Parallelization of Projection onto a Simplex

Yongzheng Dai and Chen Chen
ISE, The Ohio State University, Columbus, OH, USA

April 19, 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 an efficient scheme to decompose and distribute the problem across processors, with application to a broad range of serial approaches. Our method becomes especially effective when the projection is highly sparse; which is the case, for instance, in large-scale problems with i.i.d. entries. We also fill in theoretical gaps in serial algorithm analysis; moreover we develop and analyze a variety of parallel analogues using our scheme. Numerical experiments conducted on a wide range of large-scale instances further demonstrate the effectiveness of our parallel algorithms. As our scheme can be implemented in a distributed manner, even greater practical speedups are anticipated for more specialized hardware with high levels of parallelism.

1 Introduction

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

$$\text{proj}_{\Delta_b}(d) := \arg\min_{v \in \Delta_b} \|v - d\|_2,$$  \hspace{1cm} (1)

where $\Delta_b$ 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 to determine projections onto certain other polyhedra. Such projections arise in numerous settings such as: image

1

arXiv:2204.08153v1 [math.OC] 18 Apr 2022
processing, e.g. labeling [27], or multispectral unmixing [7]; portfolio optimization [11]; and machine learning [9]. As a particular example, projection onto a simplex can be used to project onto the parity polytope (see e.g. Wasson et al. [35]):

\[
\text{proj}_{\mathbb{P}^n}(d) := \arg\min_{v \in \mathbb{P}^n} \|v - d\|_2,
\]

where \(\mathbb{P}^n\) is a \(n\)-dimensional parity polytope:

\[
\mathbb{P}^n := \text{conv}(\{v \in \{0, 1\}^n | \sum_{i=1}^n v_i = 0 \text{ (mod 2)}\}).
\]

Projection onto the parity polytope arises in linear programming (LP) decoding [3,28,36–38] for the purpose of signal processing.

Another example is projection onto a \(\ell_1\) ball:

\[
\mathcal{B}_b := \{v \in \mathbb{R}^n | \sum_{i=1}^n |v_i| \leq b\}.
\]

Duchi et al. [20] demonstrate that the solution to this problem can be recovered from projection onto a simplex using the same scaling factor. Furthermore, projection onto a \(\ell_1\) ball can, in turn, be used as a subroutine in gradient-projection methods (see e.g. van den Berg [5]) for a variety of machine learning problems that use \(\ell_1\) penalty, such as: LASSO [33]; basis-pursuit denoising [5,6,15]; sparse representation in dictionaries [19]; variable selection [34]; and classification [2].

Finally, we note that methods for projection onto the scaled standard simplex and \(\ell_1\) ball can be extended to projection onto the weighted simplex and weighted \(\ell_1\) ball [30], i.e.

\[
\Delta_{w,b} := \{v \in \mathbb{R}^n | \sum_{i=1}^n w_i v_i = b, v \geq 0\},
\]

\[
\mathcal{B}_{w,b} := \{v \in \mathbb{R}^n | \sum_{i=1}^n w_i |v_i| \leq b\},
\]

where \(w > 0\) is the weight vector, and \(b > 0\) is the scaling factor.

Projection onto the weighted simplex can, in turn, be used to solve the continuous quadratic knapsack problem [32]. Moreover, \(\ell_p\) regularization can be handled by iteratively solving weighted \(\ell_1\) projections [12,13,16].

1.2 Contributions

Much of the literature on projections onto polyhedra has focused on serial algorithms. To our knowledge, there is only one published parallel method for the projection problem (1), proposed by Wasson et al. [35], 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, and thereby develop parallel analogues to serial projection algorithms developed by Condat [17], Duchi et al. [20], Kiwiel [24], and Michelot [29], as well as an enhancement of the method of Wasson et al. The key insight to our approach is that the projection of any subvector of \( d \) onto the simplex of the corresponding space will have zero-valued entries only if the full-dimension projection has corresponding zero-valued entries. Hence we can (at least partially) determine in parallel the sparsity pattern of the projection. We provide thorough theoretical analyses of all aforementioned algorithms, and also conduct computational experiments. Our computational results demonstrate significant speedups with our parallel methods on a standard multicore CPU on large-scale (\( n > 10^5 \)) 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. Section 4 extends the parallel 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 review 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. Throughout this section, 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 [23]). 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

\[
v_i^* = \max\{d_i - \tau, 0\}, \quad \forall i = 1, \ldots, n,
\]

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

Hence, finding projection (1) can be reduced to a univariate search problem for the correct pivot \( \tau \). Note that the positive entries of \( v^* \) correspond to entries where \( d_i > \tau \). So for a given value \( t \in \mathbb{R} \) and the index set \( \mathcal{I} := \{1, \ldots, n\} \), we define the active index set \( \mathcal{I}_t \subseteq \mathcal{I} \) 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 \( \tau \):

\[
f(t) := \begin{cases} 
\frac{\sum_{i \in \mathcal{I}_t} d_i - b}{|\mathcal{I}|} - t, & t < \max_i \{d_i\} \\
-b, & t \geq \max_i \{d_i\}
\end{cases}
\]
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,
\[
f(t) = \frac{\sum_{i \in \mathcal{I}_t} d_i - b}{|\mathcal{I}_t|} - t,
\]
\[
= \frac{\sum_{i \in \mathcal{I}_t} (d_i - t) - b}{|\mathcal{I}_t|},
\]
\[
= \sum_{i=1}^n \max\{d_i - t, 0\} - b.
\]

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 \( \tau \) 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 \( \tau \) is its unique root. These results can be leveraged to develop search algorithms for \( \tau \), 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:
\[
d_{\pi_1} \geq d_{\pi_2} \geq \ldots \geq d_{\pi_n}.
\]

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 \( \kappa \) active elements, and since \( f(\tau) = 0 \), we have \( \tau = \frac{\sum_{i=1}^\kappa d_{\pi_i} - b}{\kappa} \).

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., [4]).

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. Both sorting and scan operations 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).
Algorithm 1: Sort and Scan (Held, Wolfe, and Crowder [23])

**Input:** vector \( d = (d_1, \ldots, d_n) \), scaling factor \( b \).

**Output:** projection \( v^* \).

1. Sort \( d \) as \( d_{\pi_1} \geq \cdots \geq d_{\pi_n} \);
2. Set \( \kappa := \max_{1 \leq j \leq n} \{ j : \sum_{i=1}^{j} d_{\pi_i} - b < d_{\pi_j} \} \);
3. Set \( \tau := \sum_{i=1}^{\kappa} d_{\pi_i} - b \);
4. Set \( v^*_i := \max\{d_i - \tau, 0\} \) for all \( 1 \leq i \leq n \);
5. return \( v^* = (v^*_1, \ldots, v^*_n) \).

2.3 Pivot and Partition

Sort and Scan begins by sorting all elements of \( d \), yet only the greatest \( \kappa \) terms are needed to calculate \( \tau \). Pivot and Partition, proposed by Duchi et al. [20], can be interpreted as a hybrid sort-and-scan that may avoid sorting all elements. We present as Algorithm 2 a variant of this method by Condat [17].

Algorithm 2: Pivot and Partition

**Input:** vector \( d = (d_1, \ldots, d_n) \), scaling factor \( b \).

**Output:** projection \( v^* \).

1. Set \( I := \{1, \ldots, n\}, I_\tau := \emptyset, I_p := \emptyset \);
2. while \( I \neq \emptyset \) do
3. | Select a pivot \( p \in [\min_{i \in I} \{d_i\}, \max_{i \in I} \{d_i\}] \);
4. | Set \( I_p := \{i \mid d_i > p, i \in I\} \);
5. | if \((\sum_{i \in I_p \cup I_\tau} d_i - b)/(|I_p| + |I_\tau|) > p\) then
6. | | Set \( I := I_p \);
7. | else
8. | | Set \( I_\tau := I_\tau \cup I_p, I := I \setminus I_p \);
9. | end
10. end
11. Set \( \tau := \sum_{i \in I_\tau} d_i - b / |I_\tau| \);
12. Set \( v^*_i := \max\{d_i - \tau, 0\} \) for all \( 1 \leq i \leq n \);
13. return \( v^* = (v^*_1, \ldots, v^*_n) \).

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

Several strategies have been proposed for selecting a pivot within a given
interval. Duchi et al. [20] choose a random value in the interval, while Kiwiel [24] uses the median value. The classical approach of Michelot [29] can be interpreted as a special case that sets the initial pivot as $p^{(1)} = (\sum_{i \in I} d_i - b)/|I|$, 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. The algorithm is presented separately as Algorithm 3

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{Input:} vector $d = (d_1, \ldots, d_n)$, scaling factor $b$.
\State \textbf{Output:} projection $v^*$.
\State 1 Set $I_p := \{1, \ldots, n\}$, $I := \emptyset$;
\State 2 do
\State 3 Set $I := I_p$;
\State 4 Set $p := (\sum_{i \in I} d_i - b)/|I|$;
\State 5 Set $I_p := \{i \in I \mid d_i > p\}$;
\State 6 while $|I| > |I_p|$;
\State 7 Set $v_i^* := \max\{d_i - p, 0\}$ for all $1 \leq i \leq n$;
\State 8 return $v^* := (v_1^*, \ldots, v_n^*)$.
\end{algorithmic}
\end{algorithm}

Complexity results are summarized in Table 8. Condat [17] established worst-case results for all pivots, as well as average case complexity (over the uniform distribution) specifically for the random pivot rule. We note that the median pivot method is (to our knowledge) the only worst-case linear-time algorithm for projection onto a simple, but it relies on a median-of-medians subroutine [10], which has a high constant factor; our experiments seem to confirm this issue. We fill in the theoretical gaps below establishing an average runtime of $O(n)$ for the median and Michelot pivot rules.

| Pivot rule  | Worst Case  | Average Case |
|-------------|-------------|--------------|
| Random      | $O(n^2)$    | $O(n)$       |
| Median      | $O(n)$      | $O(n)$       |
| Michelot    | $O(n^2)$    | $O(n)$       |

Table 1: Complexity results for Pivot and Partition. New results are bolded.

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

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

\[ \square \]

**Proposition 2.4.** Michelot’s method has average runtime of \( O(n) \).

**Proof.** Let \( \delta_i \) be the number of elements that Algorithm 3 removes from the (candidate) active set \( I_p \) in iteration \( i \), and let \( T \) be the total number of iterations. Then

\[
\sum_{i=1}^{T} \delta_i = n - |I_\tau|.
\]

From Lemma 2.3 and Theorem A.3, the algorithm will (recursively) remove half the entries in expectation from \( I_p \) in each iteration as \( n \) approaches infinity, so (asymptotically) \( E[\delta_i] = n/2^i \). Furthermore, in a given iteration \( j \) the algorithm scans \( |I_p(j)| \) terms. So

\[
E[|I_p(j)|] = E[n - \sum_{i=1}^{j-1} \delta_i] = n - \sum_{i=1}^{j-1} \frac{n}{2^i} = \frac{n}{2^{j-1}} = 2E[\delta_j];
\]

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

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 therefore omit a full proof of its \( O(n) \) average runtime for brevity.

### 2.4 Condat’s Method

In Michelot’s method, calculating each pivot requires a scan. Condat’s method [17], presented as Algorithm 5, can be seen as modifying Michelot’s algorithm 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 \( \tau \) from below, this results in fewer iterations (see proof of Theorem 2.8). Second, Condat’s method dynamically updates the pivot value whenever an inactive entry is removed from \( I_p \); in contrast, Michelot’s method updates the pivot every iteration by summing over all entries.

\[ \square \]
Algorithm 4: Filter

\begin{itemize}
  \item \textbf{Input:} vector \( d = (d_1, \ldots, d_n) \), scaling factor \( b \).
  \item \textbf{Output:} the Filter index set \( I_t \).
\end{itemize}

1. Set \( I_p := \{1\} \), \( I_w := \emptyset \), \( p := d_1 - b \);
2. for \( i = 2 \) : \( n \) do
3. \hspace{1em} if \( d_i > p \) then
4. \hspace{2em} Set \( p := p + \frac{d_i - p}{|I_p| + 1} \);
5. \hspace{2em} if \( p > d_i - b \) then
6. \hspace{3em} Set \( I_p := I_p \cup \{i\} \);
7. \hspace{3em} else
8. \hspace{4em} Set \( I_w := I_w \cup I_p, I_p := \{i\} \), \( p := d_i - b \);
9. \hspace{2em} end
10. \hspace{1em} end
11. end
12. for \( i \in I_w \) do
13. \hspace{1em} if \( d_i > p \) then
14. \hspace{2em} Set \( I_p := I_p \cup \{i\} \), \( p := p + \frac{d_i - p}{|I_p|} \);
15. \hspace{2em} end
16. end
17. return \( I_p \).

Condat [17] supplies a worst-case complexity of \( O(n^2) \). We supplement this with average-case analysis. Several technical lemmas, presented below, are needed to develop the final result.

Lemma 2.5. Filter provides a pivot \( p \) such that \( \tau \geq p \geq \sum_{i \in I_t} d_i - b \).

Proof. The upper bound \( p \leq \tau \) is given by construction of \( p \) (see Condat [17, Section 3, Paragraph 2]).

Now, we establish the lower bound on \( p \) by considering the sequence \( p^{(1)} \leq \ldots \leq p^{(n)} \in \mathbb{R} \), which represent 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 \( I_p^{(1)} \subseteq \ldots \subseteq I_p^{(n)} \). Filter initializes with \( p^{(1)} := d_1 - b \) and \( I_p^{(1)} := \{1\} \). For \( i = 2, \ldots, n \), if \( d_i > p^{(i-1)} \),
\[ p^{(i)} := p^{(i-1)} + (d_i - p^{(i-1)})/(|I_p^{(i-1)}| + 1), \quad I_p^{(i)} := I_p^{(i-1)} \cup \{i\}; \]
otherwise \( p^{(i)} := p^{(i-1)} \), and \( I_p^{(i)} := I_p^{(i-1)} \). Then it can be shown that \( p^{(i)} = (\sum_{j \in I_p^{(i)}} d_j - b)/|I_p^{(i)}| \) (see Condat [17, Section 4, Paragraph 2]), and \( p \geq p^{(n)} \) (see Condat [17, Section 3, Paragraph 5]). Now observe that
\[ \sum_{i \in I} d_i - b = \sum_{i \in I_p^{(n)}} d_i - b + \sum_{i \in I_p^{(n)} \setminus I_p^{(n)}} d_i \]
Algorithm 5: Condat’s Method

**Input:** vector $d = (d_1, \ldots, d_n)$, scaling factor $b$.

**Output:** projection $v^*$.

1. Set $I_p := \text{Filter}(d, b)$, $p := \frac{\sum_{i \in I_p} d_i - b}{|I_p|}$, $I := \emptyset$;
2. do
   3. Set $I := I_p$;
   4. for $i \in I : d_i \leq p$ do
      5. Set $I_p := I_p \setminus \{i\}$;
      6. Set $p := p + \frac{d_i}{|I_p|}$;
   end
8. while $|I| > |I_p|$;
9. Set $v^* := \max\{d_i - p, 0\}$ for all $1 \leq i \leq n$;
10. return $v^* = (v^*_1, \ldots, v^*_n)$.

For any $i \in I \setminus I_p$, since $I_p^{(i)} \subseteq I_p^{(n)}$ then $i \not\in I_p^{(i)}$. By construction of $p^{(i-1)}$ and $I_p^{(i)}$, we have $d_i \leq p^{(i-1)} \leq p^{(n)}$. Thus, $\sum_{i \in I \setminus I_p^{(n)}} (d_i - p^{(n)}) \leq 0$, and $p^{(n)} \geq (\sum_{i \in I} d_i - b)/|I|$. So $p \geq p^{(n)} \geq (\sum_{i \in I} d_i - b)/|I|$.

As $\sum_{i \in I} d_i - b)/|I|$ is initial pivot value of Michelot’s method (see Algorithm 3 line 4), Lemma 2.5 implies that Filter provides a (weakly) better starting pivot. We can also establish rigorously that Condat’s method initializes with one iteration of the while loop (line 2-8 in Algorithm 5) with the initial pivot $p$. Then Condat’s method removes some terms from $I_p$ to generate a new index set $I_C$ and get a new pivot $p^C$. We refer to this as one pass of Condat’s method, and it sets $I_C$ and $p^C$ for the next iteration. Likewise, we call one pass of Michelot’s method as applying one iteration of the while loop (line 2-6 in Algorithm 3) with the initial pivot $p$ and getting $I_M$ and $p^M$. For a given scalar $t < \tau$ suppose we have an initial pivot $p$ such that $t < p \leq \tau$.

Let $t_C$ and $t_M$ be the total number of iterations taken by Condat’s method and Michelot’s method on a given instance.

Let $I_0^C, \ldots, I_n^C$ be index sets for Condat’s method with corresponding pivots...
\[p^C_0, \ldots, p^C_n.\] Likewise, let \(I^M_0, \ldots, I^M_n\) and \(p^M_0, \ldots, p^M_n\) be for Michelot’s method. (Note that \(I^C_i = I^M_{i+1}\) can be true when \(p^C_i = \tau\). We suppose there are \(n\) index sets for completion.)

**Lemma 2.6.** \(I^C_i \subseteq I^M_i\), and \(p^C_i \geq p^M_i\) for \(i = 0, \ldots, n\).

**Proof.** We will prove this by induction. For the base case, \(I^C_i\) is obtained by Filter. So \(I^C_0 \subseteq I = I^M_0\). Moreover, from Lemma 2.5, \(p^C_0 \geq p^M_0\).

Now for any iteration \(i \geq 1\), suppose \(I^C_i \subseteq I^M_i\), and \(p^C_i \geq p^M_i\). From line 5 in Algorithm 3, \(I^M_{i+1} := \{j \in I^M_i : d_j > p^M_i\}\). From Condat [17, Section 3, Paragraph 3], Condat’s method uses a dynamic pivot between \(p^C_i\) to \(p^C_{i+1}\) to remove inactive entries that would otherwise remain in Michelot’s method. Therefore, \(I^C_{i+1} \subseteq \{j \in I^C_i : d_j > p^C_i\} \subseteq I^M_{i+1}\), and for any \(j \in I^M_{i+1}\setminus I^C_{i+1}\), \(d_j \leq p^C_{i+1}\). Observe that

\[
\frac{p^C_{i+1} - p^M_{i+1}}{\sum_{j \in I^C_{i+1}} d_j - b} = \frac{\sum_{j \in I^M_i} d_j - b}{|I^M_{i+1}|} - \frac{\sum_{j \in I^C_{i+1}} d_j - b}{|I^C_{i+1}|} = \frac{\sum_{j \in I^M_i} d_j + \sum_{j \in I^M_i \setminus I^C_i} d_j - b}{|I^C_{i+1}| + |I^M_{i+1} \setminus I^C_{i+1}|},
\]

which implies \(p^C_{i+1} \geq p^M_{i+1}\).

It follows by the induction that \(I^C_i \subseteq I^M_i\), and \(p^C_i \geq p^M_i\).

**Corollary 2.7.** \(\sum_{i=1}^{\ell^C} |I^C_i| \leq \sum_{i=1}^{\ell^M} |I^M_i|\).

**Proof.** Note that by design both algorithms remove elements (without replacement) from their candidate sets \(I^C_i, I^M_i\) at every iteration; moreover, they terminate with the pivot value \(\tau\) and so at their final iterations \(I^C_{\ell^C} = I^M_{\ell^M} = \mathcal{I}\). So, together with Lemma 2.6, we have for any \(i = 0, \ldots, n\), \(I^C_i \subseteq I^M_i\). So \(I^M_{\ell^M} = \mathcal{I}\) implies \(I^C_{\ell^C} = \mathcal{I}\); thus \(\ell^C \leq \ell^M\). Therefore \(\sum_{i=1}^{\ell^C} |I^C_i| \leq \sum_{i=1}^{\ell^M} |I^M_i|\).

**Proposition 2.8.** Condat’s method has an average runtime complexity \(O(n)\).

**Proof.** Since \(I_w \subseteq \mathcal{I}\) at any iteration, Filter will scan at most \(2|\mathcal{I}|\) entries; including \(O(1)\) operations to update \(p\), the Filter runtime is \(O(n)\).

After applying Filter, from Corollary 2.7, the total operations spent scanning in Condat’s method is less than (or equal to) the \(O(n)\) operations for Michelot’s method (established in Theorem 2.4); hence Condat’s average runtime is \(O(n)\).

