An Accurate and Efficient Large-scale Regression Method through Best Friend Clustering

Kun Li  
Institute of Computing Technology,  
Chinese Academy of Sciences  
University of Chinese Academy of Sciences  
Beijing, China  
likungw@gmail.com

Yunquan Zhang  
Institute of Computing Technology,  
Chinese Academy of Sciences  
Beijing, China  
zyq@ict.ac.cn

Liang Yuan  
Institute of Computing Technology,  
Chinese Academy of Sciences  
Beijing, China  
yuanliang@ict.ac.cn

Gongwei Chen  
Institute of Computing Technology,  
Chinese Academy of Sciences  
University of Chinese Academy of Sciences  
Beijing, China  
gongwei.chen@vip1.ict.ac.cn

ABSTRACT
As the data size in Machine Learning fields grows exponentially, it is inevitable to accelerate the computation by utilizing the ever-growing large number of available cores provided by high-performance computing hardware. However, existing parallel methods for clustering or regression often suffer from problems of low accuracy, slow convergence, and complex hyperparameter-tuning. Furthermore, the parallel efficiency is usually difficult to improve while striking a balance between preserving model properties and partitioning computing workloads on distributed systems. In this paper, we propose a novel and simple data structure capturing the most important information among data samples. It has several advantageous properties supporting a hierarchical clustering strategy that is irrelevant to the hardware parallelism, well-defined metrics for determining optimal clustering, balanced partition for maintaining the compactness property, and efficient parallelization for accelerating computation phases. Then we combine the clustering with regression techniques as a parallel library and utilize a hybrid structure of data and model parallelism to make predictions. Experiments illustrate that our library obtains remarkable performance on convergence, accuracy, and scalability.

KEYWORDS
Large-scale Clustering, Parallel Regression, Distributed Machine Learning, Scalable Algorithm

1 INTRODUCTION
Machine Learning (ML) has become one of the crucial mainstays of information technology over past decades, albeit commonly hidden, part of modern life pervasively. As technologies based on machine learning like artificial intelligence (AI) rise prominently, the data size that researchers have to study with is also growing exponentially [12, 35]. This makes training time for models range from hours to week, which poses intense pressures across computation, networking, and storage. Today, High Performance Computing (HPC) and parallel optimization techniques catalyze the modern revolution in machine learning. More and more companies are turning to HPC as the solution for ML-based productivity and AI-enabled innovation, such as Google Cloud TPU, Amazon AWS AIHPC, and Microsoft Azure. Within the field of machine learning, clustering and regression are two fundamental and crucial techniques, which are distinguished as the representative unsupervised and supervised learning methods [17, 19, 33].

Clustering methods divide data into different groups by data attributes. The aim is that samples in the same group are similar to each other and different from those in other groups. Basically, various types of clusterings are distinguished as hierarchical (nested) versus partitional (unnested) methods, where Agglomerative Hierarchical Clustering (AHC) and K-Means are two typical methods that are widely used. AHC and its variants make a set of nested clusters organized as a hierarchical tree [15, 26, 38]. K-Means is simple and efficient on a variety of problems [16, 31, 36, 45]. Recently, Graph Clustering (GC) such as Minimum Spanning Tree clustering (MSTC) also stands prominently for detecting clusters with irregular boundaries by graph theory. It uses either the Prim’s algorithm or Kruskal’s algorithm to construct a minimum spanning tree (MST) first, sorts edges to remove inconsistent ones for connected components, and repeats until a threshold loops [25, 32].

Regression involves intensive computations in the model training phase inherently. A standard distributed method for regression...
is to map the computation on \( p \)-processor system evenly and then perform a global reduction regularly \([14, 40, 42]\). However, these distributed regression methods work poorly in scaling cases due to busy synchronization sweeps. Well-known approaches use Divide-and-Conquer (DC) algorithm to optimize the parallel process, such as DCKRR and DC-SVM \([29, 52]\). The basic idea of Divide-and-Conquer Regression (DCR) is to divide the data into \( p \) similar parts, generate \( p \) similar training models, and average \( p \) models for a final solution. The memory and computation overhead is reduced by DCR, while the accuracy is not guaranteed in various cases \([50]\).

More recent work is the parallel optimization for KRR: Balanced KRR v2 (BKRR2), which is the latest version by researchers \([49]\). BKRR utilizes K-Means to partition \( n \) samples to \( p \) clusters, where \( p \) is the number of processes. Each cluster contains \( n/p \) dispatched samples, and then \( p \) models are generated correspondingly. The regression results on distributed models are gathered for an average. Based on the BKRR, BKRR2 is then proposed to improve the accuracy by training models independently without a global reduction. Although BKRR2 proves that it could achieve higher accuracy and efficiency than the current fastest method \([49, 50]\), various problems are still not clearly addressed.

First, the accuracy and the convergence for K-Means are not always satisfactory on the large-scale dataset. It requires clustering iteration at least thousands of times for the experiments in BKRR2. Since the clustering center is changing dynamically, the volume of transferred data is massive in each iteration of the parallel clustering process for K-Means. BKRR2 also avoids confronting this problem directly and implement K-Means only with a process. Second, since the \( K \) is a hyperparameter to assign #clusters in K-Means, its value is simply set to #processes in BKRR2 and that is not robust. The clustering results deviate from the real distribution drastically as #processes change, which leads a poor accuracy in many cases. Even worse, the training time increases with more generated clusters assigned by increasing processes, which is essentially not adequate for large-scale training. Furthermore, BKRR2 achieves load-balance at the cost of accuracy. In the process of clustering, samples are traversed and dispatched to their nearest cluster center. However, a more relevant sample to some "full" cluster cannot be added to it for its late traversal order by the design of BKRR2.

