Parallel Pareto Optimization for Subset Selection

Chao Qian[1,2], **Jing-Cheng Shi**[1], **Yang Yu**[1], **Ke Tang**[2], **Zhi-Hua Zhou**[1*]
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[2]UBRI, School of Computer Science and Technology,
University of Science and Technology of China, Hefei 230027, China
{qianc, shijc, yuy, zhouzh}@lamda.nju.edu.cn, ketang@ustc.edu.cn

Abstract

Subset selection that selects a few variables from a large set is a fundamental problem in many areas. The recently emerged Pareto Optimization for Subset Selection (POSS) method is a powerful approximation solver for this problem. However, POSS is not readily parallelizable, restricting its large-scale applications on modern computing architectures. In this paper, we propose PPOSS, a parallel version of POSS. Our theoretical analysis shows that PPOSS has good properties for parallelization while preserving the approximation quality: when the number of processors is limited (less than the total number of variables), the running time of PPOSS can be reduced almost linearly with respect to the number of processors; with increasing number of processors, the running time can be further reduced, eventually to a constant. Empirical studies verify the effectiveness of PPOSS, and moreover suggest that the asynchronous implementation is more efficient with little quality loss.

1 Introduction

Given a total set of n variables, the subset selection problem is to select a subset of size at most k for optimizing some given objective. One origin of this problem is the column subset selection problem [Gu and Eisenstat, 1996], which aims at selecting a few columns from a matrix that capture as much of the matrix as possible. Since then, subset selection has been significantly extended and numerous applications of it have emerged, e.g., feature selection, sparse learning, compressed sensing, etc.

Subset selection is NP-hard in general [Davis *et al.*, 1997]. Much effort has been put into developing polynomial-time approximation algorithms. These algorithms can be mainly categorized into two branches, greedy algorithms and convex relaxation methods. Greedy algorithms iteratively add or remove one variable that makes the given objective currently optimized [Gilbert *et al.*, 2003; Tropp, 2004]. Albeit widely used in practice, the performance of these algorithms is limited due to their greedy nature. Convex relaxation methods relax the original problem by replacing the set size constraint (i.e., the ℓ_0-norm constraint) with convex constraints, e.g., the ℓ_1-norm constraint [Tibshirani, 1996] and the elastic net penalty [Zou and Hastie, 2005]. However, the optimal solutions of the relaxed problem could be distant to that of the original problem.

Recently, Pareto optimization has been shown to be very powerful for the subset selection problem [Qian *et al.*, 2015c]. The Pareto Optimization for Subset Selection (POSS) method treats subset selection as a bi-objective optimization problem, which requires optimizing the given objective and minimizing the subset size simultaneously. Then, a bi-objective evolutionary algorithm with theoretical guarantee [Yu *et al.*, 2012; Qian *et al.*, 2015a; 2015b] is applied to solve it. Finally, the best solution satisfying the size constraint is picked out from the solution set produced by POSS. POSS is proved to achieve the best previous known polynomial-time approximation guarantee [Das and Kempe, 2011] on the sparse regression problem [Miller, 2002], a representative example of subset selection. Particularly, it can also even find an optimal solution on an important subclass of sparse regression [Das and Kempe, 2008]. In addition to the theoretical guarantee, POSS has also achieved significantly better empirical performance than the greedy and the relaxation methods.

POSS requires calling $2ek^2n$ ($e \approx 2.71828$ is Euler's number) number of objective function evaluations [Qian *et al.*, 2015c] to achieve a high quality solution, which could be unsatisfactory from the practical viewpoint when k and n are large. On the other hand, POSS is a sequential algorithm that cannot be readily parallelized, which hinders the exploration of modern computer facilities for applying POSS to large-scale real-world problems.

In this paper, we propose a parallel version of POSS, called PPOSS. Instead of generating one solution at a time (as in POSS), PPOSS generates as many solutions as the number of processors at a time, and can be easily parallelized. More important, on subset selection with monotone objective functions, we prove that, while preserving the solution quality,
(1) when the number of processors is limited (less than the number n of variables), the running time of PPOSS can be reduced almost linearly w.r.t. the number of processors;
(2) with increasing number of processors, the running time

*This work was supported by the NSFC (61333014, 61375061, 61329302), the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the 2015 Microsoft Research Asia Collaborative Research Program.

can be continuously reduced, eventually to a constant.

We further show that the obtained approximation quality reaches the best previous known approximation bound on two well-known instances in the studied class: any submodular objective function and sparse regression with a specific non-submodular objective function. Experimental results on sparse regression tasks verify our theoretical analysis, and suggest that the asynchronous implementation of PPOSS is more efficient, while introduces little loss in solution quality.

The rest of the paper is organized as follows. Section 2 introduces the subset selection problem and the POSS method. In Section 3, we propose the PPOSS method and give the theoretical analysis. Section 4 presents the empirical studies. Section 5 concludes this paper.

2 Subset Selection

The subset selection problem as presented in Definition 1 is to select a subset S from a total set V of variables such that a given objective f is minimized with the constraint $|S| \leq k$, where $|\cdot|$ denotes the size of a set. It is generally NP-hard [Natarajan, 1995; Davis *et al.*, 1997]. Note that we only consider minimization since maximizing f is equivalent to minimizing $-f$.

Definition 1 (Subset Selection)**.** *Given all variables $V = \{X_1, \ldots, X_n\}$, an objective f and a positive integer k, it is to find a set of at most k variables minimizing f, i.e.,*

$$\arg\min_{S \subseteq V} f(S) \quad \textit{s.t.} \quad |S| \leq k. \tag{1}$$

Sparse regression [Miller, 2002] as presented in Definition 2 is an example of subset selection. It is to find a sparse solution to the linear regression problem. Note that we do not distinguish between S and its index set $I_S = \{i \mid X_i \in S\}$ for notational convenience, and we assume that all variables are normalized to have expectation 0 and variance 1.

Definition 2 (Sparse Regression)**.** *Given all observation variables $V = \{X_1, \ldots, X_n\}$, a predictor variable Z and a positive integer k, define the* mean squared error *of a subset $S \subseteq V$ as*

$$MSE_{Z,S} = \min_{\boldsymbol{\alpha} \in \mathbb{R}^{|S|}} \mathbb{E}\left[(Z - \sum\nolimits_{i \in S} \alpha_i X_i)^2\right].$$

Sparse regression is to find a set of at most k variables minimizing the mean squared error, i.e.,

$$\arg\min_{S \subseteq V} MSE_{Z,S} \quad \textit{s.t.} \quad |S| \leq k.$$

In the previous theoretical analysis for sparse regression [Das and Kempe, 2008; 2011; Qian *et al.*, 2015c], an equivalent form is often used

$$\arg\max_{S \subseteq V} R^2_{Z,S} \quad \textit{s.t.} \quad |S| \leq k, \tag{2}$$

where $R^2_{Z,S} = (Var(Z) - MSE_{Z,S})/Var(Z)$ is the *squared multiple correlation* [Diekhoff, 1992; Johnson and Wichern, 2007], and can be simplified to be $1 - MSE_{Z,S}$, because Z is assumed to be normalized to have variance 1 (i.e., $Var(Z) = 1$). Das and Kempe [2011] proved that the greedy algorithm achieves the best known approximation guarantee. The greedy algorithm iteratively adds one variable with the largest R^2 improvement until k variables are selected. It can produce a subset S with $R^2_{Z,S} \geq (1 - e^{-\gamma}) \cdot OPT$, where OPT denotes the optimal function value of Eq. (2) and γ is the submodularity ratio [Das and Kempe, 2011].

2.1 The POSS Method

Qian et al. [2015c] proposed a Pareto optimization method for subset selection, called POSS. Let a binary vector $\boldsymbol{s} \in \{0,1\}^n$ represent a subset S of V, where $s_i = 1$ if the variable X_i is in S and $s_i = 0$ otherwise. Note that $\boldsymbol{s} \in \{0,1\}^n$ and its corresponding subset will not be distinguished for notational convenience. The POSS method reformulates the original problem Eq. (1) as a bi-objective minimization problem

$$\arg\min_{\boldsymbol{s} \in \{0,1\}^n} \quad (f_1(\boldsymbol{s}), f_2(\boldsymbol{s})),$$

where

$$f_1(\boldsymbol{s}) = \begin{cases} +\infty, & \boldsymbol{s} = \{0\}^n, \text{ or } |\boldsymbol{s}| \geq 2k \\ f(\boldsymbol{s}), & \text{otherwise} \end{cases}, \quad f_2(\boldsymbol{s}) = |\boldsymbol{s}|.$$

That is, POSS minimizes the original objective and the subset size simultaneously. Note that setting f_1 to $+\infty$ is to exclude trivial or overly bad solutions.

For comparing solutions in the bi-objective setting, both the two objective values need to be compared. For two solutions $\boldsymbol{s}$ and $\boldsymbol{s}'$, $\boldsymbol{s}$ *weakly dominates* $\boldsymbol{s}'$ (i.e., $\boldsymbol{s}$ is *better* than $\boldsymbol{s}'$, denoted as $\boldsymbol{s} \preceq \boldsymbol{s}'$) if $f_1(\boldsymbol{s}) \leq f_1(\boldsymbol{s}') \wedge f_2(\boldsymbol{s}) \leq f_2(\boldsymbol{s}')$ (i.e., $\boldsymbol{s}$ has a smaller or equal value on both the objectives); $\boldsymbol{s}$ *dominates* $\boldsymbol{s}'$ (i.e., $\boldsymbol{s}$ is *strictly better*, denoted as $\boldsymbol{s} \prec \boldsymbol{s}'$) if $\boldsymbol{s} \preceq \boldsymbol{s}'$ and either $f_1(\boldsymbol{s}) < f_1(\boldsymbol{s}')$ or $f_2(\boldsymbol{s}) < f_2(\boldsymbol{s}')$ (i.e., $\boldsymbol{s}$ has a strictly smaller value on one objective, and meanwhile has a smaller or equal value on the other objective). The domination relationship can be formally stated as follows:

- $\boldsymbol{s} \preceq \boldsymbol{s}'$ if $f_1(\boldsymbol{s}) \leq f_1(\boldsymbol{s}') \wedge f_2(\boldsymbol{s}) \leq f_2(\boldsymbol{s}')$,
- $\boldsymbol{s} \prec \boldsymbol{s}'$ if $\boldsymbol{s} \preceq \boldsymbol{s}' \wedge (f_1(\boldsymbol{s}) < f_1(\boldsymbol{s}') \vee f_2(\boldsymbol{s}) < f_2(\boldsymbol{s}'))$.

But if both $\boldsymbol{s}$ is not better than $\boldsymbol{s}'$ and $\boldsymbol{s}'$ is not better than $\boldsymbol{s}$, they are *incomparable*. Note that an isolation function $I : \{0,1\}^n \to \mathbb{R}$ was further introduced in [Qian *et al.*, 2015c] to determine if two solutions are allowed to be compared. However, I is set to be a constant function in this paper, and it can be just ignored because every solution has the same I value.

The procedure of POSS is described in Algorithm 1. It starts from the solution representing an empty set (line 1) and then iteratively tries to improve the solutions in the archive P (lines 3-12). In each iteration, a new solution $\boldsymbol{s}'$ is generated by randomly flipping bits of an archived solution $\boldsymbol{s}$ selected from the current P (lines 4-5); its two objective values are then evaluated (line 6); if $\boldsymbol{s}'$ is not strictly worse than any previously archived solution (line 7), it will be added into P, and meanwhile those previously archived solutions worse than $\boldsymbol{s}'$ will be removed from P (lines 8-9). After T iterations, the best solution (i.e., having the smallest f_1 value) satisfying the size constraint in P is selected (line 13).

For sparse regression, POSS using $\mathbb{E}[T] \leq 2ek^2n$ and a constant isolation function is proved to find a solution with $|S| \leq k$ and $R^2_{Z,S} \geq (1 - e^{-\gamma}) \cdot OPT$ [Qian *et al.*, 2015c], the best known polynomial-time approximation guarantee previously obtained by the greedy algorithm [Das and Kempe, 2011]. For a subclass of sparse regression called Exponential Decay [Das and Kempe, 2008], POSS with $\mathbb{E}[T] = O(k^2(n-k)n \log n)$ and a proper isolation function finds an optimal solution while the greedy algorithm cannot. Here, $\mathbb{E}[T]$ denotes the expected number of iterations.

Algorithm 1 POSS

Input: all variables $V = \{X_1, \ldots, X_n\}$, a given objective f and an integer parameter $k \in [1, n]$
Parameter: the number of iterations T
Output: a subset of V with at most k variables
Process:

1: Let $\boldsymbol{s} = \{0\}^n$ and $P = \{\boldsymbol{s}\}$.
2: Let $t = 0$.
3: **while** $t < T$ **do**
4: Select $\boldsymbol{s}$ from P uniformly at random.
5: Generate $\boldsymbol{s'}$ by flipping each bit of $\boldsymbol{s}$ with prob. $\frac{1}{n}$.
6: Evaluate $f_1(\boldsymbol{s'})$ and $f_2(\boldsymbol{s'})$.
7: **if** $\nexists \boldsymbol{z} \in P$ such that $\boldsymbol{z} \prec \boldsymbol{s'}$ **then**
8: $Q = \{\boldsymbol{z} \in P \mid \boldsymbol{s'} \preceq \boldsymbol{z}\}$.
9: $P = (P \setminus Q) \cup \{\boldsymbol{s'}\}$.
10: **end if**
11: $t = t + 1$.
12: **end while**
13: **return** $\arg\min_{\boldsymbol{s} \in P, |\boldsymbol{s}| \le k} f_1(\boldsymbol{s})$

3 Parallel Pareto Optimization

The proposed Parallel POSS (PPOSS) is shown in Algorithm 2, which simply modifies POSS to generate as many solutions as the number of processors in an iteration.

In each iteration of PPOSS, it first randomly selects a solution $\boldsymbol{s}$ from the current archive P (line 4), and then implements lines 6-7 in parallel. On each processor, it independently generates a new solution $\boldsymbol{s'}$ from $\boldsymbol{s}$ (line 6), and evaluates both the two objective values (line 7). Note that $\boldsymbol{s'_i}$ denotes $\boldsymbol{s'}$ at the i-th processor. After the procedure of parallelization, those newly generated solutions are used to update the archive P (lines 9-14), which makes P always maintain non-dominated solutions produced so far. Compared to POSS, PPOSS generates multiple new solutions in parallel in each iteration instead of generating only one new solution. When the number of processors N is 1, it is easy to see that PPOSS is just POSS.

In the following, we will theoretically analyze the performance of PPOSS on a large subclass of subset selection, maximizing a monotone function under the set size constraint. The studied class has the form

$$\arg\max_{S \subseteq V} f(S) \quad \textit{s.t.} \quad |S| \le k, \tag{3}$$

where f is monotone, i.e., $f(S_2) \ge f(S_1)$ for any $S_1 \subseteq S_2$. Note that the monotonicity is often satisfied by the objective function of subset selection, e.g., $R^2_{Z,S}$ in Eq. (2). Without loss of generality, we assume that f is normalized, i.e., $f(\emptyset) = 0$. In the following analysis, we will use the submodularity ratio as presented in Definition 3, which characterizes how close a set function f is to submodularity. f is submodular iff $f(S_2) - f(S_1) \le \sum_{X \in S_2 \setminus S_1} (f(S_1 \cup \{X\}) - f(S_1))$ for any $S_1 \subseteq S_2$ [Nemhauser *et al.*, 1978], which is equivalent to that $\gamma_{U,k}(f) \ge 1$ for any U and k. When f is clear, we will use $\gamma_{U,k}$ shortly.

Definition 3 (Submodularity Ratio [Das and Kempe, 2011])**.** *Let f be a non-negative set function. The submodularity ratio of f with respect to a set U and a parameter $k \ge 1$ is*

$$\gamma_{U,k}(f) = \min_{L \subseteq U, S: |S| \le k, S \cap L = \emptyset} \frac{\sum_{X \in S} (f(L \cup \{X\}) - f(L))}{f(L \cup S) - f(L)}.$$

Algorithm 2 Parallel POSS (PPOSS)

Input: all variables $V = \{X_1, \ldots, X_n\}$, a given objective f and an integer parameter $k \in [1, n]$
Parameter: the number of iterations T and the number of processors N
Output: a subset of V with at most k variables
Process:

1: Let $\boldsymbol{s} = \{0\}^n$ and $P = \{\boldsymbol{s}\}$.
2: Let $t = 0$.
3: **while** $t < T$ **do**
4: Select $\boldsymbol{s}$ from P uniformly at random.
5: **begin parallel** on N processors
6: Generate $\boldsymbol{s'_i}$ by flipping each bit of $\boldsymbol{s}$ with prob. $\frac{1}{n}$.
7: Evaluate $f_1(\boldsymbol{s'_i})$ and $f_2(\boldsymbol{s'_i})$.
8: **end parallel**
9: **for** each $\boldsymbol{s'_i}$
10: **if** $\nexists \boldsymbol{z} \in P$ such that $\boldsymbol{z} \prec \boldsymbol{s'_i}$ **then**
11: $Q = \{\boldsymbol{z} \in P \mid \boldsymbol{s'_i} \preceq \boldsymbol{z}\}$.
12: $P = (P \setminus Q) \cup \{\boldsymbol{s'_i}\}$.
13: **end if**
14: **end for**
15: $t = t + 1$.
16: **end while**
17: **return** $\arg\min_{\boldsymbol{s} \in P, |\boldsymbol{s}| \le k} f_1(\boldsymbol{s})$

3.1 Linear Speedup in the Number of Iterations

We prove the approximation bound of PPOSS in Theorem 1, where OPT denotes the optimal function value of Eq. (3) and $\mathbb{E}[T]$ denotes the expected number of iterations T. We can see that for finding a solution with the multiplicative approximation guarantee $1 - e^{-\gamma_{\min}}$, PPOSS achieves linear speedup in the number of iterations T when the number of processors N is asymptotically smaller than the number of variables n; the required T can reduce to be $O(1)$ when N is sufficiently large. The proof needs Lemma 1 that for any subset $S \subseteq V$, there always exists another variable, the inclusion of which can improve f by at least a quantity proportional to the current distance to the optimum. Our proof idea is inspired from that of Theorem 1 in [Qian *et al.*, 2015c].

Lemma 1. *For any $S \subseteq V$, there exists one variable $\hat{X} \in V \setminus S$ such that*

$$f(S \cup \{\hat{X}\}) - f(S) \ge \frac{\gamma_{S,k}}{k}(OPT - f(S)).$$

Proof. Let S^* be an optimal solution of Eq. (3), i.e., $f(S^*) = OPT$. From the definition of submodularity ratio (i.e., Definition 3), we can easily derive that

$$\sum_{X \in S^* \setminus S} (f(S \cup \{X\}) - f(S)) \ge \gamma_{S,k} \cdot (f(S \cup S^*) - f(S)).$$

Note that $|S^* \setminus S| \le k$ and $f(S \cup S^*) \ge f(S^*) = OPT$ by

monotonicity. Let $\hat{X} = \arg\max_{X \in S^* \setminus S} f(S \cup \{X\})$. Then,

$$f(S \cup \{\hat{X}\}) - f(S) \geq \frac{\gamma_{S,k}}{|S^* \setminus S|}(f(S \cup S^*) - f(S)) \geq \frac{\gamma_{S,k}}{k}(OPT - f(S)).$$ □

Theorem 1. *For maximizing a monotone function under the set size constraint (i.e., Eq. (3)), the expected number of iterations until PPOSS finds a solution $\boldsymbol{s}$ with $|\boldsymbol{s}| \leq k \wedge f(\boldsymbol{s}) \geq (1 - e^{-\gamma_{\min}}) \cdot OPT$, where $\gamma_{\min} = \min_{\boldsymbol{s}:|\boldsymbol{s}|=k-1} \gamma_{\boldsymbol{s},k}$, is*

(1) *if $N = o(n)$, then $\mathbb{E}[T] \leq 2ek^2n/N$;*

(2) *if $N = \Omega(n^i)$ for $1 \leq i \leq k$, then $\mathbb{E}[T] = O(k^2/i)$;*

(3) *if $N = \Omega(n^{\min\{3k-1,n\}})$, then $\mathbb{E}[T] = O(1)$.*

Proof. The theorem is proved by analyzing the increase of a quantity $J_{\max}$, which is the maximum integer value of $j \in [0, k]$ such that in the archive set P, there exists a solution $\boldsymbol{s}$ with $|\boldsymbol{s}| \leq j$ and $f(\boldsymbol{s}) \geq (1 - (1 - \frac{\gamma_{\min}}{k})^j) \cdot OPT$. That is, $J_{\max} = \max\{j \in [0, k] \mid \exists \boldsymbol{s} \in P, |\boldsymbol{s}| \leq j \wedge f(\boldsymbol{s}) \geq (1-(1-\frac{\gamma_{\min}}{k})^j) \cdot OPT\}$. As PPOSS starts from the solution $\{0\}^n$ (line 1 of Algorithm 2), the initial value of $J_{\max}$ is 0. Note that $J_{\max} = k$ implies that we have found a solution $\boldsymbol{s}$ in P satisfying that $|\boldsymbol{s}| \leq k$ and $f(\boldsymbol{s}) \geq (1 - (1 - \frac{\gamma_{\min}}{k})^k) \cdot OPT \geq (1 - e^{-\gamma_{\min}}) \cdot OPT$. Thus, we just need to analyze the expected number of iterations until $J_{\max} = k$.

Assume that $J_{\max} = i < k$. Let $\boldsymbol{s}$ be a corresponding solution with the value i, i.e., $|\boldsymbol{s}| \leq i$ and

$$f(\boldsymbol{s}) \geq (1 - (1 - \frac{\gamma_{\min}}{k})^i) \cdot OPT. \tag{4}$$

We first show that $J_{\max}$ cannot decrease. If $\boldsymbol{s}$ is kept in P, $J_{\max}$ obviously will not decrease. If $\boldsymbol{s}$ is removed, there must exist one newly generated solution $\boldsymbol{s'_i}$ which weakly dominates $\boldsymbol{s}$, and $\boldsymbol{s'_i}$ will be added into P (lines 11-12). Since $\boldsymbol{s'_i} \preceq \boldsymbol{s}$, we have $|\boldsymbol{s'_i}| \leq |\boldsymbol{s}| \leq i \wedge f(\boldsymbol{s'_i}) \geq f(\boldsymbol{s}) \geq (1 - (1 - \frac{\gamma_{\min}}{k})^i) \cdot OPT$. Thus, $J_{\max}$ must be at least i, i.e., $J_{\max}$ does not decrease.

We then show that $J_{\max}$ can always increase by at least $l \in [1, k - i]$ in one iteration with some probability. We know from Lemma 1 that flipping one specific 0-bit of $\boldsymbol{s}$ (i.e., adding a specific variable into the corresponding subset) can generate a new solution $\boldsymbol{s'}$, which satisfies that

$$f(\boldsymbol{s'}) - f(\boldsymbol{s}) \geq \frac{\gamma_{\boldsymbol{s},k}}{k}(OPT - f(\boldsymbol{s})).$$

Then, we have

$$\begin{aligned} f(\boldsymbol{s'}) &\geq (1 - \frac{\gamma_{\boldsymbol{s},k}}{k})f(\boldsymbol{s}) + \frac{\gamma_{\boldsymbol{s},k}}{k} \cdot OPT \\ &\geq (1 - (1 - \frac{\gamma_{\boldsymbol{s},k}}{k})(1 - \frac{\gamma_{\min}}{k})^i) \cdot OPT \\ &\geq (1 - (1 - \frac{\gamma_{\min}}{k})^{i+1}) \cdot OPT, \end{aligned}$$

where the second '$\geq$' is by Eq. (4), and the last one is by $\gamma_{\boldsymbol{s},k} \geq \gamma_{\min}$, which can be easily derived from $|\boldsymbol{s}| < k$ and $\gamma_{\boldsymbol{s},k}$ decreasing with $\boldsymbol{s}$. By sequentially implementing such a step l times, we can get a solution $\boldsymbol{s'}$ satisfying that

$$f(\boldsymbol{s'}) \geq (1 - (1 - \frac{\gamma_{\min}}{k})^{i+l}) \cdot OPT.$$

This also implies that such a solution can be found by flipping l specific 0-bits of $\boldsymbol{s}$ simultaneously in one iteration. Note that $|\boldsymbol{s'}| = |\boldsymbol{s}| + l \leq i + l$. Once such a solution $\boldsymbol{s'}$ is generated, no solution in the current P can dominate it; otherwise, $J_{\max}$ has already been larger than i, which contradicts with the assumption $J_{\max} = i$. Thus, $\boldsymbol{s'}$ will be included into P (line 12), and then $J_{\max} \geq i + l$. We then only need to analyze the probability of generating $\boldsymbol{s'}$ in one iteration. Note that it is sufficient to select $\boldsymbol{s}$ in line 4 and then flip those l specific 0-bits in at least one processor. Let $P_{\max}$ denote the largest size of P during the run of PPOSS. The probability of selecting $\boldsymbol{s}$ in line 4 is least $\frac{1}{P_{\max}}$ due to uniform selection. The probability of flipping l specific bits of $\boldsymbol{s}$ and keeping other bits unchanged in line 6 is $\frac{1}{n^l}(1 - \frac{1}{n})^{n-l}$. Because line 6 is implemented independently by each processor, the probability of generating $\boldsymbol{s'}$ is at least

$$\begin{aligned} &\frac{1}{P_{\max}} \cdot (1 - (1 - \frac{1}{n^l}(1 - \frac{1}{n})^{n-l})^N) \\ &\geq \frac{1}{P_{\max}} \cdot (1 - (1 - \frac{1}{en^l})^N) \geq \frac{1}{P_{\max}} \cdot (1 - e^{-\frac{N}{en^l}}), \end{aligned}$$

where the first '$\geq$' is by $(1 - \frac{1}{n})^{n-l} \geq \frac{1}{e}$ for $l \geq 1$ and the second one is by $(1 - \frac{1}{en^l})^{en^l} \leq \frac{1}{e}$.

Thus, we have shown that (1) $J_{\max}$ cannot decrease; (2) $J_{\max}$ can increase by at least l in one iteration with probability at least $\frac{1}{P_{\max}} \cdot (1 - e^{-\frac{N}{en^l}})$. According to these two points, the expected number of iterations to increase $J_{\max}$ by at least l (called one successful step) is at most $P_{\max}/(1 - e^{-\frac{N}{en^l}})$. Because $\lceil k/l \rceil$ successful steps are sufficient to make $J_{\max} = k$, the total expected number of iterations is

$$\mathbb{E}[T] \leq \lceil k/l \rceil \cdot P_{\max}/(1 - e^{-\frac{N}{en^l}}).$$

We then show that $P_{\max} \leq 2k$. From lines 9-14 of Algorithm 2, it is easy to see that any two solutions in the archive P must be incomparable. This implies that each value of one objective can correspond to at most one solution in P. Note that the second objective $|\boldsymbol{s}| \in \{0, 1, \ldots, 2k - 1\}$, because any solution with $|\boldsymbol{s}| \geq 2k$ has $+\infty$ value on the first objective and must be excluded from P. Thus, $P_{\max} \leq 2k$, and

$$\mathbb{E}[T] \leq 2k\lceil k/l \rceil/(1 - e^{-\frac{N}{en^l}}),$$

where $1 \leq l \leq k$. If $N = o(n)$, let $l = 1$ and then $e^{-\frac{N}{en}} = 1 - \frac{N}{en} + O((\frac{N}{n})^2) \approx 1 - \frac{N}{en}$. Thus, $\mathbb{E}[T] \leq 2ek^2n/N$. If $N = \Omega(n^i)$ for $1 \leq i \leq k$, let $l = i$ and then $1 - e^{-\frac{N}{en^i}} = \Theta(1)$. Thus, we have $\mathbb{E}[T] = O(k^2/i)$.

We finally prove claim (3) by analyzing the probability P_{opt} of generating an optimal solution in one iteration. As analyzed above, any solution $\boldsymbol{s}$ in P has $|\boldsymbol{s}| \leq 2k - 1$. Note that an optimal solution contains at most k number of 1-bits. Thus, the Hamming distance between any solution in P and an optimal solution is at most $HD = \min\{3k - 1, n\}$. Regardless of the selected solution in line 4, the probability of generating an optimal solution in one iteration at each processor is at least $\frac{1}{n^{HD}}(1 - \frac{1}{n})^{n-HD} \geq \frac{1}{en^{HD}}$. Because it is sufficient to generate an optimal solution in at least one processor, the probability P_{opt} is at least $1 - (1 - \frac{1}{en^{HD}})^N = \Theta(1)$, since $N = \Omega(n^{HD})$. Thus, $\mathbb{E}[T] \leq 1/P_{opt} = O(1)$. □

3.2 The Approximation Guarantee is Good

We have shown that PPOSS achieves linear speedup in the number of iterations T for finding a solution with the multiplicative approximation guarantee $1 - e^{-\gamma_{\min}}$. A natural question is then how good this approximation bound can be. In the following, we will show that it reaches the best previous known guarantee on two well-known instances of our studied problem class Eq. (3): f being any submodular function; the sparse regression problem Eq. (2) where $f = R^2$ is a specific non-submodular function.

When f in Eq. (3) is submodular, $\gamma_{\boldsymbol{s},k} \geq 1$ for any $\boldsymbol{s}$ and k. Thus, the obtained bound in Theorem 1 becomes $f(\boldsymbol{s}) \geq (1 - \frac{1}{e}) \cdot OPT$, which is optimal in general [Nemhauser and Wolsey, 1978], and is optimal even for the special case maximum coverage unless $P = NP$ [Feige, 1998].

For sparse regression, a special case with f being non-submodular, Das and Kempe [2011] proved that the greedy algorithm produces a subset S with $R^2_{Z,S} \geq (1 - e^{-\gamma_{S,k}}) \cdot OPT$, which is the best currently known approximation guarantee. Thus, the obtained bound in Theorem 1 indicates that PPOSS achieves nearly this best known one.

3.3 Almost Linear Speedup in the Running Time

In the above two subsections, we have shown that PPOSS with the number of iterations T decreasing linearly in the number of processors N can achieve a good approximation guarantee. Let t_{poss} and t_{pposs} denote the running time of each iteration for POSS and PPOSS, respectively. If $t_{poss} = t_{pposs}$, we can conclude that PPOSS achieves linear speedup in the running time. In the following, we will show that t_{poss} and t_{pposs} are close in expectation.

Let t_s, t_g, t_e and t_u denote the running time of selecting an archived solution $\boldsymbol{s}$ (line 4 of Algorithm 2), generating a new solution $\boldsymbol{s'}$ (line 6), evaluating the objective values of $\boldsymbol{s'}$ (line 7) and updating the archive P (lines 10-13), respectively. We first make some assumptions. Note that t_e usually depends on the number of 1-bits of $\boldsymbol{s'}$ (i.e., the number of selected variables), denoted by v. For the ease of analysis, we assume the linear dependence, i.e., $t_e = c \cdot v$, where c is a constant. We will not consider the extra parallel overhead in the analysis.

Theorem 2. *The expected difference between the running time of each iteration for POSS and PPOSS is*

$$\mathbb{E}[t_{pposs} - t_{poss}] \leq (N-1) \cdot t_u + c/2.$$

Proof. From the procedure of Algorithms 1 and 2, we have

$$t_{poss} = t_s + t_g + t_e + t_u;$$
$$t_{pposs} = t_s + \max\{t_g^i + t_e^i \mid 1 \leq i \leq N\} + N \cdot t_u,$$

where t_g^i and t_e^i denote t_g and t_e at the i-th processor, respectively. It is easy to see that t_g is fixed. Thus, $t_{pposs} = t_s + t_g + \max\{t_e^i \mid 1 \leq i \leq N\} + N \cdot t_u$. Then, we have

$$t_{pposs} - t_{poss} = \max\{t_e^i \mid 1 \leq i \leq N\} - t_e + (N-1)t_u$$
$$= c \cdot \max\{v_{pp}^i \mid 1 \leq i \leq N\} - c \cdot v_p + (N-1)t_u,$$

where v_p and v_{pp} denote the number of 1-bits of the newly generated solution $\boldsymbol{s'}$ for POSS and PPOSS, respectively.

We then analyze the expectation of $\max\{v_{pp}^i \mid 1 \leq i \leq N\}$ and v_p. Let u_p and u_{pp} denote the number of 1-bits of the selected solution $\boldsymbol{s}$ in line 4 for POSS and PPOSS, respectively. Note that $\boldsymbol{s'}$ is generated by flipping each bit of $\boldsymbol{s}$ with probability $1/n$. We can easily derive: $\mathbb{E}[v_p] = u_p + 1 - 2u_p/n$, $\mathbb{E}[v_{pp}^i] = u_{pp} + 1 - 2u_{pp}/n$, $Var(v_p) = Var(v_{pp}^i) = 1 - 1/n$. Note that $v_{pp}^1, v_{pp}^2, \ldots, v_{pp}^N$ are i.i.d. since each processor runs independently. By applying the upper bound of the expected highest order statistic [Gumbel, 1954; Hartly and David, 1954], we get

$$\mathbb{E}[\max\{v_{pp}^i | 1 \leq i \leq N\}] \leq \mathbb{E}[v_{pp}^i] + \sqrt{Var(v_{pp}^i)} \cdot \frac{N-1}{2N-1}.$$

Because the archive P contains at most one solution for each $|\boldsymbol{s}| = j < 2k$ (see the proof of Theorem 1) and $\boldsymbol{s}$ is selected from P uniformly at random, the distributions of u_p and u_{pp} have little difference. For the ease of analysis, we assume that they are the same, denoted as p. We can then get

$$\mathbb{E}[t_{pposs} - t_{poss}] = (N-1) \cdot t_u + c \cdot \sum\nolimits_j p(j) \cdot$$
$$(\mathbb{E}[\max\{v_{pp}^i \mid 1 \leq i \leq N\} \mid u_{pp} = j] - \mathbb{E}[v_p \mid u_p = j])$$
$$\leq (N-1) \cdot t_u + c \cdot \sum\nolimits_j p(j) \cdot \sqrt{1 - \frac{1}{n}} \cdot \frac{N-1}{2N-1}$$
$$\leq (N-1) \cdot t_u + c/2.$$

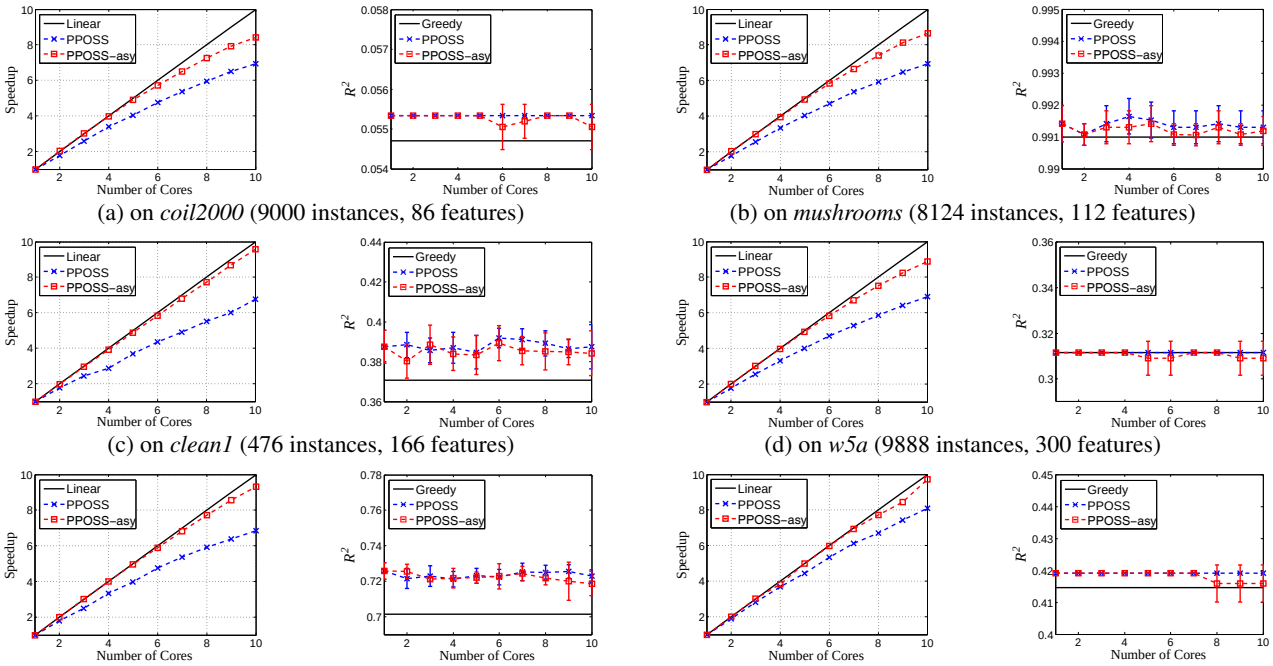
□

From lines 10-13 of Algorithm 2, we can see that t_u is mainly the running time of comparing $\boldsymbol{s_i'}$ with the solutions in P. The total number of comparisons is at most $2k$, because the largest size of P is $2k$. Note that comparing with the objective function evaluation time $t_e = c \cdot v$, the time of comparing solutions is usually much smaller, and can even be neglected. Thus, the difference $(N-1)t_u + c/2$ between t_{pposs} and t_{poss} is small compared to t_{poss} itself, and then PPOSS can almost achieve linear speedup in the running time.

3.4 Further Acceleration

As analyzed in Theorem 2, PPOSS needs the extra $(N-1)t_u$ running time (i.e., $2k(N-1)$ comparisons) to update the archive P. We then give an accelerating strategy. We put lines 10-13 of PPOSS into the parallelization process, and change $P_i = (P \setminus Q) \cup \{\boldsymbol{s_i'}\}$ to be $P_i = P \setminus Q$ for each processor. Let $R = \{\boldsymbol{s_i'} \mid \nexists \boldsymbol{z} \in P, \boldsymbol{z} \prec \boldsymbol{s_i'}\}$ record the newly generated solutions satisfying the condition of line 10. After parallelization, we compute the largest non-dominated subset of R, which contains the new solutions that should be kept, and it needs at most $|R|(|R|-1)/2$ comparisons. By combining it with $\cap_{i=1}^N P_i$ (the previously archived solutions which should be kept), we get the next archive P. Thus, this strategy decreases the number of comparisons from $2k(N-1)$ to $|R|(|R|-1)/2$. Note that $|R|$ is usually much smaller than N, because the solutions in P become better as the algorithm runs and the newly generated solutions are easily dominated. In our experiment, we use this accelerating strategy.

The extra running time of each iteration for PPOSS is actually due to the cost of synchronization. The term $c/2$ is because we need to wait until all the processors finish, and the extra number of comparisons $2k(N-1)$ (or $|R|(|R|-1)/2$)



(a) on *coil2000* (9000 instances, 86 features)

(b) on *mushrooms* (8124 instances, 112 features)

(c) on *clean1* (476 instances, 166 features)

(d) on *w5a* (9888 instances, 300 features)

(e) on *gisette* (7000 instances, 5000 features)

(f) on *farm-ads* (4143 instances, 54877 features)

Figure 1: On each data set, left: speedup, right: the average and standard deviation of the R^2 value (the larger the better).

is because we need to make P maintain non-dominated solutions produced so far. They can be avoided by running each processor asynchronously. The asynchronous PPOSS (called PPOSS-asy) performs lines 4-10 of Algorithm 1 at each processor asynchronously and independently, but shares an iteration counter. We will test its performance in the experiments, but the theoretical analysis is left in the future.

4 Experiments

Experiments for sparse regression (Eq. (2)) are conducted on 6 data sets[1] to investigate the performance of PPOSS. We set the sparsity $k = 8$, and for PPOSS, we use the setting suggested by Theorem 1: $T = \lfloor 2ek^2n/N \rfloor$. We test the number of cores N from 1 to 10. All of the experiments are coded in Java and run on an identical configuration: a server with 4 Intel(R) Xeon(R) CPU E5-2640 v2 (8 real cores each, 20M Cache, 2.00GHz, hyper-threading) and 32GB of RAM. The kernel is Linux 2.6.18-194.el5.

We compare the speedup as well as the solution quality measured by R^2 values with different number of cores N. The solution quality is also compared with that of the greedy algorithm (forward regression), which is the previous algorithm with the best known guarantee [Das and Kempe, 2011].

For PPOSS with each N value on each data set, we repeat 10 runs independently and report the average speedup and R^2 values. The results are plotted in Figure 1. From the left plots in the subfigure of each data set, we can observe that PPOSS (blue line) achieves speedup around 7 when the number of cores is 10. From the right plots in each subfigure, we can see that the R^2 values of PPOSS with different number of cores are stable, and better than that of the greedy algorithm except that they are the same on *w5a* data set. Note that on some data sets (e.g., *coil2000*), the standard deviation of the R^2 value by PPOSS is 0, which is because PPOSS always converges to the same good solutions in 10 runs.

We also test the performance of PPOSS-asy, as the red lines shown in Figure 1. In the run of PPOSS-asy, all the cores share an iteration counter, and terminate until the counter reaches $\lfloor 2ek^2n \rfloor$ (i.e., the number of iterations of POSS). As expected, PPOSS-asy achieves better speedup than PPOSS, because the extra cost of each iteration in the synchronous setting is avoided. Meanwhile, the R^2 values obtained by PPOSS-asy are slightly worse than that of PPOSS because of the miss-synchronization.

5 Conclusion

In this paper, we propose the parallel Pareto optimization method for subset selection (PPOSS). Theoretically, we prove that for a large subclass of subset selection, PPOSS has good properties for parallelization while preserving the approximation quality: when the number of processors is limited, almost linear speedups can be achieved; with increasing number of processors, the running time can be further reduced, eventually to a constant. Note that parallel greedy methods cannot enjoy many processors, since the number of greedy steps cannot be parallelizable. This implies that, given sufficient processors, PPOSS can be both faster and more accurate than parallel greedy methods. Empirical results verify our theoretical analysis, and also show that the asynchronous PPOSS can further improve the speedup with little loss of performance.

[1]The data sets are from `http://archive.ics.uci.edu/ml/` and `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

References

[Das and Kempe, 2008] A. Das and D. Kempe. Algorithms for subset selection in linear regression. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC'08)*, pages 45–54, Victoria, Canada, 2008.

[Das and Kempe, 2011] A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pages 1057–1064, Bellevue, WA, 2011.

[Davis *et al.*, 1997] G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approximation*, 13(1):57–98, 1997.

[Diekhoff, 1992] G. Diekhoff. *Statistics for the Social and Behavioral Sciences: Univariate, Bivariate, Multivariate*. William C Brown Pub, 1992.

[Feige, 1998] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[Gilbert *et al.*, 2003] A. C. Gilbert, S. Muthukrishnan, and M. J. Strauss. Approximation of functions over redundant dictionaries using coherence. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 243–252, Baltimore, MD, 2003.

[Gu and Eisenstat, 1996] M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996.

[Gumbel, 1954] E. J. Gumbel. The maxima of the mean largest value and of the range. *The Annals of Mathematical Statistics*, 25:76–84, 1954.

[Hartly and David, 1954] H. O. Hartly and H. A. David. Universal bounds for mean range and extreme observations. *The Annals of Mathematical Statistics*, 25:85–99, 1954.

[Johnson and Wichern, 2007] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson, 6th edition, 2007.

[Miller, 2002] A. Miller. *Subset Selection in Regression*. Chapman and Hall/CRC, 2nd edition, 2002.

[Natarajan, 1995] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

[Nemhauser and Wolsey, 1978] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[Nemhauser *et al.*, 1978] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical Programming*, 14(1):265–294, 1978.

[Qian *et al.*, 2015a] C. Qian, Y. Yu, and Z.-H. Zhou. On constrained Boolean Pareto optimization. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 389–395, Buenos Aires, Argentina, 2015.

[Qian *et al.*, 2015b] C. Qian, Y. Yu, and Z.-H. Zhou. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*, pages 2935–2941, Austin, TX, 2015.

[Qian *et al.*, 2015c] C. Qian, Y. Yu, and Z.-H. Zhou. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems 28 (NIPS'15)*, pages 1765–1773, Montreal, Canada, 2015.

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

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

[Yu *et al.*, 2012] Y. Yu, X. Yao, and Z.-H. Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, 180-181:20–33, 2012.

[Zou and Hastie, 2005] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.