In addition to the theory we developed above, experiments from [17, 31] demonstrate that the Filter is a practically useful preprocessing technique that can significantly improve solution times. Thus we develop a parallel Filter in Section 3.
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. [31], can be interpreted as a modification that uses multiple pivots and partitions (buckets) per iteration.

**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 \( I = \{1,\ldots,n\} \), \( I_\tau = \emptyset \);
2. for \( t = 1 : T \) do
3.  set \( I_1,\ldots,I_c := \emptyset \);
4.  for \( j = 1 : c \) do
5.   set \( p_j := (\max_{i \in I} \{d_i\} - \min_{i \in I} \{d_i\}) \cdot (c - j)/c + \min_{i \in I} \{d_i\} \);
6.   for \( i \in I : d_i \geq p_j \) do
7.     set \( I_j := I_j \cup \{i\} \), \( I := I \setminus \{i\} \);
8.   end
9. end
10. for \( j = 1 : c \) do
11.  set \( p = \frac{\sum_{i \in I_\tau} d_i + \sum_{i \in I_j} d_i - b}{|I_\tau| + |I_j|} \);
12.  if \( p \geq p_j \) then
13.    set \( I := I_j \);
14.    break the inner loop;
15.  else if \( j < c \) and \( p > \max_{i \in I_{j+1}} \{d_i\} \) then
16.    set \( I_\tau := I_\tau \cup I_j \);
17.    break the outer loop;
18.  else
19.    set \( I_\tau := I_\tau \cup I_j \);
20. end
21. end
22. set \( \tau := \frac{\sum_{i \in I_\tau} d_i - b}{|I_\tau|} \);
23. set \( v^*_i := \max \{d_i - \tau, 0\} \) for all \( 1 \leq i \leq n \);
24. return \( v^* := (v^*_1,\ldots,v^*_n) \).

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 \( I_j \) 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 numerical tolerance \( D \) of the true
pivot τ, i.e. such that |τ − τ| ≤ D. This can be ensured (see [31]) by setting

\[ T = \log_c \frac{R}{D}, \]

where \( R := \max_{i \in I} \{d_i\} - \min_{i \in I} \{d_i\} \) denotes the range of \( d \). Perez et al. [31] prove the worst-case complexity is \( O((n + c) \log_c(R/D)) \).

**Proposition 2.9.** The Bucket method has an average runtime of \( O(cn) \).

**Proof.** Let \( I(t) \) denote the index set \( I \) at the start of iteration \( t \) in the outer for loop (line 2), and \( I_j(t) \) denote the index set of the \( j \)th bucket, \( I_j \), at the end of the first inner for loop (line 4).

For a given outer for loop iteration \( t \) (line 2), the first inner for loop (line 4) uses \( O(c|I(t)|) \) 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 \( |I(t)| \) iterations. The second inner for loop (line 10) also uses \( O(c|I(t)|) \) operations. In line 11, the first sum \( \sum_{i \in I} \tau d_i \) can be updated dynamically (in the manner of a scan) as a cumulative sum as \( \tau \) is updated in line 16 or 19, thus requiring a constant number of operations per iteration \( j \). The second sum \( \sum_{i \in I_j} d_i \) is bounded above by \( O(|I(t)|) \) since \( I_j \subseteq I(t) \). Thus each iteration \( j \) of the outer for loop uses \( O(c|I(t)|) \) operations.

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

\[ E[\sum_{t=1}^T |I(t)|] = \sum_{t=1}^T \frac{n}{c^{t-1}} = \frac{n}{c-1} \left(1 - \frac{D}{R}\right). \]

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

We were unable to develop an effective parallel version of the algorithm. The obvious parallelism, handing each bucket to a different processor, runs into the issue that such a workload is not balanced in general. Nevertheless we have included this serial algorithm for completeness as it is a prominent method for which we have produced a new result analyzing average runtime.

3 Parallel Algorithms

In this section we modify an existing parallel method proposed by Wasson et al. [35], and we also propose a novel parallel structure to parallelize remaining serial projection methods presented in Section 2. Then, corresponding complexity analysis are attached after each algorithm. Throughout this section, for the purposes of average case analysis, we assume uniformly distributed inputs: \( d_1, ..., d_n \) are i.i.d \( \sim U[l, u] \). Furthermore, our parallel hardware model supposes there are \( k \) cores without communication cost.
3.1 Parallel Sort and Parallel Scan

Sort and Scan sorts the input vector and then applies scanning. Wasson et al. \[35\] parallelize this method in a natural way: first applying a parallel merge sort (see e.g. \[18, p. 797\]) and then a parallel scan \[25\]. However, parallel prefix sum calculates \(\sum_{i=1}^{j} d_{\pi_i}\) for all \(j \in I\), but only \(\sum_{i=1}^{\kappa} d_{\pi_i}\) is needed to calculate \(\tau\). We add an (potential) enhancement in Algorithm 7 to work around this issue. Namely, checks are added (lines 7 and 14) in the for loops to allow for possible early termination.

Algorithm 7: Partial Parallel Scan Algorithm (PPScan)

Input: sorted vector \(d_{\pi_1}, \ldots, d_{\pi_n}\), scaling factor \(b\)

Output: \(\tau\)

1. Set \(T := \lceil \log_2 n \rceil\), \(s[1], \ldots, s[n] = d_{\pi_1}, \ldots, d_{\pi_n}\);
2. for \(j = 1 : T\) do
   3. for \(i = 2^j : 2^j \cdot \min(n, 2^T)\) 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 \(\kappa := \min(p + 2^{j-1}, n), s[\kappa] := s[\kappa] + s[p]\);
   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 \(\tau\).

As we are adding constant operations per loop, Algorithm 7 has the same complexity as the original Parallel Sort and Scan.

3.2 Distributed Sparsity

Our next ideas for parallelization are motivated by the following theoretical results.

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

Proof. Sort \(d\) such that \(d_{\pi_1} \geq d_{\pi_2} \geq \ldots \geq d_{\pi_n}\). Thus for a given order statistic,
Algorithm 8: Parallel Merge Sort and Partial Parallel Scan Project Algorithm

**Input:** vector $d = (d_1, \ldots, d_n)$, scaling factor $b$.

**Output:** projection $v^*$.

1. Parallel merge sort $d$ as $d_{\pi_1} \geq \cdots \geq d_{\pi_n}$;
2. Set $\tau = \text{PPScan}\{d_{\pi_i}\}_{1 \leq i \leq n}, b$;
3. Parallel set $v^*_i := \max\{d_i - \tau, 0\}$ for all $1 \leq i \leq n$;
4. return $v^* = (v^*_1, \ldots, v^*_n)$.

(see e.g. [22, p. 63]),

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

Define $N := |I_{\tau}|$ 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 $I_{\pi_i}$), we have

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

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

Furthermore, from 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 inequality (6) yields

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

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

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

$$\implies E[|I_{\tau}|] = E[N] < \sqrt{\frac{2b(n+1)}{u-l}} + \frac{1}{4} + \frac{1}{2}. \quad (7)$$

which is in $O(\sqrt{n})$.  \[ \square \]
Theorem 3.2. Given \(d = (d_1, ..., d_n)\), where \(d_1, ..., d_n\) are i.i.d. an arbitrary distribution \(X\), whose PDF is \(f_X\) and probability function is \(f_X\). For any \(\epsilon > 0\), \(\Pr\left(\frac{\|I_1\|}{n} \leq \epsilon\right) = 1\) as \(n \to \infty\).

Proof. Let \(t \in \mathbb{R}\) such that \(1 - F_X(t) = \epsilon\). Then we first prove \(\Pr(\tau > t) = 1\) as \(n \to \infty\).

From Corollary 2.2, \(\tau > t\) iff \(f(t) > 0\). From Equation (5), after given \(t\), the value of \(f(t)\) is determined by \(\sum_{i \in I_t} d_i\) and \(|I_t|\). We will discuss them separately.

Define the indicator variable \(\delta_i := \begin{cases} 1, & \text{if } d_i > t \quad \text{for } i \in I; \text{ let } \tau := \sum_{i=1}^{n} \delta_i, \\ 0, & \text{otherwise} \end{cases}\) and \(p := 1 - F_X(t)\). So \(S_n = |I_t|\), and \(S_n \sim B(n, p)\) (see Theorem A.4).

When \(n \to \infty\), apply Central Limit Theorem (see e.g. [21]) to approximate the value of \(S_n\) with Gaussian distribution as follows.

\[
S_n^* := \frac{S_n - np}{\sqrt{np(1-p)}} \sim N(0,1).
\]

Let \(Z_q\) is the Z-score value for standard normal distribution. Then we have

\[
P(\frac{S_n - np}{\sqrt{np(1-p)}} \leq q \cdot 1) = Z_q, \quad (8)
\]

\[
\Rightarrow P(|S_n| \leq |np - q\sqrt{np(1-p)}, np + q\sqrt{np(1-p)}|) = Z_q,
\]

\[
\Rightarrow P(|S_n| = [|np - q\sqrt{np(1-p)}, np + q\sqrt{np(1-p)}|] \geq Z_q), \quad (9)
\]

So \(|I_t|\) can be estimated by \(n, p, q\).

Alternatively, let’s estimate \(\sum_{i \in I_t} d_i\). For \(d_i\) with \(i \in I_t\), it is a conditional variable \(d_i|d_i > t\), which is under the tail part of the original distribution \(X\). So its probability function is

\[
f_X|_{t}(x) := \begin{cases} (1 - F_X(t))^{-1} f_X(x), & x > t, \\ 0, & \text{otherwise}. \end{cases}
\]

Let \(E := \int_{x=t}^{\infty} x f_X(x) \, dx\) and \(V := \int_{x=t}^{\infty} x^2 f_X(x) \, dx - E^2\), where \(E > t\).