In this paper, we design an efficient parallel regression library to address the pending problems in existing methods. First, we propose a new graph structure called Best Friend Graph to capture the most important information among the data samples. Based on the proposed structure, a novel clustering method, Best Friend Clustering (BFC), is presented. It reduces computation over AHC approaches, improves accuracy to K-Means methods, and enhances scalability than MST-based algorithms. Then we dig out the inherent properties of the graph structure and design a well-defined metric to determine the optimal aggregation.

Next, a balanced partition algorithm for data parallelism is proposed based on the idea of backtracking. The partition results reflect the real distribution of clusters to the fullest extent. Instead of deleting edges by a non-increasing order serially in MST-based work \([8, 25]\), we gather samples in the same group by swapping their pointers simply. The larger clusters are then split by backtracking and smaller clusters are merged from the optimal hierarchy to obtain total \( n/p \) samples in each process. Thus, the storage for edges is released, and the procedure could be parallelized easily.

Furthermore, Best Friend Clustering is applied to parallel regression. Although the above-proposed methods can be used separately, we combine them with multiple regression techniques as a complete parallel regression library. Each process generates single or multiple training models independently according to the clustering results. For each test sample, it selects the best one from these models straightforward in the prediction phase. Thus, model parallelism is achieved efficiently in each process. To be as general as possible, our library has no specific limits on the dimensionality of the dataset and the format of the distance measure.

At last, we present an experimental study on our Best Friend Clustering with three different clustering methods. Then the accuracy and scalability are analyzed for our regression library with state-of-the-art work. The encouraging performance demonstrates the distinct superiority of the proposed methods on distributed systems.

The main contributions in this paper include:

- A novel Best Friend Clustering method is proposed for large-scale clustering, which is accurate, fast, and parameter-free.
- Theoretical properties of the proposed clustering method are studied thoroughly, and we design a strategy to decide the optimal clustering aggregation through a well-defined metric based on analytical properties.
- We optimize the clustering on distributed systems, apply it to parallel regression, and propose a partition algorithm for load-balancing and spatial locality.
- By utilizing a hybrid structure of data and model parallelism, we combine the proposed methods with regression techniques as a parallel library. It shows a remarkable performance on convergence, accuracy, and scalability.

2 BACKGROUND

2.1 Regression Techniques

Regression is a supervised machine learning technique that is utilized to investigate the relationship between one or more predictors and response variables for a best-fit curve. Basically, there are various algorithms that are utilized to build a regression model, and in our paper, three extensively-used techniques are molded out and implemented in our library.

2.1.1 Linear regression. Linear regression is a widely-used method to find the linear relationship between the dependent variable and one or more independent variables by employing a straight line. Given a \( d \)-dimensional training sample \( x_i \) and the corresponding measured regressand \( y_i \), the objective in training phase is to find a \( w \) such that \( y_i = w^T \cdot x_i \). This can be formulated as a least squares problem in Equation 1 to find the optimal line by minimizing the sum of the residuals. Then for a test sample \( x^* \), we can utilize the training model to predict the regressand by \( y^* = w^T \cdot x^* \), and evaluate the accuracy with Mean Squared Error (MSE) method.

\[
 w_{opt} = \arg \min_w \frac{1}{2} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \tag{1}
\]

2.1.2 Kernel Ridge regression. To avoid overfitting and solve some ill-posed problems, L2 regularization with a positive parameter \( \lambda \) is
used in Equation 2, which is called ridge regression.

\[
\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \frac{1}{2} \lambda ||\mathbf{w}||^2
\]  

(2)

Moreover, kernel method is widely utilized to map samples to a high dimensional space using a nonlinear mapping. Thus the Kernel Ridge Regression (KRR) is presented by combining ridge regression with kernel method, and it learns model in high dimensional space for a better prediction accuracy. The solution vector \( \mathbf{\alpha} \) can be written in closed form in Equation 3,

\[
\mathbf{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \mathbf{y}
\]  

(3)

where \( \mathbf{K} \) is a n-by-n kernel matrix constructed by \( \mathbf{K}_{i,j} = \phi(x_i, x_j) \), \( \mathbf{y} \) is the corresponding n-by-1 regressand vector. Then \( \mathbf{\alpha} \) is employed to predict regressands in prediction phase as Equation 4.

\[
\mathbf{y}^* = \sum_{i=1}^{N} \mathbf{a}_i k(x^*, x_i)
\]  

(4)

2.2.3 Support Vector Regression. Support Vector Machine (SVM) can also be used as a regression method called Support Vector Regression (SVR), holding all the key features such as maximal margin that characterize the algorithm. Both KRR and SVR can learn a non-linear model by using kernel tricks, while they differ in the loss functions, i.e., ridge and epsilon-insensitive loss respectively. In the case of SVR, a margin of tolerance (epsilon) is set to the SVM, and we can tune it to gain the desired accuracy of our model. Moreover, slack variables \( \xi \) and \( \varepsilon \) can also be added to guard against outliers like SVM. The solution is obtained in Equation 5 that is subject to Equation 6, where \( \mathbf{w} \) is the magnitude of the normal vector to the surface that is being approximated, \( C \) is a tuneable regularization.

\[
\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{N} \xi_i + \varepsilon_i^*
\]

\[
y_i = \mathbf{w}^T \mathbf{x}_i \leq \varepsilon + \varepsilon_i^*
\]

\[
\mathbf{w}^T \mathbf{x}_i - y_i \leq \varepsilon + \xi_i
\]

\[
\xi_i, \varepsilon_i^* \geq 0
\]  

(5)

(6)

2.2 Clustering Methods

Many clustering methods are proposed in the literature to recognize the cluster of different characteristics. In this subsection, AHC, K-Means, and MSTC are introduced briefly.

2.2.1 Agglomerative Hierarchical Clustering. Algorithm 1 describes AHC formally. Generally AHC algorithm can produce a better-quality clustering result. However, the slow convergence speed, extensive computation operations, and expensive storage requirements make scaling problematic on a larger dataset. Moreover, since no global objective function is directly minimized, it remains to be determined when to stop merging, and the clustered decisions cannot be undone.

Algorithm 1 Agglomerative Hierarchical Clustering Algorithm

1: Let each sample be a cluster.
2: repeat
3: Merge the nearest two clusters as a new one.
4: Update the proximity matrix.
5: until Only one cluster remains.

2.2.2 K-Means. The basic steps are illustrated in Algorithm 2. K-Means is simple and efficient on a variety of problems. However, non-globular clusters or clusters of different densities and sizes cannot be solved by K-Means, and the outliers in data can also affect results. Furthermore, \( K \) value is a hyperparameter specified in advance, which determines the quality of clustering significantly.

Algorithm 2 K-Means Algorithm

1: Choose K initial sample as centroids.
2: repeat
3: Assign each sample to its nearest centroid to form K clusters.
4: Recompute K centroids.
5: until Centroids remains unchanged.

2.2.3 MST-based Clustering. The basic idea of MSTC is as follows. The algorithm can obtain a comparatively better result on clusters with irregular boundaries, while it is highly sequential and computationally intensive [8].

Algorithm 3 MST-based Clustering Algorithm

1: Compute and sort the pairwise distances.
2: repeat
3: Run MST algorithm to form clusters.
4: until k clusters.

3 METHOD

In this section, we first propose a simple, efficient, and accurate method for large-scale clustering. Section 3.1 and Section 3.2 provide the definition of Best Friend Graph and properties of this new method. They serve as the fundamental data structure of our method. Then a strategy to decide the optimal clustering aggregation through a well-defined metric is presented in Section 3.3. Next, we use a hybrid structure of data and model parallelism for parallel regression. Section 3.4 presents the detailed description of balanced partition for data parallelism. It achieves load-balance by utilizing merge and split operations on the hierarchical clustering structure. At last, the parallel regression method achieved with model parallelism is described in Section 3.5.

3.1 Best Friend Clustering

Clustering is intuitively inspired by the most fundamental social relation, friend circle, for detecting the potential groupings in the data. Existing clustering methods normally consider relationships between all pairs of samples. This leads to a slower convergence rate and a large overhead, which are not adequate for large-scale datasets on distributed systems. In reality, many of the relationships can be ignored. Our clustering method arises from the observation that people often make a decision with their best friends. Therefore, we simplify the friend cycle by only considering the most important best friend relationship.

A Best Friend Graph \( G(V, E) \) is defined on a dataset of \( n \) input samples for clustering. The vertex set \( V \) consists of these \( n \) data
elements. Each vertex $i \in V$ is associated with a directed best friend edge $(i, j)$ where the destination $j$ is its best friend, i.e. $i$’s nearest neighbor. Note that if there exist multiple nearest neighbors with the same distance to $i$, we simply choose the vertex with the smallest lexicographic order as the only best friend of $i$. Therefore, a Best Friend Graph $G(V, E)$ with $|V| = n$ vertices contains $n$ directed best friend edges. Figure 1 illustrates a tractable example of the Best Friend Graph. It contains top 12 city data by Global Cities Index [47] and 12 directed edges identifying the best friend relationships among them.

![Figure 1: Schematic diagram for Best Friend Graph with a case study of global cities.](image)

Based on the Best Friend Graph, a cluster is defined as a group of connected vertices. Evidently, the city samples are clustered into four groups by related friends in the first step. The clusters distinguished by Best Friend Clustering are consistent with the geographic taxonomies. For example, Los Angeles and San Francisco are recognized as the Western US, Chicago, Washington DC, and New York City are then classified as the Eastern US. Specifically, each new cluster is represented by a center point sample that is computed as the mean value of all samples in it. Let $T_k(V_k, E_k)$ be a connected subgraph $k$ in $G$ and $v_k = |V_k|$. The center point $\bar{x}_k$ representing $T_k$ is defined by Equation 7.

$$\bar{x}_k = \frac{1}{v_k} \sum_{x_i \in V_k} x_i \quad (7)$$

We can then build a new Best Friend Graph of all generated cluster centers in the previous step. In our example, the four clusters are combined into two groups as illustrated in Figure 2. We recursively apply this approach to the new clusters until there is only one cluster containing all the samples. The whole process converges quickly in logarithmic time as a cluster contains at least two samples in each hierarchy. Furthermore, the computation and communication time are reduced progressively with hierarchies.

![Figure 2: Schematic diagram for Best Friend Clustering with a case study of global cities.](image)

### 3.2 Properties

In this subsection, we analyze several crucial properties of Best Friend Graph, which are the basis to decide the optimal aggregation.

**Lemma 3.1.** There exists at least one directed cycle in a Best Friend Graph.

**Proof.** If we replace all directed edges with undirected edges, a Best Friend Graph becomes an undirected graph with $n$ vertices and $n$ edges. By induction, we can easily prove that it involves at least one undirected cycle. Transforming the edges on the cycle back to directed edges, we obtain a directed cycle. Otherwise, there must be one vertex $i$ associated with two best friend edges $(i, *)$ on the cycle, a contradiction to the best friend graph definition.

**Lemma 3.2.** Each weakly connected component of a Best Friend Graph contains one and only one directed cycle.

**Proof.** Each weakly connected component $G'(V', E')$ of a Best Friend Graph $G(V, E)$ is still a Best Friend Graph according to the definition. With Lemma 3.1, $G'$ contains at least one cycle. Thus we obtain $|E'| \geq |V'|$. We must have $|E'| = |V'|$, i.e. only one cycle in $G'$, otherwise will get $|E| > |V| = n$, contradicting the Best Friend Graph definition.

**Lemma 3.3.** The edge weights on a directed path of a Best Friend Graph are non-increasing.

This is obvious from the definition. For example, for the directed path $i \rightarrow j \rightarrow k$, we have $\omega(i, j) \geq \omega(j, k)$, otherwise $j$’s best friend should be $i$ rather than $k$.

**Lemma 3.4.** The length of a directed cycle in a Best Friend Graph is two and the weights of edges on a cycle are identical.

**Proof.** Assume there is only one nearest neighbor to each vertex, i.e. the weight $\omega$ of a best friend edge $(i, j)$ is strictly smaller than the distance between $i$ and other vertex $\omega(i, j) < \omega(i, k), k \neq i, j$. If there exists a directed cycle involving more than two vertices, e.g. $i \rightarrow j \rightarrow k \rightarrow i$, we have $\omega(i, j) > \omega(j, k) > \omega(k, i) = \omega(i, k)$. Thus the best friend edge of $i$ should be $(i, k)$, a contradiction.

Otherwise, we have $\omega(i, j) \geq \omega(j, k) \geq \omega(k, i) = \omega(i, k)$. Since $(i, j)$ is a best friend edge, we obtain $\omega(i, j) = \omega(i, k)$ and $\omega(i, j) = \omega(j, k) = \omega(k, i)$. According to the smallest lexicographic order rule in Best Friend Graph definition, we get an inconsistent lexicographic order $j < k < i < j$.

**Corollary 3.1.** Starting from any vertex and traversing along the Best Friend Graph will enter the cycle.

**Definition 3.1.** A best friend forest $F(V, E)$ of a Best Friend Graph $G(V, E)$ is defined by replacing all directed edges with undirected ones and removing one edge on each cycle.

**Theorem 3.1.** A connected component, a tree $T(V_T, E_T)$ of a Best Friend Graph $G(V, E)$ is defined by replacing all directed edges with undirected ones and removing one edge on each cycle.

**Proof.** Let $T^*(V_T, E_T)$ be the MST of the complete graph $G(V_T, E)$. For an edge $(i, j) \in E_T - E_r$, we obtain an undirected path in $T(V_T, E_T)$. According to Corollary 3.1, the corresponding directed path connecting $i$ and $j$ in the Best Friend Graph is one of the following three cases: $i \rightarrow \cdots \rightarrow j$, $i \leftarrow \cdots \leftarrow j$ and $i \rightarrow \cdots \rightarrow j$.
As discussed in Theorem 3.1, Best Friend Clustering specifies a MST. Repeat the process for all edges in (i,j) in $E_T$, we will obtain $T$ with a same weight to $T^*$. Thus $T(V_T,E_T)$ is also a MST.

Figure 3 shows the corresponding best friend forest of the example in Figure 1. With Theorem 3.1, each connected component in Figure 3 is a minimum spanning tree, and the graph is a minimum spanning forest $F(V,E)$.

3.3 Optimal Aggregation

The target of clustering is expected to make high intracluster compactness and intercluster dispersion [4]. Since a set of clustering hierarchies are built, we turn to choose an optimal clustering level as the input to our regression model by using a rational metric. As discussed in Theorem 3.1, Best Friend Clustering specifies a minimum spanning forest for each hierarchy intrinsically, and it connects all scattered clusters as a whole network. Thus, the clustering validity is analyzed based on the MST network, where metrics are quantified by the properties of MST.

**Definition 3.2.** Let a $T_i = (V_i, E_i)$ denote a minimum spanning tree in a best friend forest $F$. The intracluster compactness $c_i$ for $T_i$ is defined as:

$$c_i = \frac{1}{e_i} \sum_{(j,k) \in E_i} \omega(j,k), \quad (8)$$

where $e_i = |E_i|$.

Algorithm 4 shows the procedure of the best friend forest construction and the distance calculation of each connected component. The first function finds all best friend edges and records them in a global array. The second function traverses all connected components. It finds an unvisited node as a root of a MST and feeds it structure and the distance calculation of each connected component.

**Algorithm 4 Calculating Compactness**

Require: $\text{visited}[] = 0$, $c[] = 0$, $e[] = 0$, $\text{mst_num} = 0$

1. function **INITIALIZEBESTFRIENDFOREST**
2. for $i = 0 \rightarrow n - 1$ do
3. $j = \text{findBestFriend}(i)$
4. $\text{addEdge}(i, j, \omega(i, j))$
5. end for
6. end function
7. function **TRAVESEMST**
8. for $i = 0 \rightarrow n - 1$ do
9. if $\text{visited}[i]$ then
10. $\text{SEARCH}(i)$
11. $c[\text{mst_num}] = e[\text{mst_num}]$
12. $\text{mst_num} = \text{mst_num} + 1$
13. end if
14. end for
15. end function
16. function **SEARCH**($i$)
17. $\text{visited}[i] = 1$
18. $e[\text{mst_num}] = 1$
19. while $\text{getNextNeighbor}(i, &k)$ do
20. if $\text{visited}[k]$ then
21. $c[\text{mst_num}] = e[\text{mst_num}]$
22. $\text{SEARCH}(k)$
23. end if
24. end while
25. end function

Figure 3: Best friend forest with a case study of global cities.

$k \leftarrow \cdots \leftarrow j$. Since adding $(i,j)$ to it $T(V_T,E_T)$ leads to a cycle, there must be an edge $(u,v)$ in the path that does not belong to $T^*(V_T,E_T)$. In any of the three path cases, we have $\omega(i,j) \geq \omega(u,v)$ according to Lemma 3.3. We must have $\omega(i,j) = \omega(u,v)$, otherwise we will get spanning tree with a smaller weight by replacing $(i,j)$ with $(u,v)$. This contradicts the assumption that $T^*(V_T,E_T)$ is a MST. Repeat the process for all edges in $(i,j)$ in $E_T$, we will obtain $T$ with a same weight to $T^*$. Thus $T(V_T,E_T)$ is also a MST.

Definition 3.3. Let a best friend forest $F_k$ be the $k$th hierarchy produced by Best Friend Clustering. Assume that there are $m$ clusters $T_1, T_2, ..., T_m$ in $F_k$, and cluster $T_i$ contains $c_i$ samples. Then the intercluster dispersion $d_i$ for $T_i$ is defined as:

$$d_i = \min(d(X_j, X_i)) | 1 \leq j \leq m, j \neq i), \quad (9)$$

where $X_j$ and $X_i$ are the new cluster centers and $d(X_j, X_i)$ is the Euclidean distance between cluster $T_i$ and $T_j$.

The following metric combines the intracluster compactness $c_i$ and the intercluster dispersion $d_i$. It serves to determine the optimal clustering level.

**Definition 3.4.** Let MSTs $T_1, T_2, ..., T_m$ denote clusters in a best friend forest $F_k$ for the $k$th dendrogram hierarchy produced by Best Friend Clustering. Then the $\text{HCI}(k)$ is defined as a linear combination of the intracluster compactness and intercluster dispersion:

$$\text{HCI}(k) = \frac{1}{m} \sum_{i=1}^{m} \left( \frac{d_i - c_i}{d_i + c_i} \right), \quad (10)$$

and the optimal cluster number is:

$$k_{opt} = \arg \max \{\text{HCI}(k)\}. \quad (11)$$

With scaled distance denoted on each edge, hierarchical MST results are depicted in Figure 4 for the first two clustering levels. We have $c_1 = 1, c_2 = (3 + 5 + 4 + 7)/4 = 4.75, c_3 = 2, c_4 = (2 + 1)/2 = 1.5$ in $S_1$, and $c_1 = 30, c_2 = 16$ in $S_2$. Since similar vertices are merged
iteratively, an important observation is that the $d_i$ for cluster $i$ in $S_k$ is exactly the best friend weight for vertex $i$ in $S_{k+1}$. For instance, $d_1 = d_2 = 30, d_3 = d_4 = 16$ are obtained in $S_1$ by the information acquired in $S_2$. Thus we have $HCI(1) = 0.82$ and $HCI(2) = 0.51$, and the clustering result of $S_1$ is evaluated better than $S_2$.

### 3.4 Balanced Partitions

Based on the profiling results, we observe that the partitions by clustering are typically irregular and imbalanced. This makes computing nodes load-imbalanced, and thus we need to devise a new partition algorithm to achieve data parallelism. In our design, a balanced partition on $p$ computing nodes means the number of samples on each node is close to $n_p = n/p$. Based on the data organization of clustering result, we propose a balanced partition algorithm by utilizing a backtracking mechanism, which is composed of MERGE and SPLIT operations.

MERGE is performed on the piecemeal clusters with small sizes. Figure 5 (a) shows a case with 4 processes on the clustering result $S_1$. Since we have $n_p = 12/3 = 4$, the sizes of cluster $C_3$ and $C_4$ are too small compared to $n_p$. Therefore, we sort the $S_1$ by cluster size, and merge the small clusters into the same node for a total size close to $n_p$ on it. It is worth noting that the models are still trained independently on each node, which means the objective of MERGE is to make up the $n_p$ instead of mixing models on different clusters. In Figure 5 (a), $C_3$ and $C_4$ are merged into process 3 as group $G_3$ and $G_4$. The total sizes are 3, 5, and 4 respectively on process 1 to 3, which achieves a balanced partition for $S_1$.

SPLIT is utilized to separate the large clusters of which the size is much larger than $n_p$. Since the sample pointers in same sample pointers are moved together in each iteration, we achieve SPLIT operation by backtracking mechanism based on the clustering array structure. A case with 5 processes on $S_2$ is illustrated in Figure 5 (b). In this case, we have $n_p = 12/5 = 2.4$ while the original cluster $C_3$ is larger than $n_p$ apparently. Thus we perform backtracking on the $S_2$ by the tracks of pointers. The backtracked $S_2$ contains 2 new split clusters, where the size of $C_3$ is still large than $n_p$. Then we perform another SPLIT to separate $C_3$ into $G_3$ and $G_4$ respectively. Therefore, the group $G_1$ to $G_3$ are dispatched to nodes evenly, and they are trained as independent models. By utilizing the backtracking mechanism, a principle is followed that closer samples are always guaranteed to gather together after the SPLIT operation, and spatial locality is also exploited simultaneously.

### 3.5 Independent Prediction

Parallel regression methods normally construct $p$ independently models, where $p$ is the number of processors (hardware parallelism). They often take $p$ into consideration from the beginning. Two major disadvantages exist with this approach.

First, $p$ is essentially irrelevant to the input data and may mismatch the intrinsic structure of data samples. Our method employs the Best Friend Graph hierarchically and efficiently constructs a series of cluster levels that does not depend on any predefined value. Second, existing methods may require a data reorganization to improve the load-balance. However, this procedure may lose the relationship information among the data that is moved from one cluster to others and hurts the compactness of the final models.

**Algorithm 5 Parallel Regression**

```plaintext
1. Require: $n$ samples for training, $k$ samples for testing, best-clustered results $A$
2. $p \leftarrow$ rank of a process
3. Initialize best Mean Squared Error $MSE^* \leftarrow \infty$
4. Balanced Partitions on $n$ samples in $A$ for $AP$
5. \textbf{function} REGRESSION
6. \hspace{1em} \textbf{Training}(AP)
7. \hspace{2em} Reduce: $MSE = (\sum_{p=1}^{P} e^p)/k$
8. \hspace{2em} \textbf{if} $(p = 0) \&\&(MSE < MSE^*)$ \textbf{then}
9. \hspace{3em} $MSE^* = MSE$
10. \hspace{1em} \textbf{end if}
11. \textbf{end function}
12. \textbf{function} \textbf{Training}(AP)
13. $C^p \leftarrow$ Cluster Number in $AP$ on Rank $p$
14. \hspace{1em} for $i = 0 \rightarrow C^p$ \textbf{do}
15. \hspace{2em} Training Model $M^p_i$ for Cluster $i$ on Rank $p$
16. \hspace{1em} \textbf{end for}
17. \textbf{end function}
18. \textbf{function} \textbf{Testing}(MP)
19. Initialize local MSE $e^p \leftarrow 0$
20. \hspace{1em} for $i = 0 \rightarrow k - 1$ \textbf{do}
21. \hspace{2em} if FindNearCluster(i) = $C^*$ \textbf{then}
22. \hspace{3em} Select best $M^*$ for prediction
23. \hspace{3em} Update $e^p$
24. \hspace{2em} \textbf{end if}
25. \hspace{1em} \textbf{end for}
26. \textbf{end function}
```

Unlike the existing work [49, 52], model parallelism is utilized in our work for making predictions independently on each process. For a given test sample $x^*$, we only use the corresponding model $M^*$ to perform a prediction if its closest cluster center is $C^*$ on process $p$. Instead of conducting communication regularly, the errors are accumulated on each node first, and only a Reduce operation is required at last to make statistical analyses. Since an intact message is cheaper than scattered messages in MPI communication, the latency overhead is further reduced by this optimization. The parallel regression is summarized in Algorithm 5 formally.

## 4 IMPLEMENTATION

### 4.1 Parallelization

To reduce the computation, memory, and communication overheads, we design an efficient parallel implementation of our method. The parallelization of the first function of Algorithm 4 is straightforward. The data samples are evenly distributed to all processors and the calculation is parallelized accordingly. The distribution of the example is shown in Figure 6 where each process is dispatched with 3 samples for computing their nearest neighbors respectively.

The traversal of the best friend forest and the calculation of compactness, i.e. the second function of Algorithm 4 seems to be an inherently serial task. However, provided with Lemmas in the previous section, we are able to identify individual components (trees) with the edge information in the best friend forest. With
Algorithm 6 Paralllelization of TRAVERSEMST in Algorithm 4

1: Gather the whole Best Friend Forest
2: hash.initialize()
3: mstNum = 0
4: for \( i = 0 \) to \( n - 1 \) do
5:    \( j = \text{getBestFriend}(i) \)
6:    if hash.exist((i, j)) then
7:        startNode[mstNum] = i
8:        mstNum++
9:    else
10:        hash.add((i, j))
11:    end if
12: end for
13: parfor \( i = 0 \) to \( mstNum - 1 \) do
14:    Search(startNode[i])
15: end parfor

Lemma 3.2, we know that each cycle identifies a tree, and with Lemma 3.4 a cycle is easy to find by searching its two equal edges.

Therefore, all the best friend edges are gathered to all processors. Every processor then finds all pairs of equal edges by sorting all \( |V| \) edges or using a hash method. The number of trees in the best friend forest is the number of pairs of equal edges. Each processor traverses a set of trees and the traversal of each tree starts with either of the two nodes on its cycle. Overall, it only transfers short messages that only contain two values: the pairs of nearest neighbors and the corresponding shortest distances in our implementation. Algorithm 6 provides the parallel version of the second function TRAVERSEMST in Algorithm 4.

4.2 Vectorization

The widely-used Euclidean distance is employed for distance calculation in our work. The process of finding the nearest neighbor dominates the overheads of the computation. To further leverage the ability of vector processing units in modern CPUs, we group \( ol \) data samples in a vector register and perform \( ol \) calculations in a SIMD style, where \( ol \) is the vector length. This improves the computation efficiency significantly.

Figure 5: Balanced partitions to achieve load balancing on each process. Figure 5 (a) describes the MERGE for small tasks while Figure 5 (b) shows the SPLIT for large tasks based upon backtracking mechanism. Where appropriate, abbreviated forms are used for city names.

Figure 6: Data organization in Best Friend Clustering. Clusters are distinguished by colors and boxes in two steps respectively.

4.3 Data Organization

Provided with the load-balancing scheme described in Section 3.4, we design a simple data organization method to guarantee the SPLIT operations produce partitions with good compactness. To this end, sample pointers of the same cluster are swapped to stay together after calculating each clustering level. Except the first clustering level, each point actually represents a cluster. The relative positions of points in a low-level cluster are fixed, i.e. they are swapped as a whole big point. This data organization is critical to balanced partition since it guarantees that the closer samples are in the array, the more similarities match. For example in Figure 6, although London, Paris, and Hong Kong are clustered into the same group in the second \( S_2 \) hierarchy, London and Paris contain more similarities as they are far away from Hong Kong in the clustering array.

5 EXPERIMENTS

5.1 Setup

Table 1: Description of Classic Shape Datasets.

| Dataset          | #Samples | #Labels | #Dimensions |
|------------------|----------|---------|-------------|
| Zahn’s Compound  | 399      | 6       | 2           |
| Aggregation      | 788      | 7       | 2           |
| R15              | 600      | 15      | 2           |

Platforms. We develop the library in C++ and our experiments are performed on a high-performance cluster. Each machine of the cluster is composed of two Intel Xeon Platinum 9242 processors...
with 2.30 GHz clock speed (turbo boost frequency of up to 3.80 GHz), which owns 96 physical cores organized into two sockets. The processor contains a 71.5 MB smart cache. AVX512 instruction set extension is supported and it’s able to conduct operations for 8 double-precision floating-point data in a SIMD manner.

Baselines. The experiments are conducted in two parts. First, the proposed Best Friend Clustering is evaluated with three classic 2D datasets. Results of KMeans [50], AHC [39] and MSTC [34] methods are also presented as different baselines for comparison. Then we turn to the evaluation on convergence, accuracy, and scalability for our regression library with five real large-scale datasets. Since DCKRR [53] and BKRR2 [49] are two closely related papers, they are employed as two baselines in this paper. Moreover, we alternate the clustering methods in BKRR2 with AHC [39] and MSTC [34] as AHCKRR and MSTKRR respectively. Thus the experimental configurations for parallel regression cover the whole spectrum of representative clustering methods: BFCKRR (Best Friend Clustering), BKRR2 (K-Means), AHCKRR (Agglomerative Hierarchical Clustering), and MSTKRR (Minimum Spanning Tree Clustering).

Datasets. Three classic shape datasets are employed to demonstrate the quality of clustering in Table 1, which can be obtained in the Clustering Basic Benchmark (CBB) [23]. They represent well-understood clustering problems and are widely-used benchmarks for checking the applicability of clustering algorithms [23, 24, 43, 51]. Then Datasets Million Song Data (MSD) and Cadata are used as two of our evaluated datasets for regression since they are both used in the paper of DCKRR and BKRR2 [49]. To further justify the scaling efficiency of our approach, we use another three real datasets, which contain the data on higher dimensions and interdisciplinarity research.

The details for HCIs and #clusters are given in Table 3 and Table 4 respectively. Since the connected samples generated by our clustering method are grouped together, BFC could reach a fast convergence in O(logN) rounds. The details for HCIs and #clusters are given in Table 3 and Table 4 respectively, where the best convergent hierarchies are highlighted. The average sizes for each cluster are 56, 4, 15,000, 453, and 2,379 respectively on five datasets. Moreover, the number of iterations still falls in low single digits on large-scale datasets like Gas Sensor, which illustrates that the clustering is less sensitive to the increasing training size and appropriate for scaling cases.

### 5.4 Accuracy

The proposed balanced partition strategy has little impact on the accuracy of the model since it reserves the samples with most similarities in same cluster. Nonetheless, to assure the correctness of our implementation, we perform the configurations scaling from 96 to 12,288 cores and compare with the reported accuracy (measured by MSE). To give a fair comparison, best parameters were finely tuned from the same parameter set to achieve the lowest MSE in different methods. As shown in Figure 8, BFCKRR achieves the lowest MSEs when KRR techniques are employed. Moreover, BFCLR and BFCSVR could also make a superior prediction in most cases. Among all considered baselines, DCKRR and BKRR2 produce a poor quality, which adopts no samples clustering and K-means clustering respectively. As the core increases, high accuracy is obtained steadily by our library while the MSEs of other baselines ripples drastically. This illustrates that our balanced partition algorithm adds great support to accuracy in scaling cases.

### 5.5 Scalability

Figure 9 illustrates the scalability for different methods on five datasets. We observe that our library decreases regularly as we double the number of cores, K-means in most cases. Compared with three methods mentioned above, we can observe that BFC maintains the merges quite well and clusters these data better than the considered baselines.
while both DCKRR and BKRR2 exhibit a bad scaling performance by a growing curve. This illustrates that the K-means-based or DC-based methods are poor in large-scale regression, where performances are jeopardized cumulatively by the increasing $K$ value and expensive DC operations. With similar parallel implementation achieved, MSTKRR outperforms BKRR2 in most cases. However, AHCKRR suffers from the lowest performance as agglomerative style algorithms come with a quadratic time complexity inherently.

To dissect the procedures of our library and see how it varies as the size of datasets grows, we provide a quantitative look into the cases of two typical datasets, i.e., the smallest and largest ones. Figure 10 compares the logarithmic time for clustering I/O, clustering, regression I/O, regression, and communication in our library. Upon inspection it becomes distinct that the increasing cores also aggravate the communication and I/O cost. Despite the scaling pressure brought by communication and I/O, our library still obtains a sustained scaling performance. Table 5 shows the analytical proportion and speedup of two datasets. Interestingly, although the average proportion on different datasets contains little difference, our library can obtain a higher speedup on Gas Sensor. Based on conjoint analyses on Figure 10, we can observe that the key contribution lies in the better scaling efficiency of parallelizable procedures (clustering and regression parts) on large-scale datasets.

6 RELATED WORK

Our paper shows closely concerns for two main lines of research. The first thread is to study the efficient clustering methods with better-quality on distributed systems. Basically, AHC and K-Means are two extensively used methods [41]. Bahmani et al. extended the k-means problems by a MapReduce algorithm on distributed nodes [5]. Ene et al. optimized the greedy algorithm on MapReduce to solve the k-center problems and the local search strategy to address the k-median problems [21]. Subsequently, several papers studied similar problems with the MapReduce model [2, 6, 7]. Then Bouguettaya et al. built a hierarchy based on a group of centroids generated by K-Means to improve the efficiency of AHC [10], while it only was implemented for a single node. As for Graph Clustering, massive efforts were also put into studying efficient algorithms [1, 3, 13, 22]. Nevertheless the hierarchical characters attracted little attention in these work. Grygorash et al. presented the Hierarchical euclidean-distance-based MST clustering algorithm...
Given the number of clusters as an input, HEMST accelerates the convergence progress by merging multiple edges, while the edges larger than the threshold are required to be sorted and removed in order [25]. Jin et al. split the clustering problem into various overlapped subproblems by a Prim algorithm, solved each subproblem, and then merged them into an overall solution [30].}

However, the MSTs are needed to store at the Map side and then shuffle to the Reducers. Bateni et al. [8] extended MST-based method called affinity clustering with two classic MST algorithms. It still relied heavily on MapReduce and requires moving all the edges to one machine serially for MSTs after the edges deletion. Moreover, affinity clustering did not reveal the exact running times and number of machines used in their experiments [8]. Wang et al. utilized a divide-and-conquer scheme to construct approximate MSTs, while the process to detect the long edges of the MST is also highly sequential at an early stage for clustering [46, 54]. As a result, an efficient clustering algorithm competent for parallel computing on large-scale data is in need crucially to improve the accuracy of K-means, efficiency of AHC, and scalability of GC methods.

The second focus of this paper is closely related to distributed regression, which suffers from serious scalability problems in both computation time and memory usage [11]. Mini-batch Gradient Descent (MBGD) was used for training on batched data [28], while it was proposed for serial implementation. MapReduce-based methods [14, 27, 42, 48] made computation distributed locally which holds parts of the data. However, the overall cost in terms of computation and network is high because of the busy synchronization sweeps. The Divide-and-Conquer algorithm was then adopted on

Table 5: Analytical Proportion and speedup for different procedures by our library on Cadata and Gas Sensor datasets.

| Dataset        | Cadata | Gas Sensor |
|----------------|--------|------------|
| Component      | C.I/O(%) | C.(%) | R.I/O(%) | R.(%) | M.(%) | S.(C.+R.) | S. |
| 96             | 2.80    | 63.98 | 0.34    | 28.28 | 0.84    | 1.00     | 1.00 |
| 192            | 3.58    | 54.96 | 0.54    | 25.75 | 1.49    | 1.34     | 1.18 |
| 384            | 4.54    | 46.88 | 1.54    | 25.15 | 4.00    | 1.74     | 1.36 |
| 768            | 5.94    | 40.41 | 2.21    | 22.73 | 10.73   | 2.19     | 1.50 |
| 1536           | 7.94    | 31.62 | 3.39    | 21.60 | 25.45   | 3.18     | 1.83 |
| 3072           | 8.48    | 25.35 | 4.33    | 18.22 | 43.86   | 3.99     | 1.88 |
| 6144           | 10.09   | 27.21 | 5.18    | 14.04 | 43.01   | 4.66     | 2.09 |
| 12288          | 8.29    | 20.73 | 4.48    | 12.70 | 39.45   | 4.38     | 1.59 |
| Mean           | 6.46    | 38.89 | 2.75    | 21.06 | 19.98   | 2.81     | 1.55 |

1 For better clarity, procedures for clustering, regression, and communication are abbreviated with C., R., and M. respectively.

The speedup is also abbreviated to S..
An Accurate and Efficient Large-scale Regression Method through Best Friend Clustering. Conference’21, February 2021, St. Louis, MO

The text discusses the optimization of kernel ridge regression (KRR) and presents an accurate and efficient method through best friend clustering. The paper reviews prior work on KRR and SVM, focusing on methods that address scalability and memory constraints. It introduces a balanced partition algorithm inspired by backtracking and evaluates the performance of different clustering methods on parallel systems.

**REFERENCES**

1. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2012. Analyzing graph structure via linear measurements. In Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms. SIAM, 459–467.

2. Prajesh P Anchalia, Anjan K Koundinya, and NK Srinath. 2013. MapReduce design of K-means clustering algorithm. In 2013 International Conference on Information Science and Applications (ICISA). IEEE, 1–5.

3. Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In Proceedings of the forty-sixth annual ACM symposium on Theory of computing. 574–583.

4. Tetru Asano, Binaey Shasthrawy, Mark Keil, and Frances Yao. 1988. Clustering algorithms based on minimum and maximum spanning trees. In Proceedings of the fourth annual symposium on Computational geometry. 252–257.

5. Balman Balman, Benjamin Moseley, Andrea Vattani, Raviv Kumar, and Sergei Vassilvitskii. 2012. Scalable K-Means++. Proc. VLDB Endow. 5, 7 (March 2012), 622–633. https://doi.org/10.14778/2180912.2180915.

6. Maria-Florina F Balcan, Steven Ehrlich, and Yingyu Liang. 2013. Distributed k-means and k-median clustering on General Topologies. Advances in Neural Information Processing Systems 26 (2013), 1995–2003.

7. Mohammadhossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab S Mirrokni. 2014. Distributed Balanced Clustering via Mapping Coresets. In NIPS 2591–2599.

8. Mohammad Hossein Bateni, Soheil Behnezhad, Mahsa Derakhshian, Mohammad Taghi Hajiaghayi, Ramin Mardaneh, and Vahab S Mirrokni. 2017. Affinity clustering: Hierarchical clustering at scale. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 6867–6877.

9. Ron Bekkerman, Mikhail Bilenko, and John Langford. 2011. Parallel algorithms for geometric graph problems. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE, 73–81.

10. Qing He, Tianfeng Shang, Fuhe Zhuang, and Zhongzhi Shi. 2013. Parallel extreme learning machine for regression based on MapReduce. Neurocomputing 102 (2013), 52–58.

11. Geoffery Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on 14, 8 (2012).

12. Chris Ding and Xiaofeng He. 2004. K-means clustering via principal component analysis. In Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms. SIAM, 603–618.

13. Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Ng. 2007. Map-reduce for machine learning on multicore. In Advances in neural information processing systems. 281–288.

14. William HE Day and Herbert Edelsbrunner. 1984. Efficient algorithms for agglomerative hierarchical clustering methods. Journal of classification 1, 1 (1984), 7–24.

15. Tetsuo Asano, Binay Bhattacharya, Mark Keil, and Frances Yao. 1988. Clustering methods. You presented K-Means kernel ridge regression. Thus, in this paper, the focus is paid to the comparison of two closely related papers DCKRR and BKRR2, which are also elaborated in the Introduction section.

7. **CONCLUSION**

In this paper, we first propose a Best Friend Clustering method, which is more accurate, fast, and meanwhile parameter-free. Then we devise a strategy to determine the optimal aggregation through a well-defined metric and achieve it efficiently on distributed systems. Moreover, a balanced partition algorithm inspired by backtracking is devised for load-balance in parallel implementation. At last, we integrate the proposed methods into multiple regression techniques for a parallel library, and it shows superior performance on convergence, accuracy, and scalability.
World Congress in Computer Science, Computer …., 1.

[41] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2016. *Introduction to data mining*. Pearson Education India.

[42] Choon Hui Teo, SVN Vishwanathan, Alex Smola, and Quoc V Le. 2010. Bundle Methods for Regularized Risk Minimization. *Journal of Machine Learning Research* 11, 1 (2010).

[43] Cor J. Veenman, Marcel J. T. Reinders, and Eric Backer. 2002. A maximum variance cluster algorithm. *IEEE Transactions on pattern analysis and machine intelligence* 24, 9 (2002), 1273–1280.

[44] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2009. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th annual international conference on machine learning*. 1073–1080.

[45] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. 2001. Constrained k-means clustering with background knowledge. In *ICML*, