=1}^{|\mathcal{I}_p|-1} L_j \leq n - 1 < \sum_{j=1}^{|\mathcal{I}_p|} L_j; \\
\implies & 1 + \sum_{j=1}^{|\mathcal{I}_p|-1} E[L_j] \leq n < 1 + \sum_{j=1}^{|\mathcal{I}_p|} E[L_j].
\end{aligned} \tag{16}$$

We can show L_j has a geometric distribution as follows.

Claim. $L_j \sim Geo(\frac{u-p^{(j)}}{u-l})$.

<u>Proof:</u> For each term $d_i \sim U[l,u]$, we have

$$P(d_i > p^{(j)}) = 1 - (p^{(j)} - l)/(u-l) = (u - p^{(j)})/(u-l).$$

So $d_i > p^{(j)}$ can be interpreted as a Bernoulli trial. Hence L_j is distributed with $\text{Geo}(\frac{u-p^{(j)}}{u-l})$. ■

Now, applying Jensen's Inequality to $E[L_j]$,

$$E[L_j] = E[\frac{u-l}{u-p^{(j)}}] \leq \frac{u-l}{u-E[p^{(j)}]},$$

$$\implies \ln(E[L_j]) \leq E[\ln(L_j)] = \ln(u-l) - \ln(u - E[p^{(j)}]),$$

and together with (15) we have

$$\begin{aligned} \ln(E[L_j]) \leq& \ln(\frac{u-l}{2b+u-l}) + \sum_{i=0}^{j-1} \ln(\frac{2i+2}{2i+1}) \\ =& \ln(\frac{u-l}{2b+u-l}) + \sum_{i=1}^{j} \ln(\frac{2i}{2i-1}) \end{aligned} \tag{17}$$

Now observe that

$$\lim_{i\to\infty} \frac{\ln(\frac{2i}{2i-1})}{\frac{1}{i}} = \lim_{i\to\infty} \frac{\frac{2i-1}{2i}\frac{4i-2-4i}{(2i-1)^2}}{-\frac{1}{i^2}} = \lim_{i\to\infty} \frac{2i^2}{(2i-1)(2i)} = \frac{1}{2},$$

where the first equality is from L'Hôpital's rule. Thus $E[L_j] \in \Theta(\exp(\sum_{i=1}^{j} \frac{1}{2i}))$. Furthermore, we have the classical bound on the harmonic series:

$$\sum_{i=1}^{j} \frac{1}{i} = \ln(j) + \gamma + \frac{1}{2j} \leq \ln(j) + 1,$$

where γ is the Euler-Mascheroni constant, see e.g. [26]; thus,

$$e^{\sum_{i=1}^{j} \frac{1}{2i}} = \sqrt{j} + e^{\frac{\gamma}{2} + \frac{1}{4j}},$$

which implies $E[L_j] \in O(\sqrt{j})$. Moreover (see e.g. [24, Section 1.2.7]),

$$\sum_{j=1}^{|\mathcal{I}_p|} \sqrt{j} = \frac{2}{3}|\mathcal{I}_p|\sqrt{|\mathcal{I}_p| + \frac{3}{2}} + o(\sqrt{|\mathcal{I}_p|}),$$

which implies $1 + \sum_{j=1}^{|\mathcal{I}_p|-1} E[L_j] \in O(|\mathcal{I}_p|^{\frac{3}{2}})$. From (16) it follows that $|\mathcal{I}_p| \in O(n^{\frac{2}{3}})$. □ □

Proposition 3.7. *Parallel Condat's method has an average complexity* $O(\frac{n}{k} + \sqrt[3]{kn^2})$.

Proof. In Distributed Filter (Algorithm 10), each core is given input $\mathcal{I}_i$ with $|\mathcal{I}_i| \in O(\frac{n}{k})$. (serial) Filter has linear runtime from Lemma 2.8, and the **for** loop in line 5 of Algorithm 10 will scan at most $|\mathcal{I}_i|$ terms. Thus distributed Filter runtime is in $O(\frac{n}{k})$.

From Proposition 3.6, $E[|\mathcal{I}_i|] \in O((n/k)^{\frac{2}{3}})$. Since the output of Distributed Filter is $\mathcal{I}_p = \cup_{i=1}^{k} \mathcal{I}_i$, we have that (given d is i.i.d.) $E[|\mathcal{I}_p|] \in O(\sqrt[3]{kn^2})$.

Paralle Condat's method takes the input from Distributed Filter and applies serial Condat's method (Algorithm 11, lines 2-7), excluding the serial Filter. From Proposition 2.9, Condat's method has average linear runtime, so this application of serial Condat's method has average complexity $O(\sqrt[3]{kn^2})$. $\square$ $\square$

In the worst-case we can assume Distributed Filter is ineffective ($|\mathcal{I}_p| \in O(n)$), and so the complexity of Parallel Condat's method is $O(n^2)$, same as the serial method.

3.5 Summary of Results

Table 2: Computational time complexity of serial vs parallel algorithms with problem dimension n and k cores

Method	Worst case complexity	Average complexity
Quicksort + Scan	$O(n^2)$	$O(n \log n)$
(P)Mergesort + Scan	$O(\frac{n}{k} \log n)$	$O(\frac{n}{k} \log n)$
(P)Mergesort + Partial Scan	$O(\frac{n}{k} \log n)$	$O(\frac{n}{k} \log n)$
Michelot	$O(n^2)$	$O(n)$
(P)Michelot	$O(n^2)$	$O(\frac{n}{k} + \sqrt{kn})$
Condat	$O(n^2)$	$O(n)$
(P)Condat	$O(n^2)$	$O(\frac{n}{k} + \sqrt[3]{kn^2})$

Complexity results for parallel algorithms developed throughout this section, as well as their serial counterparts, are presented in Table 2. Parallelized Sort and Scan has dependence on $\frac{1}{k}$; indeed, sorting (and scanning) are very well-studied problems from the perspective of parallel computing. However, Michelot's and Condat's methods are observed to have favorable practical performance in our computational experiments; this is expected as these more modern approaches were explicitly developed to gain practical advantages in e.g. the constant runtime factor. Conversely, our distributed method does not improve upon the worst-case complexity of Michelot's and Condat's methods, but is able to attain a $\frac{1}{k}$ factor for average complexity for $n >> k$, which is the case on large-scale instances, i.e. for all practical purposes. Our computational experiments confirm favorable practical speedups from our parallel algorithms.

We note that the Bucket method was excluded in Section 3 for brevity: although our distributed Filter can be applied in principle, we were neither able to develop an effective practical implementation of the serial method nor was it straightforward to establish the resultant expected runtime.

4 Parallelization for Extensions of Projection onto a Simplex

This section develops extensions involving projection onto a simplex, to be used for experiments in Section 5.

4.1 Projection onto the ℓ_1 Ball

Consider projection onto an ℓ_1 ball:

$$\text{Proj}_{\mathcal{B}_b}(d) := \arg\min_{v\in\mathcal{B}_b} \|v-d\|_2, \tag{18}$$

where $\mathcal{B}_b$ is given by Equation (3). Duchi et al. [19] show Problem (18) is linearly reducible to projection onto simplex in the following proposition:

Proposition 4.1 (Duchi et al. [19]).

$$\text{proj}_{\mathcal{B}_b}(d) = \begin{cases} (d_1, \cdots, d_n), & \text{if } \sum_{i=1}^n |d_i| \le b, \\ (\text{sgn}(d_1)v_1^*, \cdots, \text{sgn}(d_n)v_n^*), & \text{otherwise,} \end{cases} \tag{19}$$

where $v^* = \text{proj}_{\Delta_b}(|d|)$, $|d| = (|d_1|, \cdots, |d_n|)$, *and*

$$\text{sgn}(t) := \begin{cases} 1, & t>0 \\ 0, & t=0 \\ -1, & t<0 \end{cases}.$$

Hence, any parallel method for projection onto a simplex can be applied to Problem (18). Algorithm 12 incorporates this paralellization (for a generic parallel simplex projection algorithm), and also paralellizes post-processing of v^*.

As mentioned in Section 1, Problem (18) can itself be used as a subroutine in solving the Lasso problem, via (e.g.) Projected Gradient Descent (PGD) (see e.g. [9, Exercise 10.2]), which we present as Algorithm 13.

4.2 Centered Parity Polytope Projection

Leveraging the solution to Problem (1), Wasson et al. [36, Algorithm 2] develop a method to project a vector onto the centered parity polytope, $\mathbb{PP}_n - \frac{1}{2}$ (recall Problem 2); we present a slightly modified version as Algorithm 14. The modification is on line 11, where we determine whether a simplex projection is required to avoid unnecessary operations; the original method executes line 14 before line 11.

Algorithm 12: Parallel ℓ_1 Ball Projection

```
Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, $k$ cores.
Output: projection $v^*$.
1  if $\sum_{i=1}^n d_i \le b$ then
2      return $v^* := d$;
3  end
4  Set $v^* :=$ Parallel_Simplex_Projection$(|d|, b)$;
5  for $i = 1 : n$ do Parallel
6      if $d_i < 0$ then
7          Set $v_i^* = -v_i^*$;
8      end
9  end
10 return $v^*$.
```

Algorithm 13: Projected Gradient Descent (PGD)

```
Input: Objective function $f$, scaling factor $b$, initial point $x_0$, constant
       stepsize $\alpha$, tolerance $\epsilon$, maximum iteration number $T$.
Output: Optimum $x^*$.
1  for $t$=0,...,$T$ do
2      Set $g := \nabla f(x^t)$;
3      Set $x^{t+1} := \text{Proj}_{\Delta_b}(x^t - \alpha \cdot g)$;
4      if $\|x^{t+1} - x^t\|_2^2 \le \epsilon$ then
5          Set $x^T = x^{t+1}$;
6          Break Loop;
7      end
8  end
9  return $x^T$.
```

4.3 Projection onto a Weighted Simplex and a Weighted ℓ_1 Ball

The weighted simplex problem is

$$\Delta_{w,b} := \{v \in \mathbb{R}^n \mid \sum_{i=1}^n w_i v_i = b, v \ge 0\},$$

and the weighted ℓ_1 ball is

$$\mathcal{B}_{w,b} := \{v \in \mathbb{R}^n \mid \sum_{i=1}^n w_i |v_i| \le b\},$$

where $w > 0$ is a weight vector, and $b > 0$ is a scaling factor.

Perez et al. [31] show there is a unique $\tau \in \mathbb{R}$ such that

$$v^* = \max\{d_i - w_i\tau, 0\},\ \forall i = 1, \cdots, n,$$

Algorithm 14: Centered Parity Polytope Projection

```
Input: vector $d = (d_1, \cdots, d_n)$
Output: projection $v^*$.
1  for $i = 1 : n$ do
2      $f_i = \begin{cases} 1, & \text{if } d_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$;
3  end
4  if $1^T f$ is even then
5      Set $i^* = \text{argmin}_{i \in 1:n} |d_i|$;
6      Update $f_{i^*} = 1 - f_{i^*}$;
7  end
8  for $i = 1 : n$ do
9      Set $v_i = d_i(-1)^{f_i}$;
10 end
11 if $1^T \text{proj}_{[-\frac{1}{2}, \frac{1}{2}]^n}(v) \geq 1 - \frac{n}{2}$ then
12     return $v^* = \text{proj}_{[-\frac{1}{2}, \frac{1}{2}]^n}(d)$
13 else
14     Set $v^* = \text{proj}_{\Delta_{-\frac{1}{2}}}(d)$;
15     for $i = 1 : n$ do
16         Update $v_i^* = v_i^*(-1)^{f_i}$;
17     end
18     return $v^*$
19 end
```

where $v^* = \text{proj}_{\Delta_{w,b}}(d) \in \mathbb{R}^n$. Thus pivot-based methods for the unweighted simplex extend to the weighted simplex in a straightforward manner. We present weighted Michelot's method as Algorithm 15, and weighted Filter as Algorithm 16.

Our parallelization depends on the exact method adapted from projection onto simplex, since projection onto the weighted simplex requires direct modification rather than oracle calls to methods for the unweighted case. Sort and Scan for weighted simplex projection can be implemented with a parallel merge sort algorithm in Algorithm 17. Our distributed structure can be applied to Michelot and Condat methods in a similar manner as with the unweighted case; these are presented respectively as Algorithms 18 and 19.

Projection onto $\mathcal{B}_{w,b}$ is linearly reducible to projection onto $\Delta_{w,b}$ via the following proposition:

Proposition 4.2 (see Perez et al. [31])**.**

$$\text{proj}_{\mathcal{B}_{w,b}}(d) = \begin{cases} (d_1, \cdots, d_n), & \text{if } \sum_{i=1}^n w_i|d_i| \leq b, \\ (\text{sgn}(d_1)v_1^*, \cdots, \text{sgn}(d_n)v_n^*), & \text{otherwise}, \end{cases} \tag{20}$$

where $v^* = \text{proj}_{\Delta_{w,b}}(|d|)$, $|d| = (|d_1|, \cdots, |d_n|)$.

Algorithm 15: Weighted Michelot's method

```
Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, weight
       $w = \{w_1, ..., w_n\}$.
Output: projection $v^*$.
1 Set $\mathcal{I}_p := \{1, ..., n\}$, $\mathcal{I} := \emptyset$;
2 do
3     Set $\mathcal{I} := |\mathcal{I}_p|$;
4     Set $p := \frac{\sum_{i\in\mathcal{I}_p} w_i d_i - b}{\sum_{i\in\mathcal{I}_p} w_i^2}$;
5     Set $\mathcal{I}_t = \{i \in \mathcal{I}_p \mid \frac{d_i}{w_i} > p\}$;
6 while $|\mathcal{I}| > |\mathcal{I}_p|$;
7 Set $v_i^* := \max\{d_i - w_i p, 0\}$ for all $1 \leq i \leq n$;
8 return $v^* := (v_1^*, \cdots, v_n^*)$.
```

Note that sgn is as defined in Proposition 4.1.

5 Numerical Experiments

All algorithms were implemented in Julia and run on a laptop with Intel Core i7 CPU (6 cores and 2.6GHz). The computer has 16 GB of memory and runs 64-bit macOS Big Sur 11.4. Run times were captured using the @benchmark package in Julia, BenchmarkTools.jl [13]. Our implementations were fairly well-optimized, and they will be made publicly available (together with data sets) once our manuscript has been accepted for publication: Github [1].

5.1 Testing Theoretical Bounds

For Theorem 3.1, we calculate the average number of active elements in projecting a vector d, drawn i.i.d. from $U[0, 1]$, onto the simplex Δ_1, with n between 10^6 and 10^7 and 10 trials per size. This empirical result is compared against the corresponding asymptotic bound of $\sqrt{2n}$ given by Theorem 3.1. Results are shown in Figure 2, and demonstrate that our asymptotic bound is rather accurate for small n.

Similarly, for Proposition 3.6, we conduct the same experiments and compare the results against the function $(2.2n)^{\frac{2}{3}}$, where the constant was found empirically. Results are shown in Figure 3.

For Lemma 2.3, we run Algorithm 3 on $U[0, 1]$ i.i.d. distributed inputs d_i with scaling factor $b = 1$, size $n = 10^6$, and 100 trials. We compare the (average) remaining number of elements after each iteration of Michelot's method and against the geometric series with a ratio as $\frac{1}{2}$. We find the average number of remaining terms after each loop of Michelot's method is close to the corresponding value of the geometric series with a ratio as $\frac{1}{2}$ from 10^6. So, the conclusion from

[1]https://github.com/foreverdyz/Parallel_Projection

Algorithm 16: Weighted Filter [31]

```
Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, weight $w$.
Output: Index set $\mathcal{I}_p$.
Set $\mathcal{I}_p := \{1\}$, $\mathcal{I}_w := \emptyset$, $p =: \frac{w_1 d_1 - b}{w_1^2}$;
for $i = 2 : n$ do
    if $\frac{d_i}{w_i} > p$ then
        Set $p := \frac{w_i d_i + \sum_{j \in \mathcal{I}_p} w_j d_j - b}{w_i^2 + \sum_{j \in \mathcal{I}_p} w_j^2}$;
        if $p > \frac{w_i d_i - b}{w_i^2}$ then
            Set $\mathcal{I}_p := \mathcal{I} \cup \{i\}$;
        else
            Set $\mathcal{I}_w := \mathcal{I}_w \cup \mathcal{I}_p$;
            Set $\mathcal{I}_p = \{i\}$, $p := \frac{w_i d_i - b}{w_i^2}$;
        end
    end
end
if $|\mathcal{I}_w| \neq 0$ then
    for $i \in \mathcal{I}_w : \frac{d_i}{w_i} > p$ do
        Set $\mathcal{I}_p := \mathcal{I}_p \cup \{i\}$;
        Set $p := \frac{w_i d_i + \sum_{j \in \mathcal{I}_p} w_j d_j - b}{w_i^2 + \sum_{j \in \mathcal{I}_p} w_j^2}$;
    end
end
return $\mathcal{I}_p$.
```

Lemma 2.3, which claims that the Michelot method approximately discards half of the vector in each loop when the size is big, is accurate.

5.2 Testing Algorithms

This subsection describes and presents results for experiments conducted on the algorithms presented in Table 2. We consider the projection onto simplex problem as well as all the extensions described in Section 4. Trials were determined by @benchmark, up to a maximum of 100 per setting.

We present experimental setup and results first, and provide discussion at the end of this subsection.

5.2.1 Projection onto Simplex

Inputs d_i are drawn i.i.d. from $U[0,1]$, $N(0,1)$ and $N(0,10^{-3})$ and $b = 1$; these are commonly used benchmark distributions, e.g. [16,19,32]. Our problem sizes range up to $n = 10^8$. Results are shown in Table 3.

Algorithm 17: Weighted Parallel Sort and Parallel Scan

```
    Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, weight $w$.
    Output: projection $v^*$.
1   Set $z := \{\frac{d_i}{w_i}\}$;
2   Parallel sort $z$ as $z_{(1)} \geq \cdots \geq z_{(n)}$, and apply this order to $d$ and $w$ ;
3   Find $\kappa := \max_{k=1,\cdots,n}\{\frac{\sum_{i=1}^{k} w_i d_i - b}{\sum_{i=1}^{k} w_i^2} \leq z_k\}$;
4   Set $\tau = \frac{\sum_{i=1}^{\kappa} w_i d_i - b}{\sum_{i=1}^{\kappa} w_i^2}$;
5   Parallel set $v_i^* := \max\{d_i - w_i\tau, 0\}$ for all $1 \leq i \leq n$;
6   return $v^* = (v_1^*, \cdots, v_n^*)$.
```

Algorithm 18: Distributed Weighted Pivot and Project

```
    Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, weight $w$, $k$ cores.
    Output: projection $v^*$.
1   Partition $d$ into subvectors $d^1, ..., d^k$ of dimension $\leq \frac{n}{k}$;
2   Set $v^i :=$ Weighted_Pivot_Project($v^i, w, b$)
      (distributed across cores $i = 1, ..., k$);
3   Set $\hat{v} := \cup_{i=1}^{k}\{v_j^i \mid v_j^i > 0\}$;
4   Set $v^* :=$ Weighted_Pivot_Project($\hat{v}, w, b$);
5   return $v^*$.
```

We also created more extreme examples, choosing input distributions that might result in pathological instances. In particular, we consider: setting d as a unit vector; drawing $d_i \sim N(0,1)$ with a high scaling factor$b = 8$; setting one entry of $d_i = 1$ (an outlier) and the rest are drawn i.i.d. from $N(0, 10^{-3})$. Results are given in Table 4.

5.2.2 ℓ_1 ball

d_i are drawn i.i.d. from $U[0,1]$, $N(0,1)$, and $N(0, 10^{-3})$; $n = 10^6$, 10^7, and 10^8; and $b = 1$. Algorithms were implemented as described in Section 4.1. Results are presented in Table 5.

5.2.3 Parity Polytope

Instances were generated with $d_i \sim U[1,2]$; sizes $n = 10^5 - 1$, $10^6 - 1$, $10^7 - 1$, and $10^8 - 1$; and $b = 1$. Algorithms were implemented as described in Section 4.2. Total runtimes are given in Table 6, with time spent on projection onto simplex in brackets.

Algorithm 19: Distributed Weighted Condat

```
   Input: vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, weight $w$, $k$ cores.
   Output: projection $v^*$.
1  Partition $d$ into subvectors $d^1, ..., d^k$ of dimension $\leq \frac{n}{k}$;
2  Set $v^i :=$ Weighted_Condat_Project($v^i, w, b$)
     (distributed across cores i=1,...,k);
3  Set $\mathcal{I}_p := \cup_{i=1}^k \{j \mid v_j^i > 0\}$;
4  Set $p := \frac{\sum_{i \in \mathcal{I}_p} w_i d_i - b}{\sum_{i \in \mathcal{I}_p} w_i^2}$, $\mathcal{I} := \emptyset$;
5  do
6      Set $\mathcal{I} = \mathcal{I}_p$;
7      for $i \in \mathcal{I}$ do
8          if $\frac{d_i}{w_i} \leq p$ then
9              Set $\mathcal{I} := \mathcal{I}_p \backslash \{i\}$, $p := \frac{\sum_{i \in \mathcal{I}} w_i d_i - b}{\sum_{i \in \mathcal{I}} w_i^2}$;
10         end
11     end
12 while $|\mathcal{I}| > |\mathcal{I}_p|$;
13 Set $v_i^* = \max\{d_i - w_i p, 0\}$ for all $1 \leq i \leq n$;
14 return $v^* = (v_1^*, \cdots, v_n^*)$.
```

5.2.4 Weighted simplex, weighted ℓ_1 ball

The input vector was generated with i.i.d. elements from $N(0,1)$ and weights were drawn i.i.d. from $U[0,1]$. Total runtimes are reported in Table 7, with time spent on projection onto simplex in brackets.

5.2.5 Lasso

We implemented PGD, described in Section 4.1, to solve Lasso problems. We set $b = 1$ and $\alpha = 0.01$. The A matrix has entries drawn i.i.d. from $N(0,1)$ and $b \in \mathbb{R}^{10}$ is drawn i.i.d. from $U[0,1]$. The initialization point is also drawn i.i.d. from $U[0,1]$. Total runtimes are given in Table 8.

5.2.6 Discussion

Across a wide range of applications and instances, we observe rather consistent patterns. In terms of serial algorithms, Condat's method tended to be the fastest, with Michelot second, and the Sort and Scan approach significantly slower. This ranking held true for the parallel versions as well: our parallel Condat's method is fairly consistently the fastest algorithm, and considerably faster than the benchmark parallel Sort and Scan approach.

In terms of serial vs parallel implementations of the 'same' algorithm, parallelization of Michelot's method achieved ∼3x speedups and parallelization of Condat's method achieved ∼2x speedups consistently across instances. These

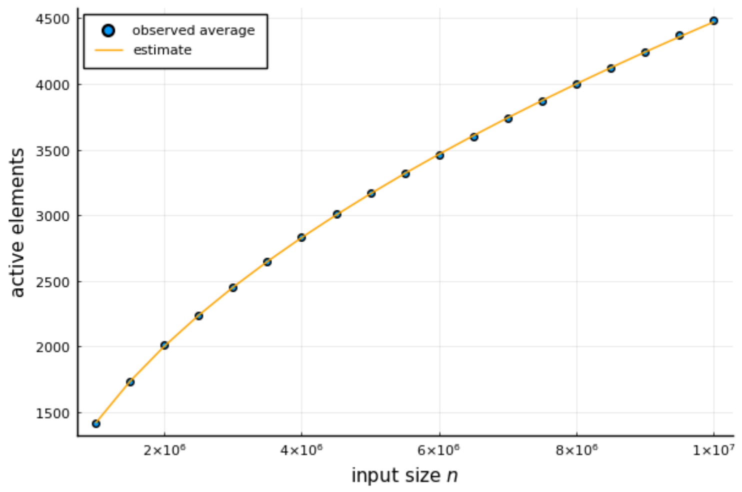

Figure 2: Compare number of active terms and value of $\sqrt{2n}$

speedups were (surprisingly) within the same range as that of Sort and Scan parallelization. Parallel sort and parallel scan are well-studied problems in parallel computing, so we expected *a priori* for such speedups to be a strict upper bound on what our method could achieve. Thus our method is not simply benefiting from its compatibility with more advanced serial projection algorithms—the distributed method itself appears to be highly effective on various large-scale instances.

We also note that our Partial Scan variant achieved modest but noticeable improvements in runtimes compared to the original parallel Sort and Scan.

6 Conclusion

We proposed a distributed preprocessing method for projection onto a simplex. Our method distributes subvector projection problems across processors in order to reduce the candidate index set on large-scale instances. The effectiveness depends on the sparsity of the projection; in the case of large-scale problems with i.i.d. inputs we can expect high levels of sparsity. A wide range of large-scale computational experiments demonstrates the consistent benefits of our method, which can be combined with various serial projection algorithms. For future work we would like to test our code on massively parallel architectures such as GPUs.

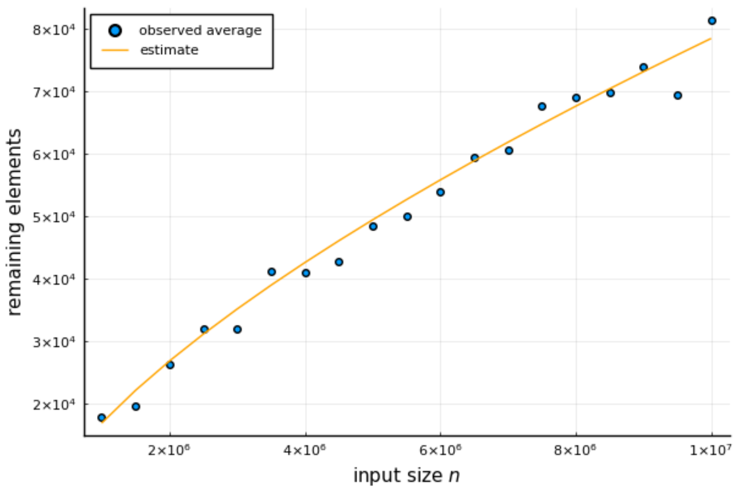

Figure 3: Compare number of remaining terms after applying Filter and value of $(2.2n)^{\frac{2}{3}}$

References

[1] Barlaud, M., Belhajali, W., Combettes, P.L., Fillatre, L.: Classification and regression using an outer approximation projection-gradient method. IEEE Transactions on Signal Processing **65**(17), 4635–4644 (2017)

[2] Barman, S., Liu, X., Draper, S.C., Recht, B.: Decomposition methods for large scale lp decoding. IEEE Transactions on Information Theory **59**(12), 7870–7886 (2013)

[3] Bentley, J.L., McIlroy, M.D.: Engineering a sort function. Software: Practice and Experience **23**(11), 1249–1256 (1993)

[4] van den Berg, E.: A hybrid quasi-newton projected-gradient method with application to lasso and basis-pursuit denoising. Math. Program. Comp. **12**, 1–38 (2020)

[5] van den Berg, E., Friedlander, M.P.: Probing the pareto frontier for basis pursuit solutions. SIAM Journal on Scientific Computing **31**(2), 890–912 (2009)

[6] Bioucas-Dias, J.M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., Chanussot, J.: Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **5**(2), 354–379 (2012)

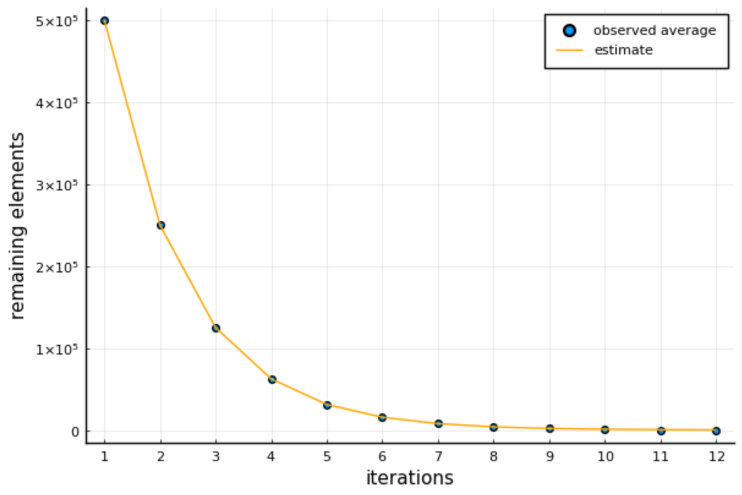

Figure 4: Compare number of remaining terms after each Michelot loop and value of geometric series with a ratio of $\frac{1}{2}$

[7] Blondel, M., Fujino, A., Ueda, N.: Large-scale multiclass support vector machine training via euclidean projection onto the simplex. In: 2014 22nd International Conference on Pattern Recognition, pp. 1289–1294 (2014)

[8] Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E., et al.: Time bounds for selection. J. Comput. Syst. Sci. **7**(4), 448–461 (1973)

[9] Boyd, S., Vandenberghe, L.: Convex optimization. Cambridge University Press (2004)

[10] Brodie, J., Daubechies, I., De Mol, C., Giannone, D., Loris, I.: Sparse and stable markowitz portfolios. Proceedings of the National Academy of Sciences **106**(30), 12267–12272 (2009)

[11] Candès, E.J., Wakin, M.B., Boyd, S.P.: Enhancing sparsity by reweighted ℓ_1 minimization. Journal of Fourier Analysis and Applications **14**, 877–905 (2008)

[12] Chartrand, R., Yin, W.: Iteratively reweighted algorithms for compressive sensing. In: 2008 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 3869–3872 (2008)

[13] Chen, J., Revels, J.: Robust benchmarking in noisy environments (2016). DOI 10.48550/ARXIV.1608.04295

[14] Chen, S.S., Donoho, D.L., Saunders, M.A.: Atomic decomposition by basis pursuit. SIAM Journal on Scientific Computing **20**(1), 33–61 (1998)

Table 3: Average runtime (s) for projection onto a simplex with typical input distributions

$U[0,1]$	$n = 10^5$	5×10^5	10^6	5×10^6	10^7	5×10^7	10^8
Sort + Scan	7.8e − 3	4.3e − 2	8.2e − 2	4.0e − 1	8.2e − 1	4.7	9.8
(P)Sort + Scan	3.6e − 3	1.4e − 2	3.6e − 2	1.9e − 1	3.9e − 1	2.3	4.7
(P)Sort + Partial Scan	3.2e − 3	1.4e − 2	3.4e − 2	1.7e − 1	3.5e − 1	2.2	4.7
Michelot	3.9e − 3	2.0e − 2	4.0e − 2	1.7e − 1	3.8e − 1	2.5	5.8
(P)Michelot	1.2e − 3	4.7e − 3	9.7e − 3	5.4e − 2	1.1e − 1	1.0	1.8
Condat	5.4e − 4	2.2e − 3	4.3e − 3	2.1e − 2	4.0e − 2	2.8e − 1	5.9e − 1
(P)Condat	4.3e − 4	1.3e − 3	2.5e − 3	1.0e − 2	2.2e − 2	1.8e − 1	3.7e − 1
$N(0,1)$	10^5	5×10^5	10^6	5×10^6	10^7	5×10^7	10^8
Sort + Scan	7.9e − 3	4.3e − 2	8.1e − 2	4.1e − 1	8.1e − 1	4.7	9.8
(P)Sort + Scan	4.5e − 3	1.6e − 2	3.3e − 2	2.0e − 1	4.8e − 1	2.5	5.0
(P)Sort + Partial Scan	3.2e − 3	1.5e − 2	3.2e − 2	1.8e − 1	4.3e − 1	2.2	5.1
Michelot	3.5e − 3	1.9e − 2	3.6e − 2	1.6e − 1	3.5e − 1	2.4	5.4
(P)Michelot	8.2e − 4	3.8e − 3	8.2e − 3	4.9e − 2	2.0e − 2	1.0	1.8
Condat	3.4e − 4	1.8e − 3	3.6e − 3	1.8e − 2	3.8e − 2	2.7e − 1	5.3e − 1
(P)Condat	2.2e − 4	7.2e − 4	1.8e − 3	7.9e − 3	1.5e − 2	1.8e − 1	3.5e − 1
$N(0,10^{-3})$	10^5	5×10^5	10^6	5×10^6	10^7	5×10^7	10^8
Sort + Scan	8.0e − 3	4.4e − 2	8.6e − 2	4.0e − 1	8.1e − 1	4.7	10.3
(P)Sort + Scan	4.0e − 3	1.6e − 2	3.4e − 2	2.0e − 1	4.7e − 1	2.4	5.1
(P)Sort + Partial Scan	3.5e − 3	1.5e − 2	3.2e − 2	1.9e − 1	4.3e − 1	2.4	4.8
Michelot	3.8e − 3	1.9e − 2	3.7e − 2	1.6e − 1	3.6e − 1	2.3	5.3
(P)Michelot	2.8e − 3	6.6e − 3	1.1e − 2	5.3e − 2	1.1e − 1	9.3e − 1	1.8
Condat	1.3e − 3	3.8e − 3	6.5e − 3	2.5e − 2	4.5e − 2	2.9e − 1	5.8e − 1
(P)Condat	3.4e − 4	1.2e − 3	2.3e − 3	9.0e − 3	1.8e − 2	1.9e − 1	3.6e − 1

[15] Chen, X., Zhou, W.: Convergence of the reweighted ℓ_1 minimization algorithm for ℓ_2-ℓ_p minimization. Computational Optimization and Applications **59**, 47–61 (2014)

[16] Condat, L.: Fast projection onto the simplex and the ℓ_1 ball. Math. Program. Ser. A **158**(1), 575–585 (2016)

[17] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)

[18] Donoho, D.L., Elad, M.: Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization. Proceedings of the National Academy of Sciences **100**(5), 2197–2202 (2003)

[19] Duchi, J., Shalev-Shwartz, S., Singer, Y., Chandra, T.: Efficient projections onto the ℓ_1-ball for learning in high dimensions. In: Proceedings of the 25th International Conference on Machine learning (ICML), pp. 272–279 (2008)

Table 4: Average runtime (s) for projection onto a simplex with extreme distributions

$n = 10^6$	unit vector	$N(0,1)$ with $b = 8$	$N(0,10^{-3})$ & 1
Sort + Scan	2.9e − 2	8.1e − 2	8.1e − 2
(P)Sort + Scan	2.2e − 2	3.4e − 2	3.4e − 2
(P)Sort + Partial Scan	1.9e − 2	3.3e − 2	3.2e − 2
Michelot	1.2e − 2	3.8e − 2	3.5e − 2
(P)Michelot	5.0e − 2	8.7e − 3	1.0e − 2
Condat	2.6e − 2	2.9e − 3	5.4e − 3
(P)Condat	1.7e − 3	1.5e − 3	2.0e − 3

[20] Fischer, H.: A history of the central limit theorem. From classical to modern probability theory. Springer (2011)

[21] Gentle, J.: Computational Statistics. Springer (2009)

[22] Held, M., Wolfe, P., Crowder, H.P.: Validation of subgradient optimization. Math. Program. **6**, 62–88 (1974)

[23] Kiwiel, K.C.: Breakpoint searching algorithms for the continuous quadratic knapsack problem. Math. Program. **112**, 473–491 (2008)

[24] Knuth, D.: The Art of Computer Programming, vol. 1, 3rd edn. Addison-Wesley (1998)

[25] Ladner, R.E., Fischer, M.J.: Parallel prefix computation. J. ACM **27**(4), 831–838 (1980)

[26] Lagarias, J.C.: Euler's constant: Euler's work and modern developments. Bull. Amer. Math. Soc. **50**, 527–628 (2013)

[27] Lellmann, J., Kappes, J., Yuan, J., Becker, F., Schnörr, C.: Convex multi-class image labeling by simplex-constrained total variation. In: Scale Space and Variational Methods in Computer Vision, pp. 150–162. Springer Berlin Heidelberg (2009)

[28] Liu, X., Draper, S.C.: The admm penalized decoder for ldpc codes. IEEE Transactions on Information Theory **62**(6), 2966–2984 (2016)

[29] Mahmoud, H.M.: Sorting: A distribution theory, vol. 54. John Wiley & Sons (2000)

[30] Michelot, C.: A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. Journal of Optimization Theory and Applications **50**, 195–200 (1986)

[31] Perez, G., Ament, S., Gomes, C.P., Barlaud, M.: Efficient projection algorithms onto the weighted ℓ_1 ball. CoRR **abs/2009.02980** (2020)

Table 5: Average runtime (s) for projection onto an ℓ_1 ball

$N(0,1)$	10^5	10^6	10^7
Sort + Scan	7.5e − 3 (6.8e − 3)	7.6e − 2 (7.1e − 2)	7.7e − 1 (7.5e − 1)
(P)Sort + Scan	3.5e − 3 (3.3e − 3)	4.1e − 2 (3.6e − 2)	4.4e − 1 (4.3e − 1)
(P)Sort + Partial Scan	2.8e − 3 (2.7e − 3)	3.9e − 2 (3.4e − 2)	4.0e − 1 (3.8e − 1)
Michelot	3.8e − 3 (3.2e − 3)	4.0e − 2 (3.5e − 2)	3.7e − 1 (3.3e − 1)
(P)Michelot	1.0e − 3 (9.0e − 4)	1.0e − 2 (7.8e − 3)	1.1e − 1 (8.6e − 2)
Condat	8.7e − 4 (2.7e − 4)	9.6e − 3 (3.1e − 3)	9.1e − 2 (3.0e − 2)
(P)Condat	4.5e − 4 (2.3e − 4)	4.2e − 3 (1.7e − 3)	3.9e − 2 (1.6e − 2)
$N(0,0.001)$	10^5	10^6	10^7
Sort + Scan	7.6e − 3 (6.9e − 3)	7.7e − 2 (7.1e − 2)	7.7e − 1 (7.2e − 1)
(P)Sort + Scan	3.5e − 3 (3.3e − 3)	4.1e − 2 (3.6e − 2)	4.6e − 1 (4.1e − 1)
(P)Sort + Partial Scan	3.1e − 3 (2.9e − 3)	4.1e − 2 (3.5e − 2)	4.3e − 1 (3.8e − 1)
Michelot	4.1e − 3 (3.5e − 3)	4.1e − 2 (3.5e − 2)	3.7e − 1 (3.2e − 1)
(P)Michelot	3.3e − 3 (2.8e − 3)	1.3e − 2 (1.1e − 2)	1.2e − 1 (9.5e − 2)
Condat	1.9e − 3 (1.4e − 3)	1.3e − 2 (6.0e − 3)	9.9e − 2 (4.0e − 2)
(P)Condat	6.8e − 4 (3.6e − 4)	4.7e − 3 (2.1e − 3)	4.4e − 2 (2.0e − 2)

Table 6: Average runtime (s) for projection onto the Parity Polytope

$U[1,2]$	$n = 10^5 - 1$	$n = 10^6 - 1$	$n = 10^7 - 1$
Sort + Scan	8.3e − 2 (5.1e − 2)	1.4e0 (8.8e − 1)	1.9e + 1 (1.3e + 1)
(P)Sort + Scan	2.6e − 2 (1.5e − 2)	4.6e − 1 (2.5e − 1)	6.7e0 (3.7e0)
(P)Sort + Partial Scan	2.3e − 2 (1.3e − 2)	5.1e − 1 (2.4e − 1)	4.9e0 (3.0e0)
Michelot	5.3e − 2 (1.6e − 2)	9.3e − 1 (2.1e − 1)	1.0e + 1 (2.3)
(P)Michelot	1.2e − 2 (3.7e − 3)	1.3e − 1 (5.1e − 2)	2.6 (6.4e − 1)
Condat	4.1e − 2 (5.4e − 3)	6.4e − 1 (5.1e − 2)	5.6 (4.2e − 1)
(P)Condat	1.0e − 2 (2.0e − 3)	9.4e − 2 (1.6e − 2)	1.2 (1.8e − 1)

[32] Perez, G., Michel Barlaud, L.F., Régin, J.C.: A filtered bucket-clustering method for projection onto the simplex and the ℓ_1 ball. Math. Program. Ser. A **182**, 445–464 (2020)

[33] Robinson, A.G., Jiang, N., Lerme, C.S.: On the continuous quadratic knapsack problem. Mathematical Programming **55**, 99–108 (1992)

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

[35] Tibshirani, R.: The lasso method for variable selection in the cox model. Statistic in Medicine **16**, 385–395 (1997)

Table 7: Average runtime (s) of projection onto a weighted simplex and a weighted ℓ_1 ball

Weighted Simplex	$n = 10^5$	$n = 5 \times 10^5$	$n = 10^6$	$n = 5 \times 10^6$	$n = 10^7$
Sort + Scan	$7.0\text{e}-2$	$5.3\text{e}-1$	$1.1\text{e}0$	$7.3\text{e}0$	$1.6\text{e}+1$
(P)Sort + Scan	$3.2\text{e}-2$	$2.4\text{e}-1$	$5.1\text{e}-1$	$3.0\text{e}0$	$5.1\text{e}0$
Michelot	$8.1\text{e}-3$	$3.9\text{e}-2$	$7.7\text{e}-3$	$3.5\text{e}-1$	$7.9\text{e}-1$
(P)Michelot	$5.4\text{e}-3$	$1.9\text{e}-2$	$3.6\text{e}-2$	$1.4\text{e}-1$	$2.6\text{e}-1$
Condat	$1.1\text{e}-3$	$4.9\text{e}-3$	$8.9\text{e}-3$	$3.8\text{e}-2$	$6.9\text{e}-2$
(P)Condat	$1.2\text{e}-3$	$3.8\text{e}-3$	$6.3\text{e}-3$	$2.4\text{e}-2$	$4.2\text{e}-2$
Weighted ℓ_1 Ball	$n = 10^5$	$n = 5 \times 10^5$	$n = 10^6$	$n = 5 \times 10^6$	$n = 10^7$
Sort + Scan	$7.1\text{e}-2$	$5.1\text{e}-1$	$1.1\text{e}0$	$7.4\text{e}0$	$1.7\text{e}+1$
(P)Sort + Scan	$3.3-2$	$2.0\text{e}-1$	$4.3\text{e}-1$	$3.0\text{e}0$	$6.9\text{e}0$
Michelot	$9.7\text{e}-3$	$4.7\text{e}-2$	$9.0\text{e}-2$	$4.2\text{e}-1$	$9.1\text{e}-1$
(P)Michelot	$7.1\text{e}-3$	$2.4\text{e}-2$	$4.4\text{e}-2$	$1.8\text{e}-1$	$3.3\text{e}-1$
Condat	$2.3\text{e}-3$	$9.5\text{e}-3$	$1.8\text{e}-2$	$7.6\text{e}-2$	$1.4\text{e}-1$
(P)Condat	$2.0\text{e}-3$	$8.0\text{e}-3$	$1.3\text{e}-2$	$5.7\text{e}-2$	$9.6\text{e}-2$

Table 8: Average runtime (s) of PGD for solving Lasso problems

$A \sim N(0,1)$ & $b \sim U[0,1]$	$A \in \mathbb{R}^{10\times 10^5}$	$A \in \mathbb{R}^{10\times 10^6}$	$A \in \mathbb{R}^{10\times 10^7}$
Sort + Scan	$1.9\text{e}-2$	$2.1\text{e}-1$	$2.2\text{e}0$
(P)Sort + Scan	$1.3\text{e}-2$	$1.5\text{e}-1$	$1.3\text{e}0$
(P)Sort + Partial Scan	$1.1\text{e}-2$	$1.4\text{e}-1$	$1.3\text{e}0$
Michelot	$1.1\text{e}-2$	$1.2\text{e}-1$	$1.3\text{e}0$
(P)Michelot	$9.4\text{e}-3$	$7.6\text{e}-2$	$7.3\text{e}-1$
Condat	$4.9\text{e}-3$	$6.2\text{e}-2$	$6.6\text{e}-1$
(P)Condat	$3.5\text{e}-3$	$5.1\text{e}-2$	$5.4\text{e}-1$

[36] Wasson, M., Milicevic, M., Draper, S.C., Gulak, G.: Hardware-based linear program decoding with the alternating direction method of multipliers. IEEE Transactions on Signal Processing **67**(19), 4976–4991 (2019)

[37] Wei, H., Jiao, X., Mu, J.: Reduced-complexity linear programming decoding based on admm for ldpc codes. IEEE Communications Letters **19**(6), 909–912 (2015)

[38] Zhang, G., Heusdens, R., Kleijn, W.B.: Large scale lp decoding with low complexity. IEEE Communications Letters **17**(11), 2152–2155 (2013)

[39] Zhang, X., Siegel, P.H.: Efficient iterative lp decoding of ldpc codes with alternating direction method of multipliers. In: 2013 IEEE International Symposium on Information Theory, pp. 1501–1505 (2013)

A Supplementary Proofs

Let $X|t$ denote the conditional variable such that $P_{X|t}(x) = P(X = x|X > t)$.

Lemma A.1. *If $X \sim U[l,u]$ and $l < t < u$, then $X|t \sim U[t,u]$.*

Proof. The CDF of X is $F_X(x) = P(X \leq x) = (x-l)/(u-l)$. Then,

$$F_{X|t}(x) = P(X \leq x|X > t) = \frac{P(X \leq x, X > t)}{P(X > t)} = (\frac{x-t}{u-l})/(\frac{u-t}{u-l}) = \frac{x-t}{u-t},$$

which implies $X|t \sim U[t,u]$. □ □

Lemma A.2. *If X, Y are independent random variables, then $X|t, Y|t$ are independent.*

Proof. X, Y are independent and so $P(X \leq x, Y \leq y) = P(X \leq x)P(Y \leq y)$ for any $x, y \in \mathbb{R}$. So considering the joint conditional probability,

$$\begin{aligned} &P(X \leq x, Y \leq y|X > t, Y > t) \\ &= \frac{P(X \leq x, Y \leq y, X > t, Y > t)}{P(X > t, Y > t)} \\ &= \frac{P(X \leq x, X > t)}{P(X > t)} \frac{P(Y \leq y, Y > t)}{P(Y > t)} \\ &= P(X \leq x|X > t)P(Y \leq y|y > t). \end{aligned}$$

□ □

Proposition A.3. *If $d_1, ..., d_n$ i.i.d. $\sim U[l,u]$ and $l < t < u$, then $\{d_i \mid i \in \mathcal{I}_t\}$ i.i.d. $\sim U[t,l]$.*

Proof. From Lemma A.1, for any $i \in \mathcal{I}_t$, $d_i \sim U[t,l]$. From Lemma A.2, for any $i, j \in \mathcal{I}_t$, $i \neq j$, d_i, d_j are conditionally independent. So, $\{d_i \mid i \in \mathcal{I}_t\}$ i.i.d. $\sim U[t,l]$. □ □

B Examples

Here we apply Theorem 3.3 to three examples:
(A) Let $d_i \sim U[0,1]$, $n_1 = 10^5$, $n_2 = 10^6$, $t = 0.95$.
(B) Let $d_i \sim N[0,1]$, $n_1 = 10^5$, $n_2 = 10^6$, $t = 1.65$.
(C) Let $d_i \sim N[0,10^{-3}]$, $n_1 = 10^5$, $n_2 = 10^6$, $t = 1.65 \times \sqrt{10^{-3}} = 0.05218$.
Observe that

$$\begin{aligned} P(\tau > t) \geq &P(\tau > t, |\mathcal{I}_p| \geq \lfloor -\sqrt{2(n\log n)p(1-p)} + np \rfloor) && \text{(21a)} \\ = &P(\tau > t \mid |\mathcal{I}_p| \geq \lfloor -\sqrt{2(n\log n)p(1-p)} + np \rfloor) && \text{(21b)} \\ \times &P(|\mathcal{I}_p| \geq \lfloor -\sqrt{2(n\log n)p(1-p)} + np \rfloor)) && \text{(21c)} \end{aligned}$$

(21b) can be calculated by (13), and (21c) can be calculated by (10).

For (A), $p = 1 - F_d(t) = 0.05$. From $n_1 p = 5000$ and $n_2 p = 50000$, we have $\sqrt{n_1 p(1-p)} = 217.9$ and $\sqrt{n_2 p(1-p)} = 68.9$. So,

$$P(|\mathcal{I}_t^1| \in [4793, 5207]) = P(|\mathcal{I}_t^2| \in [49347, 50654]) = 0.9973.$$

Now, since $f_d^*(x) = \frac{1}{0.05} = 20$, for $x \in [0.95, 1]$, we have $E = 0.975$ and $V = \frac{1}{4800}$. Applying Theorem 3.3 yields:

$$P(\tau_1 > t) \geq 0.99723, \ \forall \ |\mathcal{I}_t^1| \in [4793, 5207],$$

$$P(\tau_2 > t) \geq 0.99729, \ \forall \ |\mathcal{I}_t^2| \in [49347, 50654],$$

which implies the number of active elements in the projection should be less than 5% of n_1 or n_2 with high probability.

For (B), $p = 1 - F_d(t) = 0.05$; similar to the first example,

$$P(|\mathcal{I}_t^1| \in [4793, 5207]) = P(|\mathcal{I}_t^2| \in [49347, 50654]) = 0.9973.$$

Together with

$$f_d^* = \frac{1}{0.05}\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} = \frac{20}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}, \text{ for } x \in [1.65, +\infty),$$

we can calculate E and V:

$$E = \frac{20}{\sqrt{2\pi}}\int_{1.65}^{\infty} x e^{-\frac{x^2}{2}}\, dx = \frac{20}{\sqrt{2\pi}}e^{-\frac{1.65^2}{2}} = 2.045,$$

$$E(x^2) = \frac{20}{\sqrt{2\pi}}\int_{1.65}^{\infty} x^2 e^{-\frac{x^2}{2}}\, dx = 4.375,$$

$$\Longrightarrow V = E(x^2) - E^2 = 0.192.$$

Applying Theorem 3.3,

$$P(\tau_1 > t) \geq 0.99704, \ \forall \ |\mathcal{I}_t^1| \in [4793, 5207],$$

$$P_2(\tau > t) \geq 0.99728, \ \forall \ |\mathcal{I}_t^2| \in [49347, 50654],$$

which imply 5% of terms are active after projection with probability $> 99\%$.

For (C), $p = 1 - F_d(t) = 0.05$. Similar to the previous examples,

$$P(|\mathcal{I}_t^1| \in [4793, 5207]) = P(|\mathcal{I}_t^2| \in [49347, 50654]) = 0.9973.$$

Together with

$$f_d^* = \frac{20}{\sqrt{2\pi 10^{-3}}}e^{-\frac{x^2}{2\times 10^{-3}}}, \ \forall \ x \in [0.05218, +\infty),$$

we can calculate E and V as follows,

$$E = \frac{20}{\sqrt{2\pi 10^{-3}}}\int_{1.65\times\sqrt{10^{-3}}}^{\infty} x e^{-\frac{x^2}{2\times 10^{-3}}}\, dx = 0.03234,$$

$$E(x^2) = \frac{20}{\sqrt{2\pi 10^{-3}}} \int_{1.65\times\sqrt{10^{-3}}}^{\infty} x^2 e^{-\frac{x^2}{2\times 10^{-3}}}\, dx = 0.002187,$$

$$\implies V = E(x^2) - E^2 = 0.001142.$$

Applying Theorem 3.3,

$$P(\tau_1 > t) \geq 0.99671,\ \forall\ |\mathcal{I}_t^1| \in [4793, 5207],$$

$$P(\tau_2 > t) \geq 0.99724,\ \forall\ |\mathcal{I}_t^2| \in [49347, 50654],$$

and so 5% of terms are active after projection with probability $> 99\%$.