Based on the value of \(\sum_{i \in I_t} d_i\) and \(|I_t|\), we can calculate the probability of \(\tau > t\).

\[
\Pr(\tau > t) = \Pr(f(t) > 0)
= \Pr(\sum_{i \in I_t} d_i - b > t)
= 1 - \Pr(\sum_{i \in I_t} d_i - b \leq t)
= 1 - \Pr(\sum_{i \in I_t} d_i \leq |I_t| \cdot t + b)
= 1 - \Pr(\sum_{i \in I_t} d_i - E[\sum_{i \in I_t} d_i] \leq |I_t| \cdot t + b - E[\sum_{i \in I_t} d_i]).
\]
As $n \to \infty$, $|I_t| \to \infty$, which implies $\frac{b}{|I_t|} \to 0$; thus $E - t > \frac{b}{|I_t|}$ for a sufficient large $n$. Since $d_1, ..., d_n$ are independent, then

$$|I_t| \cdot t + b - E[\sum_{i \in I_t} d_i] = |I_t| \cdot (t - E) + b < 0, \text{ as } n \to \infty \quad (11)$$

Due to Inequality (11),

$$\sum_{i \in I_t} d_i - E[\sum_{i \in I_t} d_i] \leq |I_t| \cdot t + b - E[|I_t| \cdot t - b].$$

$$\iff |\sum_{i \in I_t} d_i| \geq |I_t| \cdot (t - E) + b < 0, \text{ as } n \to \infty \quad (12)$$

Then, we show $|I_t| \to \infty$ as $n \to \infty$. From Inequality (9), let $q = \sqrt{2 \log n}$, with $Z_q = 2 \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx - 1$

$$P(|I_p| \geq \lfloor -\sqrt{2(n \log n) p(1-p) + np} \rfloor) \geq 2 \int_{-\infty}^{\sqrt{2 \log n}} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx - 1.$$

Since

$$\frac{\partial Z_q'}{\partial n} = \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2}} \frac{\partial q}{\partial n} = \sqrt{\frac{\log n}{\pi}},$$

then $\lim_{n \to \infty} Z_q = 1$ with rate of $O(\sqrt{\log n})$.

$$P(\tau > t) \geq P(\tau > t, |I_p| \geq \lfloor -\sqrt{2(n \log n) p(1-p) + np} \rfloor) = P(\tau > t \mid |I_p| \geq \lfloor -\sqrt{2(n \log n) p(1-p) + np} \rfloor) \times P(\lfloor |I_p| \geq \lfloor -\sqrt{2(n \log n) p(1-p) + np} \rfloor \rfloor). \quad (13)$$
So when $n \to \infty$, $P(\tau > t) = 1$. Furthermore $\tau > t$ implies $I_\tau \subset I_t$; thus $|I_\tau| \leq I_t$. So, together with Inequality (9),

$$P \left( \frac{|I_\tau|}{n} \leq \epsilon \right) \geq P \left( \frac{|I_t|}{n} \leq \epsilon \right) \geq Z_q \rightarrow 1, \text{ as } n \to \infty.$$  

We provide examples for Theorem (3.2) in Appendix B.

**Proposition 3.3.** Let $\hat{d}$ be a subvector of $d$ with $m \leq n$ entries; let $\hat{v}^*$ be the projection of $\hat{d}$ onto the simplex $\{v \in \mathbb{R}^m \mid \sum_{i=1}^m v_i = b, v \geq 0\}$, and $\hat{\tau}$ be the corresponding pivot value. Then, $\tau \geq \hat{\tau}$. Consequently, $\hat{v}^*_i = 0$, then $v^*_i = 0$ for any $i$ indexed by the subvector.

**Proof.** Define two index sets,

$$I_\tau := \{i = 1, \ldots, n \mid d_i > \tau, d_i \in d\};$$

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

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

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

$$\Rightarrow \frac{\sum_{i \in I_\tau} d_i - b}{|I_\tau|} \geq \hat{\tau}.$$ 

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

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 demonstrate this empirically in Section 5. Theorem 3.2 establishes arbitrarily sparse projections over arbitrary i.i.d. distributions. Proposition 3.3 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.

Hence 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 feed nonzero entries back to the main core (reduce) for global projection. The method is outlined in Figure 1. Provided $v^*$ is sufficiently sparse, as we have argued holds (at least) for i.i.d. distributed large-scale problems, we can expect this parallel procedure to be a highly effective preprocessing technique. Indeed, we demonstrate this empirically in Section 5.

In the following subsections we describe how to apply this procedure to various serial algorithms considered in Section 2.
3.3 Parallel Pivot and Partition

Applying our idea of distributed sparsity, we parallelize Pivot and Partition in Algorithm 9.

Algorithm 9: Parallel Pivot and Partition

\[\text{Input: vector } d = (d_1, \ldots, d_n), \text{ scaling factor } b, \text{ cores number } k.\]

\[\text{Output: projection } v^{*}.\]

1. Equally partition \(d\) into \(d^1, \ldots, d^k\);
2. Distributed \(v^{*} := \text{Pivot} \_ \text{Project}(d^i, b)\);
3. Set \(v := \bigcup_{i=1}^{k} \{v_j^i \mid v_j^i > 0\}\);
4. Set \(v^{*} = \text{Pivot \_Project}(v, b)\);
5. return \(v^{*}\).

We provide below some analysis of the complexity for Algorithm 9.

Theorem 3.4. Parallel Pivot and Partition method has an averaged runtime complexity \(O\left(\frac{n}{k} + \sqrt{kn}\right)\).

Proof. First, locally project the subvector onto a scaled standard simplex. As serial Pivot and Partition method has linear runtime, locally projecting \(d^i\) whose size is \(\frac{n}{k}\) has an averaged runtime complexity \(O\left(\frac{n}{k}\right)\).

Then, from Theorem 3.1, on average, each core reduces back active elements in \(O\left(\sqrt{\frac{n}{k}}\right)\). So there are totally \(O(\sqrt{kn})\) entries in \(v\) on average, which implies the averaged runtime of the stage 2 is \(O(\sqrt{kn})\). As a result, the averaged runtime is \(O\left(\frac{n}{k} + \sqrt{kn}\right)\).
The worst case of Parallel Pivot and Partition method can be bounded by its serial method, \(O(n^2)\) for random pivot and Michelot’s method, and \(O(n)\) for median pivot.

### 3.4 Parallel Condat’s Method

We can apply again the distributed sparsity idea to Condat’s method. However, due to Lemma 3.5 as well as numerical experiments, we have found that Filter actually tends to discard most non-active elements. Therefore, we can apply our distributed idea to parallelize the Filter instead of the subsequent steps in Condat. 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 method with the distributed Filter is described in Algorithm 11.

**Algorithm 10:** Distributed Filter Algorithm (Dfilter)

**Input:** vector \(d = (d_1, \cdots, d_n)\), scaling factor \(b\), cores number \(k\).

**Output:** Index set \(\mathcal{I}\) of Stage 1.

1. Evenly partition \(d\) into \(k\) subsets \(\{I_1, \cdots, I_k\}\);
2. for \(i = 1 : k\) do parallel
   3. Update \(I_i\) with (serial) Filter;
   4. Set \(p = \frac{\sum_{j \in I_i} d_j - b}{|I_i|}\);
   5. for \(j \in I_i\) do
      6. if \(d_j \leq p\) then
         7. Set \(p := p + \frac{d_j}{|I_i|-1}\), \(I_i := I_i \setminus \{j\}\)
      8. end
   9. end
10. end
11. return \(\mathcal{I} = \bigcup_{i=1}^k I_i\).

**Lemma 3.5.** Let \(\mathcal{I}_i\) be the output of Filter on \(d\). Then \(E[|\mathcal{I}_i|] \in O(n^{3/2})\).

**Proof.** We assume (conservatively) that the if in line 5 from Algorithm 4 does not trigger (otherwise more elements are removed from \(\mathcal{I}_i\)).

Suppose the size of \(v\) is \(J\). In Filter, the pivot \(p\) will be updated \(J - 1\) times, e.g. \(p^{(1)}, \ldots, p^{(J)}\), where \(p^{(1)} := d_1 - b\). For a given \(p^{(j)}\), Filter scans remaining vector until find a \(d_i > p^{(j)}\), and update

\[
p^{(j+1)} := p^{(j)} + \frac{d_i - p^{(j)}}{j + 1}.
\]

Let \(L_j\) denote number of terms scanned between \(p^{(j)}\) and \(p^{(j+1)}\). Since the size of \(d\) is \(n\),

\[
\sum_{j=1}^{J-1} L_j \leq n - 1 < \sum_{j=1}^{J} L_j.
\]
Algorithm 11: Parallel Condat’s Method

**Input:** vector $d = (d_1, \ldots, d_n)$, scaling factor $b$, cores number $k$.

**Output:** projection $v^*$.

1. Set $\mathcal{I}_p = \text{Dfilter}(d, b, k)$, $\mathcal{I} = \emptyset$, $p := \sum_{i \in \mathcal{I}_p} \frac{d_i - b}{|\mathcal{I}_p|}$;
2. do
   3. Set $\mathcal{I} := \mathcal{I}_p$;
      4. for $i \in \mathcal{I} : d_i \leq t$ do
         5. Set $\mathcal{I}_p := \mathcal{I}_p \{i\}$, $p := p + \frac{p - d_i}{|\mathcal{I}_p|}$;
      6. end
3. while $|\mathcal{I}| > |\mathcal{I}_p|$;
4. Set $v^*_i = \max\{d_i - p, 0\}$ for all $1 \leq i \leq n$;
5. return $v^* = (v^*_1, \ldots, v^*_n)$.

Alternatively, for each term $d_i \sim U[l, u]$,

\[ P(d_i > p^{(j)}) = 1 - \frac{p^{(j)} - l}{u - l} = \frac{u - p^{(j)}}{u - l}. \]

Thus $L_j \sim \text{Geo}(\frac{u - p^{(j)}}{u - l})$. Together with (15) there is

\[ 1 + \sum_{j=1}^{j-1} E[L_j] \leq n < 1 + \sum_{j=1}^{j} E[L_j]. \quad (16) \]

Now consider $E[L_j]$. Applying Jensen’s Inequality,

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

\[ \Rightarrow \ln(E[L_j]) \leq \ln(u - l) - \ln(u - E[p^{(j)}]). \quad (17) \]

Now let’s consider $E[p^{(j)}]$. Once some $d_i > p^{(j)}$ is found, $d_i \sim U[p^{(j)}, u]$; so together with Equation (14),

\[ E[p^{(j+1)}] = E[p^{(j)} + \frac{d_i - p^{(j)}}{j + 1}] = E[p^{(j)}] + \frac{E[p^{(j)}] - u}{2j + 2}, \]

\[ \Rightarrow E[p^{(j+1)}] - u = E[p^{(j)}] - u - \frac{E[p^{(j)}] - u}{2j + 2} = (E[p^{(j)}] - u) \frac{2j + 1}{2j + 2}, \]

\[ \Rightarrow E[p^{(j)}] = u - (d_1 - b) \prod_{i=1}^{j-1} \frac{2i + 1}{2i + 2} = u - (d_1 - b) \prod_{i=1}^{j-1} \frac{2i + 1}{2i + 2}; \]

plugging into Equation (17) yields

\[ \ln(E[L_j]) \leq \ln\left(\frac{u - l}{d_1 - b}\right) + \sum_{i=1}^{j-1} \ln\left(\frac{2i + 2}{2i + 1}\right). \quad (18) \]
Finally, observe that
\[
\lim_{i \to \infty} \frac{\ln(2i+2)}{i} = \lim_{i \to \infty} \frac{2i+2 - (4i+4)}{4(2i+1)^2} = \lim_{i \to \infty} \frac{2i^2}{(2i + 1)(2i + 2)} = \frac{1}{2},
\]
and since \(\sum_{i=1}^{\infty} \frac{1}{i} = \infty\), then \(E[L_j] \in \Theta(e^{\sum_{i=1}^{\infty} \frac{1}{i}})\). 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 \(\gamma\) is the Euler-Mascheroni constant, see e.g. [26]; thus,
\[
e^{\sum_{i=1}^{\infty} \frac{1}{i}} \leq \sqrt{j} + e^2 + \frac{1}{4},
\]
which means \(E[L^j]\) is in \(O(\sqrt{j})\). Moreover
\[
\sum_{j=1}^{J} \sqrt{j} = \frac{2}{3}J\sqrt{J + \frac{3}{2}} + o(\sqrt{J}),
\]
which implies \(1 + \sum_{j=1}^{J-1} E[L_j]\) should be in \(O(J^{3/2})\). Thus from (16), \(J\) should be in the \(O(n^{2})\).

**Theorem 3.6.** Parallel Condat’s method has an average time complexity \(O\left(\frac{n}{k} + \sqrt[k]{kn^2}\right)\).

**Proof.** In stage 1, each core will filter a vector whose size is \(\frac{n}{k}\). From Theorem 2.8, the average run time should be in \(O\left(\frac{n}{k}\right)\). By Lemma 3.5, the size of vector reduced to main core for stage 2 should be in \(O\left(\sqrt[k]{kn^2}\right)\).

From Theorem 2.8 the runtime of stage 2 should be in \(O\left(\sqrt[k]{kn^2}\right)\). As a result, the overall run time complexity of Parallel Condat’s method is \(O\left(\frac{n}{k} + \sqrt[k]{kn^2}\right)\). The worst case complexity of Parallel Condat’s method remains \(O(n^2)\), i.e. the complexity of its serial version.

### 4 Parallelization for Extensions of Projection onto a Simplex

We describe in this section both serial and parallel analogues of extensions involving projection onto a simplex.

\[\text{21}\]
4.1 Projection onto the $\ell_1$ Ball

For the projection onto a $\ell_1$ ball:

$$\text{Proj}_{B_b}(d) := \arg \min_{v \in B_b} \|v - d\|_2,$$

where $B_b$ has been defined in Equation (3). Duchi et al. [20] show Problem (19) is linearly reducible to Problem (1) by following proposition:

**Proposition 4.1** (Duchi et al. [20]).

$$\text{proj}_{B_b}(d) = \begin{cases} (d_1, \cdots, d_n), & \text{if } \sum_{i=1}^n |d_i| \leq b, \\ (\text{sgn}(d_1)v^*_1, \cdots, \text{sgn}(d_n)v^*_n), & \text{otherwise,} \end{cases}$$

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, parallel methods to Problem (1) can be applied Problem (19). Namely $\ell_1$-ball projection can be parallelized by parallelizing projection onto a simplex and parallelizing post-processing of $v^*$, presented in Algorithm 12.

**Algorithm 12: Parallel $\ell_1$ Ball Project Algorithm**

**Input:** vector $d = (d_1, \cdots, d_n)$, scaling factor $b$, cores number $k$.

**Output:** projection $v^*$.

1. if $\sum_{i=1}^n d_i \leq b$ then
2. return $v^* := d$;
3. end
4. Set $v^* := \text{Parallel Simplex Project}(|d|, b)$;
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 $v^*$.

Alternatively, as we mentioned in Section 1, Problem (19) can be used as a subroutine in solving the Lasso problem, e.g. nonmonotone spectral projected-gradient method (SPG) [8], outlined in Algorithm 13. Specifically, the condition in line 15 is nonmonotone curvilinear backtracking line-search as follows,

$$f(x(\beta k \alpha)) > \max_{0 \leq j \leq \min\{i, M-1\}} \{f(x_{i-1})\} + \gamma(\nabla f(x_i))^T (x(\beta k \alpha) - x_i),$$

where $M \in \mathbb{Z}_{+}$ and $x(\alpha) := \text{proj}_{B_b}(x_i - \alpha)$. In case of $M = 1$, the condition turns back the the standard backtracking line-search [1].
Algorithm 13: The nonmonotone spectral projected-gradient method (SPG)

**Input:** $A, d, b, x_0, \alpha_{\text{min}}, \alpha_{\text{max}}, \epsilon$.

**Output:** Optimum $x^*$.

1. Initialize $i = 0$, $\alpha_0 \in [\alpha_{\text{min}}, \alpha_{\text{max}}]$;
2. do
3. Set $g_i = \nabla f(x_i)$;
4. if $i > 0$ then
5. Set $s = x_i - x_{i-1}$, $y = g_i - g_{i-1}$;
6. if $s^T y > 0$ then
7. Set $\alpha_i = \max(\min(\frac{s^T s}{s^T y}, \alpha_{\text{max}}), \alpha_{\text{min}})$;
8. else
9. $\alpha_i = \alpha_{\text{max}}$;
end
11. end
12. Set $k=0$;
13. while Condition do
14. Update $k = k + 1$;
15. end
16. Set $x_{i+1} = \text{proj}_{B_b}(x_i - \beta^k \alpha)$;
17. Update $i = i + 1$;
18. while $\|x_{i+1} - x_i\|_2 > \epsilon$;
19. return $x^* := x_i$.

4.2 Parity Polytope
Leveraging the solution to Problem (1), Wasson et al. [35] develop a method to project a vector onto the centered parity polytope, $\mathbb{P}_n \setminus \frac{1}{2}$. We tune a bit steps of the algorithm proposed by Wasson et al. [35, Algorithm 2]; and the variant is presented in Algorithm 14. On line 11, we determine whether a simplex projection is required to avoid unnecessary operations, while the algorithm from Wasson et al. [35] executes line 14 before line 11. Likewise we parallelize the projection onto simplex in our experiments.

4.3 Projection onto Weighted Simplex and Weighted $\ell_1$ Ball
Consider a problem
\[
\text{Proj}_{\Delta_{w,b}}(d) := \arg \min_{v \in \Delta_{w,b}} \|v - d\|_2.
\] (21)

Perez et al. [30] show there is a unique $\tau \in \mathbb{R}$ such that
\[
v^* = \max\{d_i - w_i \tau, 0\}, \ \forall i = 1, \ldots, n,
\]
Algorithm 14: Centered Parity Polytope Projection Algorithm

Input: vector $d = (d_1, \cdots, d_n)$
Output: projection $v^*$.

1 for $i = 1 : n$ do
2 \quad $f_i = \begin{cases} 1, & \text{if } d_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$;
3 end
4 if $1^T f$ is even then
5     Set $i^* = \arg\min_{i \in 1:n} |d_i|$;
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 $1^T \proj_{[-\frac{1}{2}, \frac{1}{2}]^n}(v) \geq 1 - \frac{n}{2}$ then
12 \quad return $v^* = \proj_{[-\frac{1}{2}, \frac{1}{2}]^n}(d)$
13 else
14     Set $v^* = \proj_{\Delta \nabla \frac{1}{2}}(d)$;
15     for $i = 1 : n$ do
16         Update $v_i^* = v_i^* (-1)^{f_i}$;
17     end
18 \quad return $v^*$
19 end

where $v^* = \proj_{\Delta \nabla \frac{1}{2}}(d) \in \mathbb{R}^n$. Thus pivot-based methods for the unweighted case extend in a straightforward manner. We present weighted Michelot’s method in Algorithm 15, and weighted Filter in Algorithm 16.

Our parallelization depends on the exact method adapted from projection onto simplex. The Sort and Scan idea 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 the unweighted case; this is in Algorithms 18 and 19. Furthermore, weighted $\ell_1$ ball projection can be obtained from the weighted simplex projection from Proposition 4.2.

Furthermore $\proj_{B_{w,b}}$ is linear reducible to Problem (21) via the following proposition:

**Proposition 4.2** (see Perez et al. [30]).

\[
\proj_{B_{w,b}}(d) = \begin{cases} (d_1, \cdots, d_n), & \text{if } \sum_{i=1}^n w_i |d_i| \leq b, \\ (\sgn(d_1)v_1^*, \cdots, \sgn(d_n)v_n^*), & \text{otherwise}, \end{cases}
\]

where the $v^* = \proj_{\Delta \nabla \frac{1}{2}}(|d|)$, $|d| = (|d_1|, \cdots, |d_n|)$,
Algorithm 15: Weighted Michelot Project Algorithm

**Input:** vector $d = (d_1, \ldots, d_n)$, scaling factor $b$, weight $w = \{w_1, \ldots, w_n\}$.

**Output:** projection $v^*$.

1. Set $\mathcal{I}_p := \{1, \ldots, n\}$, $\mathcal{I} := \emptyset$;
2. do
3. Set $\mathcal{I} := |\mathcal{I}_p|$;
4. Set $p := \frac{\sum_{i \in \mathcal{X}_p} w_i d_i - b}{\sum_{i \in \mathcal{X}_p} w_i^2}$;
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 $v^* := (v_1^*, \ldots, v_n^*)$.

where $\text{sgn}$ is defined in Proposition 4.1.

5 Numerical Experiments

The computational complexities of different methods studied in this section have been summarized in Table 2. 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. When we compare run times, we use the `@benchmark` in Julia package, BenchmarkTools.jl [14]. All code, both the baseline and our proposed methods are well-optimized, and will be publicly available once our manuscript has been accepted for publication: Github

The remainder of this section is organized as follows. First, we check the bounds provided by Theorem 3.1, Lemma 2.3, and Lemma 3.5. Second, we compare run times for main methods to project onto the scaled standard simplex, and we also conduct some robustness tests for these methods. Third, we compare run times for corresponding methods to project onto the $\ell_1$ ball and parity polytope. Finally, we implement these methods to project onto the weighted simplex and weighted $\ell_1$ ball and conduct some experiments to compare their run times.

5.1 Testing Theoretical Bounds

For Theorem 3.1, we calculate the average number of active elements in projecting a vector in $\mathcal{U}[0, 1]$ onto $\Delta_1$ (10 instances per entry), with $n$ between $10^6$ and $10^7$. The result is shown both in Table 3 and Figure 2; and we also attach the square roots of the double corresponding size $n$ as $\sqrt{2n}$. The Theorem is clearly providing accurate estimates.

\[\text{https://github.com/foreverdyz/Parallel\_Projection}\]
Algorithm 16: Weighted Filter Technique Algorithm (wFilter) [30]

**Input:** vector $d = (d_1, \ldots, d_n)$, scaling factor $b$, weight $w$.

**Output:** Index set $I_p$.

```plaintext
1 Set $I_p := \{1\}$, $I_w := \emptyset$, $p := \frac{w_1d_1-b}{w_1^2}$;

2 for $i = 2 : n$ do

3    if $\frac{d_i}{w_i} > p$ then

4        Set $p := \frac{w_id_i + \sum_{j \in I_p} w_j d_j - b}{w_i^2 + \sum_{j \in I_p} w_j^2}$;

5        if $p > \frac{w_id_i-b}{w_i^2}$ then

6            Set $I_p := I \cup \{i\}$;

7        else

8            Set $I_w := I_w \cup I_p$, $I_w, I_p = \{i\}$, $p := \frac{w_id_i-b}{w_i^2}$;

9        end

10    end

11 end

12 if $|I_w| \neq 0$ then

13    for $i \in I_w : \frac{d_i}{w_i} > p$ do

14        Set $I_p := I_p \cup \{i\}$;

15        Set $p := \frac{w_id_i + \sum_{j \in I_p} w_j d_j - b}{w_i^2 + \sum_{j \in I_p} w_j^2}$;

16    end

17 end

18 return $I_p$.
```

Similarly, for Lemma 3.5, we conduct the same experiments and show the result in Table 3. We also compare against the function $(2^n)^\frac{2}{3}$ (the exponent is given from our Lemma, the constant was found empirically), shown in Figure 2.

For Lemma 2.3, we run Algorithm 3 in $U[0,1]$ with scaling factor $b = 1$ and a size as $10^6$ 100 times. We record remaining number of elements after each loop and calculate the average of these numbers. We show the result in Table 4, and we also attach a geometric series with a ratio as $\frac{1}{2}$ from $10^6$. 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 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 Experiments for Scaling Standard Simplex

For Algorithm 9, we implemented the Michelot pivot rule due to its superior performance empirically; we exclude the other rules for brevity. First, we gen-
Algorithm 17: Parallel Sort and Parallel Scan Weighted Project Algorithm

**Input:** vector \( d = (d_1, \cdots, d_n) \), scaling factor \( b \), weight \( w \).

**Output:** the projecting result \( v^* \).

1. Set \( z := \left\{ \frac{d_i}{w_i} \right\} \);
2. Parallel sort \( z \) as \( z^{(1)} \geq \cdots \geq z^{(n)} \), and sort \( d, w \) with such order;
3. Find \( K := \max_k\{\sum_{i=1}^k w_i d_i - b \sum_{i=1}^k w_i^2 \leq z_k\} \);
4. Set \( \tau = \frac{\sum_{i=1}^K w_i d_i - b}{\sum_{i=1}^n w_i} \);
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 Project Algorithm

**Input:** vector \( d = (d_1, \cdots, d_n) \), scaling factor \( b \), weight \( w \), cores number \( m \).

**Output:** the projecting result \( v^* \).

1. Equally partition \( d \) into \( m \) sequences as \( \{u_1, \cdots, u_m\} \);
2. Set \( v = \emptyset \);
3. Distributed project \( u_i \) onto the weighted simplex with pivot method;
4. Reduce activity terms in \( u_i \) to \( v \);
5. Project \( v \) onto the weighted simplex with pivot algorithms as \( v^* \);
6. **return** \( v^* \).

We generate data with \( d_i \) i.i.d. in \( U[0, 1], N(0, 1) \) and \( N(0, 10^{-3}) \) and \( b = 1 \), which are all commonly used benchmark distributions, e.g. [17, 20, 31]. Our problem sizes range up to \( n = 10^8 \). Results are shown in Table 5. The runtime is the mean time from @benchmark, and we bold the time that is prominently less than other runtime based on identical project methods. We find our parallel algorithms are faster than these corresponding serial algorithms leveraging a modest number of cores.

### 5.3 Experiments for \( \ell_1 \) Ball

We generate data from \( U[0, 1], N(0, 1) \), and \( N(0, 10^{-3}) \) with sizes of \( 10^6 \), \( 10^7 \), and \( 10^8 \), and \( b = 1 \). We use various methods to determine the projection and report the result in Table 7. The parallel methods demonstrate substantial speedups.

### 5.4 Experiments for Parity Polytope

We generate data from \( U[1, 2] \) with sizes \( n = 10^5 - 1, 10^6 - 1, 10^7 - 1, \) and \( 10^8 - 1 \), with \( b = 1 \). Vectors are projected onto the standard simplex before
Algorithm 19: Distributed Weighted Condat Project Algorithm

**Input:** vector \( d = (d_1, \ldots, d_n) \), scaling factor \( b \), weight \( w \), cores number \( m \).

**Output:** projection \( v^* \).

1. Equally partition \( d \) into \( m \) sequences as \( \{w_1, \ldots, w_m\} \);
2. Distributed weighted Filter each \( w_i \), and reduce the results to index set \( I \);
3. Set \( p := \sum_{i \in I} w_i d_i - b \sum_{i \in I} w_i^2 \), \( l := 0 \);
4. while \( |I| \neq l \) do
   5.  Set \( l := |I| \);
   6.  for \( i \in I \) do
      7.  if \( \frac{d_i}{w_i} \leq p \) then
      8.      Remove \( i \) from \( I \), set \( p := \frac{\sum_{i \in I} w_i d_i - b}{\sum_{i \in I} w_i^2} \);
      9.  end
   10. end
   11. end
12. Set \( v_i^* = \max\{d_i - w_i p, 0\} \) for all \( 1 \leq i \leq n \);
13. return \( v^* = (v_1^*, \ldots, v_n^*) \).

being projected onto the centering parity polytope. Results are shown in Table 8, again demonstrating the advantage of our parallelization scheme.

5.5 Experiments for Weighted Simplex and Weighted \( \ell_1 \) Ball

We implement methods to project onto the weighted simplex and weighted \( \ell_1 \) ball from [30] and Algorithm 17, 18, 19. The vector is generated i.i.d. from \( N(0, 1) \) and weight from i.i.d. \( U[0, 1] \). Results are reported in Table 9.

5.6 Experiments for Lasso Problem

We implement the Algorithm 13 to solve the Lasso Problem with different projection methods. We preset the \( b = 1, \alpha_{\min} = 10^{-3}, \alpha_{\max} = 1, \epsilon = 10^{-3}, \gamma = 1, \beta = 0.5, M = 5, \) and \( \alpha_0 = 1 \). The \( A \in \mathbb{R}^{10 \times 10^5} \) is in \( N(0, 1) \), and \( b \in \mathbb{R}^{10} \) is in \( U[0, 1] \). We initialize the start points as all-zero vectors. The result is reported in Table 10.

6 Conclusion

In this paper we have proposed a distributed scheme for the problem of projecting onto a simplex. In particular, we solve subvector projection problems in parallel in order to greatly reduce the candidate index set; thus we can view
Table 2: Computational time complexity of serial vs parallel analogues given problem dimension \( n \) and \( k \) cores

| Method                  | Worst case complexity | Average complexity |
|-------------------------|-----------------------|--------------------|
| Quicksort + Scan        | \( O(n^2) \)          | \( O(n \log n) \)  |
| (P)Mergesort + Scan     | \( O\left(\frac{n}{k} \log n\right) \) | \( O\left(\frac{n}{k} \log n\right) \) |
| (P)Mergesort + Partial Scan | \( O\left(\frac{n}{k} \log n\right) \) | \( O\left(\frac{n}{k} \log n\right) \) |
| Michelot                | \( O(n^2) \)          | \( O(n) \)         |
| (P)Michelot             | \( O(n^2) \)          | \( O\left(\frac{n}{k} + \sqrt{kn}\right) \) |
| Condat                  | \( O(n^2) \)          | \( O(n) \)         |
| (P)Condat               | \( O(n^2) \)          | \( O\left(\frac{n}{k} + \sqrt{kn^2}\right) \) |

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

it as a parallel preprocessing scheme. Our methods are justified both theoretically and empirically. For future work we would like to explore the potential on massively parallel architectures such as GPUs.
Table 3: Average numbers of active elements and square roots of corresponding sizes; average number of remaining elements after Filter and cubic roots of corresponding square value of size

| Size($\times 10^6$) | Active elements | $\sqrt{2n}$ | Remaining elements | $(2.2n)^{\frac{1}{3}}$ |
|---------------------|-----------------|-------------|--------------------|-----------------------|
| 1.0                 | 1415            | 1414.2      | 17947.8            | 16915.4               |
| 1.5                 | 1731.3          | 1732.2      | 19709.2            | 22165.4               |
| 2.0                 | 2021.2          | 2000.0      | 26328.2            | 26851.5               |
| 2.5                 | 2234.5          | 2236.1      | 31963.6            | 31158.4               |
| 3.0                 | 2447.2          | 2449.5      | 32102.6            | 35185.4               |
| 3.5                 | 2650.6          | 2645.8      | 41205.3            | 38993.6               |
| 4.0                 | 2837.0          | 2828.4      | 41107.4            | 42624.1               |
| 4.5                 | 3012.3          | 3000.0      | 42850.4            | 46105.9               |
| 5.0                 | 3168.3          | 3162.3      | 48564.9            | 49460.9               |
| 5.5                 | 3317.4          | 3316.6      | 50075.8            | 52705.6               |
| 6.0                 | 3458.0          | 3464.1      | 54051.7            | 55853.4               |
| 6.5                 | 3599.9          | 3605.6      | 59540.0            | 58914.7               |
| 7.0                 | 3738.2          | 3741.7      | 60725.4            | 61898.6               |
| 7.5                 | 3870.5          | 3873.0      | 67677.6            | 64812.1               |
| 8.0                 | 3998.5          | 4000.0      | 67677.6            | 67661.5               |
| 8.5                 | 4125.5          | 4123.1      | 69121.5            | 70452.2               |
| 9.0                 | 4244.9          | 4242.6      | 74052.6            | 73188.6               |
| 9.5                 | 4369.6          | 4358.9      | 69506.8            | 75874.8               |
| 10.0                | 4486.9          | 4472.1      | 81481.2            | 78514.2               |

References

[1] Armijo, L.: Minimization of functions having Lipschitz continuous first partial derivatives. Pacific Journal of Mathematics 16(1), 1 – 3 (1966). DOI pjm/1102995080

[2] 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)

[3] 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). DOI 10.1109/TIT.2013.2281372

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

[5] van den Berg, E.: A hybrid quasi-newton projected-gradient method with application to lasso and basis-pursuit denoising. Mathematical Programming Computation 12, 1–38 (2020)
Table 4: Average numbers of remaining elements after each loop of Michelot’s method and a geometric series with $\frac{1}{2}$

| Loop number | Remaining elements | Geometric series |
|-------------|--------------------|-----------------|
| 1           | 500024.04          | 500000          |
| 2           | 249970.59          | 250000          |
| 3           | 124982.27          | 125000          |
| 4           | 62520.42           | 62500           |
| 5           | 31244.25           | 31250           |
| 6           | 15597.25           | 15625           |
| 7           | 7759.04            | 7812.5          |
| 8           | 3826.28            | 3906.25         |
| 9           | 1795.62            | 1963.125        |
| 10          | 701.12             | 981.5625        |
| 11          | 155.21             | 490.7625        |
| 12          | 8.69               | 245.38125       |

[6] 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)

[7] 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). DOI 10.1109/JSTARS.2012.2194696

[8] Birgin, E.G., Martínez, J.M., Raydan, M.: Nonmonotone spectral projected gradient methods on convex sets. SIAM Journal on Optimization 10(4), 1196–1211 (2000)

[9] 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). DOI 10.1109/ICPR.2014.231

[10] 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)

[11] 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)

[12] Candès, E.J., Wakin, M.B., Boyd, S.P.: Enhancing sparsity by reweighted $\ell_1$ minimization. Journal of Fourier Analysis and Applications 14, 877–905 (2008)
Figure 3: Compare number of remaining terms after each Michelot loop and value of geometric series with a ratio of $\frac{1}{2}$.

[13] 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). DOI 10.1109/ICASSP.2008.4518498

[14] Chen, J., Revels, J.: Robust benchmarking in noisy environments. arXiv e-prints arXiv:1608.04295 (2016)

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

[16] Chen, X., Zhou, W.: Convergence of the reweighted $\ell_1$ minimization algorithm for $\ell_2$-$\ell_p$ minimization. Computational Optimization and Applications 59(1–2) (2014). DOI 10.1007/s10589-013-9553-8. URL https://doi.org/10.1007/s10589-013-9553-8

[17] Condat, L.: Fast projection onto the simplex and the $l_1$ ball. Mathematical Programming 158(1), 575–585 (2016)

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

[19] Donoho, D.L., Elad, M.: Optimally sparse representation in general (nonorthogonal) dictionaries via $l_1$ minimization. Proceedings of the National Academy of Sciences 100(5), 2197–2202 (2003)

[20] 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) (2008)
Table 5: The runtime(s) of Simplex Experiments

|            | $U[0, 1]$ | $5 	imes 10^5$ | $10^6$ | $5 	imes 10^6$ | $10^7$ | $5 	imes 10^7$ | $10^8$ |
|------------|-----------|-----------------|--------|-----------------|--------|-----------------|--------|
| 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   | $4.0e-3$  | $2.6e-2$        | $4.1e-2$| $1.7e-1$        | $3.8e-1$| $2.5$           | $5.6$  |
| (P)Michelot| $1.6e-3$  | $5.3e-3$        | $1.1e-2$| $5.8e-2$        | $1.1e-1$| $9.9e-1$        | $2.0$  |
| Condat     | $5.1e-4$  | $2.0e-3$        | $3.7e-3$| $1.8e-2$        | $3.5e-2$| $2.6e-1$        | $5.3e-1$|
| (P)Condat  | $4.6e-4$  | $1.5e-3$        | $2.8e-3$| $1.0e-2$        | $2.3e-2$| $2.0e-1$        | $3.6e-1$|

|            | $N(0, 1)$ | $5 	imes 10^5$ | $10^6$ | $5 	imes 10^6$ | $10^7$ | $5 	imes 10^7$ | $10^8$ |
|------------|-----------|-----------------|--------|-----------------|--------|-----------------|--------|
| 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   | $3.6e-3$  | $1.9e-2$        | $3.8e-2$| $1.7e-1$        | $3.5e-1$| $2.3$           | $5.2$  |
| (P)Michelot| $1.6e-3$  | $4.3e-3$        | $9.4e-3$| $4.9e-2$        | $9.9e-2$| $8.6e-1$        | $1.8$  |
| Condat     | $2.7e-4$  | $1.3e-3$        | $3.0e-3$| $1.5e-2$        | $3.2e-2$| $2.6e-1$        | $4.8e-1$|
| (P)Condat  | $2.2e-4$  | $7.1e-4$        | $1.7e-3$| $7.9e-3$        | $1.6e-2$| $1.8e-1$        | $3.5e-1$|

[21] Fischer, H.: A history of the central limit theorem. From classical to modern probability theory. Springer (2011). DOI 10.1007/978-0-387-87857-7

[22] Gentle, J.: Computational Statistics. Springer (2009). DOI 10.1007/978-0-387-98144-4

[23] Held, M., Wolfe, P., Crowder, H.P.: Validation of subgradient optimization. Mathematical Programming 6, 62–88 (1974)

[24] Kiwiel, K.C.: Breakpoint searching algorithms for the continuous quadratic knapsack problem. Mathematical Programming 112, 473–491 (2008)

[25] Ladner, R.E., Fischer, M.J.: Parallel prefix computation. J. ACM 27(4), 831–838 (1980). DOI 10.1145/322217.322232. URL https://doi.org/10.1145/322217.322232

[26] Lagarias, J.C.: Euler’s constant: Euler’s work and modern developments. Bull. Amer. Math. Soc. 50, 527–628 (2013)
Table 6: The runtime(s) of Simplex Robustness Experiments

| Size is $n = 10^6$ | Unit Vector | $N(0, 1)$ with $b = 8$ | $N(0, 10^{-3})$ & 1 (outlier) |
|--------------------|-------------|-------------------------|---------------------------------|
| Sort + Scan        | 2.8e-2      | 8.2e-2                  | 7.1e-2                          |
| (P)Sort + Scan     | 2.2e-2      | 3.5e-2                  | 3.3e-2                          |
| (P)Sort + Partial Scan | 1.9e-2   | 3.4e-2                  | 3.1e-2                          |
| Michelot           | 8.2e-2      | 3.3e-2                  | 3.4e-2                          |
| (P)Michelot        | 5.3e-2      | 9.0e-1                  | 1.0e-2                          |
| Condat             | 2.6e-2      | 2.9e-3                  | 5.6e-3                          |
| (P)Condat          | 1.7e-3      | 1.9e-3                  | 1.7e-3                          |

Table 7: The runtime (s) of $\ell_1$ Ball Experiments

| $N(0, 1)$ | $10^5$ | $10^6$ | $10^7$ | $10^8$ |
|-----------|--------|--------|--------|--------|
| Sort + Scan | 7.0e-3 | 6.5e-2 | 7.2e-1 | 8.3    |
| (P)Sort + Scan | 3.4e-3 | 3.5e-2 | 3.8e-1 | 4.8    |
| (P)Sort + Partial Scan | 2.7e-3 | 3.6e-2 | 3.5e-1 | 4.5    |
| Michelot | 3.2e-3 | 3.3e-2 | 3.1e-1 | 4.2    |
| (P)Michelot | 8.0e-4 | 8.0e-3 | 9.5e-2 | 1.9    |
| Condat | 3.3e-4 | 3.7e-3 | 3.6e-2 | 5.5e-1 |
| (P)Condat | 2.2e-4 | 1.7e-3 | 1.7e-2 | 3.4e-1 |

| $N(0, 0.001)$ | $10^5$ | $10^6$ | $10^7$ | $10^8$ |
|---------------|--------|--------|--------|--------|
| Sort + Scan   | 7.1e-3 | 7.6e-2 | 7.0e-1 | 8.7    |
| (P)Sort + Scan | 3.4e-3 | 3.7e-2 | 3.9e-1 | 5.0    |
| (P)Sort + Partial Scan | 2.9e-3 | 3.5e-2 | 3.8e-1 | 4.6    |
| Michelot | 3.4e-3 | 3.4e-2 | 3.1e-1 | 4.1    |
| (P)Michelot | 3.4e-4 | 1.2e-2 | 1.0e-1 | 1.7    |
| Condat | 1.4e-3 | 6.5e-3 | 4.7e-2 | 6.1e-1 |
| (P)Condat | 4.1e-4 | 2.3e-3 | 1.9e-2 | 3.5e-1 |

[27] Lellmann, J., Kappes, J., Yuan, J., Becker, F., Schnörr, C.: Convex multi-class image labeling by simplex-constrained total variation. In: X.C. Tai, K. Mørken, M. Lysaker, K.A. Lie (eds.) Scale Space and Variational Methods in Computer Vision, pp. 150–162. Springer Berlin Heidelberg, 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). DOI 10.1109/TIT.2016.2555847

[29] 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)
Table 8: The runtime (s) of Parity Polytope Experiments

| Method                      | $n = 5 \times 10^5$ | $n = 10^6$ | $n = 5 \times 10^6$ | $n = 10^7$ |
|-----------------------------|---------------------|------------|---------------------|------------|
| Sort + Scan                 | 1.6e - 2            | 2.9e - 1   | 4.2                 |            |
| (P)Sort + Scan              | 1.4e - 2            | 2.6e - 1   | 3.4                 |            |
| Michelot                    | 1.7e - 2            | 2.3e - 1   | 2.7                 |            |
| (P)Michelot                 | 4.1e - 3            | 5.7e - 2   | 6.9e - 1            |            |
| Condat                      | 5.8e - 3            | 6.5e - 2   | 5.8e - 1            |            |
| (P)Condat                   | 2.7e - 3            | 2.1e - 2   | 1.7e - 1            |            |

Table 9: The runtime (s) of Weighted Simplex and Weighted $\ell_1$ Ball Experiments

| Method                      | $n = 5 \times 10^5$ | $n = 10^6$ | $n = 5 \times 10^6$ | $n = 10^7$ |
|-----------------------------|---------------------|------------|---------------------|------------|
| Sort + Scan                 | 7.2e - 2            | 1.2        | 7.6                 | 1.8e + 1   |
| (P)Sort + Scan              | 5.4e - 2            | 6.4e - 1   | 3.8                 | 1.0e + 1   |
| Michelot                    | 3.4e - 2            | 3.3e - 1   | 1.7                 | 3.6        |
| (P)Michelot                 | 1.3e - 2            | 1.0e - 1   | 4.0e - 1            | 7.4e - 1   |
| Condat                      | 7.1e - 3            | 6.3e - 2   | 2.7e - 1            | 5.8e - 1   |
| (P)Condat                   | 6.4e - 3            | 4.5e - 2   | 1.6e - 1            | 2.7e - 1   |

| Method                      | $n = 5 \times 10^5$ | $n = 10^6$ | $n = 5 \times 10^6$ | $n = 10^7$ |
|-----------------------------|---------------------|------------|---------------------|------------|
| Sort + Scan                 | 8.1e - 2            | 1.2        | 8.0                 | 1.8e + 1   |
| (P)Sort + Scan              | 5.3e - 2            | 5.8e - 1   | 4.0                 | 1.3e + 1   |
| Michelot                    | 4.1e - 2            | 3.6e - 1   | 2.0                 | 5.4        |
| (P)Michelot                 | 1.2e - 2            | 9.2e - 2   | 3.7e - 1            | 1.3        |
| Condat                      | 7.8e - 3            | 7.3e - 2   | 3.0e - 1            | 9.2e - 1   |
| (P)Condat                   | 6.0e - 3            | 4.9e - 2   | 2.0e - 1            | 4.7e - 1   |

[30] Perez, G., Ament, S., Gomes, C.P., Barlaud, M.: Efficient projection algorithms onto the weighted $\ell_1$ ball. CoRR abs/2009.02980 (2020). URL https://arxiv.org/abs/2009.02980

[31] Perez, G., Michel Barlaud, L.F., Régn, J.C.: A filtered bucket-clustering method for projection onto the simplex and the $\ell_1$ ball. Mathematical Programming 182, 445–464 (2020)

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

[33] Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) 58(1), 267–288 (1996). URL http://www.jstor.org/stable/2346178

[34] Tibshirani, R.: The lasso method for variable selection in the cox model. Stat. Med. (1997)
Table 10: The runtime (s) of solving $\min_x \|Ax - d\|_2$, s.t $\|x\|_1 \leq b$, where $A \in \mathbb{R}^{10 \times 10^6}$, $d \in \mathbb{R}^{10}$, and $b = 1$.

| Method                        | Total Runtime | Projection |
|-------------------------------|---------------|------------|
| $A \sim N(0, 1)$ & $b \sim U[0, 1]$ | 8.0           | 3.2        |
| Sort + Scan                   | 7.4           | 2.6        |
| (P)Sort + Scan                | 7.0           | 2.1        |
| Michelot                      | 5.7           | 1.8        |
| (P)Michelot                   | 4.5           | 1.3        |
| Condat                        |               |            |

[35] 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). DOI 10.1109/TSP.2019.2929944

[36] 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). DOI 10.1109/LCOMM.2015.2418261

[37] Zhang, G., Heusdens, R., Kleijn, W.B.: Large scale lp decoding with low complexity. IEEE Communications Letters 17(11), 2152–2155 (2013). DOI 10.1109/LCOMM.2013.101413.130826

[38] 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). DOI 10.1109/ISIT.2013.6620477
A Supplementary Proofs

Lemma A.1. Let $X|t$ denote the conditional variable such that $P_{X|t}(x) = P(X = x|X > t)$. If $X \sim U[l, u]$ and $l < t < u$, $X|t \sim U[t, u]$.

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

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

which implies $X|t \sim U[t, u]$. \hfill \Box

Lemma A.2. If $X, Y$ are independent, $X|t, Y|t$ are still independent.

Proof. $X, Y$ are independent iff $F_{X,Y}(x,y) = F_X(x)F_Y(y)$, which equals

$$P(X \leq x, Y \leq y) = P(X \leq x)P(Y \leq y).$$

Observe that,

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

So $X|t$ and $Y|t$ are still independent. \hfill \Box

Theorem A.3. If $d_1, ..., d_n$ i.i.d. $\sim U[l, u]$ and $l < t < u$, then $\{d_i | i \in I_t\}$ i.i.d. $\sim U[t, l]$.

Proof. First, from Lemma A.1, any $i \in I_t, d_i \sim U[t,l]$. Second, from Lemma A.2, any $i, j \in I_t, i \neq j, d_i, d_j$ are independent. So, $\{d_i | i \in I_t\}$ i.i.d. $\sim U[t, l]$. \hfill \Box

Theorem A.4. If $d_1, ..., d_n$ are i.i.d. with an arbitrary distribution $X$ whose PDF is $F_X$. Let $\delta_i := \begin{cases} 1, & \text{if } d_i > t \\ 0, & \text{otherwise} \end{cases}$, $S_n := \sum_{i=1}^n \delta_i$, and $p = 1 - F_X(t)$. Then $S_n \sim B(n, p)$.

Proof. Observe that $P(\delta_i = 1) = P(d_i > t) = p$, and $P(\delta_i = 0) = P(d_i \leq t) = 1-p$; thus $\delta_i \sim \text{Bernoulli}(p)$. Since $d_1, ..., d_n$ are independent, then $\delta_1, ..., \delta_n$ are independent. So, $\delta_1, ..., \delta_n$ are i.i.d. Bernoulli($p$); thus $S_n := \sum_{i=1}^n \delta_i \sim B(n, p)$. \hfill \Box

B Examples

Apply Theorem 3.2 to some examples that $d$ in different size and $d_i$ comes from different distributions.
(1) Let \( d_i \sim U[0, 1], n_1 = 10^5, n_2 = 10^6, t = 0.95. \)

(2) Let \( d_i \sim N(0, 1), n_1 = 10^5, n_2 = 10^6, t = 1.65. \)

(3) Let \( d_i \sim N(0, 10^{-3}), n_1 = 10^5, n_2 = 10^6, t = 1.65 \times \sqrt{10^{-3}} = 0.05218. \)

For (1), \( p = 1 - F_d(t) = 0.05. \) 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, \( P(|I_1^2| \in [4793, 5207]) = P(|I_2^2| \in [49347, 50654]) = 0.9973. \)

Meanwhile, due to \( f_d^*(x) = \frac{1}{0.05} e^{-\frac{x^2}{2}} = 20 \sqrt{2\pi} e^{-\frac{x^2}{2}}, \) for \( x \in [0.95, 1], \) we have \( E = 0.975 \) and \( V = 1.4800. \) As a result, applying Theorem 3.2, we gain:

\[
P(\tau_1 > t) \geq 0.99723, \quad \forall |I_1^2| \in [4793, 5207],
\]
\[
P(\tau_2 > t) \geq 0.99729, \quad \forall |I_2^2| \in [49347, 50654],
\]

which mean the number of active elements after projected the vector should be less than 5% of \( n_1 \) or \( n_2 \) with high probability.

For (2), \( p = 1 - F_d(t) = 0.05; \) similar to the first example,

\[
P(|I_1^2| \in [4793, 5207]) = P(|I_2^2| \in [49347, 50654]) = 0.9973.
\]

Together with

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

let’s calculate \( E \) and \( V: \)

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

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

\[
\implies V = E(x^2) - E^2 = 0.192.
\]

Based on \( E \) and \( V, \) applying Theorem 3.2,

\[
P(\tau_1 > t) \geq 0.99704, \quad \forall |I_1^2| \in [4793, 5207],
\]

\[
P_2(\tau > t) \geq 0.99728, \quad \forall |I_2^2| \in [49347, 50654],
\]

which imply that there only remains 5% active terms after projected with a probability closing to 1.

For (3), \( p = 1 - F_d(t) = 0.05; \) similar to previous examples,

\[
P(|I_1^2| \in [4793, 5207]) = P(|I_2^2| \in [49347, 50654]) = 0.9973.
\]
Together with
\[ f^*_d = \frac{20}{\sqrt{2\pi 10^{-3}}} e^{-\frac{x^2}{2 \times 10^{-3}}}, \quad \forall x \in [0.05218, +\infty), \]
let’s calculate the \( E \) and \( V \) as follows,
\[ E = \frac{20}{\sqrt{2\pi 10^{-3}}} \int_{1.65 \times \sqrt{10^{-3}}}^{\infty} xe^{-\frac{x^2}{2 \times 10^{-3}}} \, dx = 0.03234, \]
\[ E(x^2) = \frac{20}{\sqrt{2\pi 10^{-3}}} \int_{1.65 \times \sqrt{10^{-3}}}^{\infty} x^2 e^{-\frac{x^2}{2 \times 10^{-3}}} \, dx = 0.002187, \]
\[ \implies V = E(x^2) - E^2 = 0.001142. \]
As a result, applying Theorem 3.2, we gain:
\[ P(\tau_1 > t) \geq 0.99671, \quad \forall |I_1| \in [4793, 5207], \]
\[ P(\tau_2 > t) \geq 0.99724, \quad \forall |I_2| \in [49347, 50654], \]
which show there at most remains only 5% active entries after projected with a high probability.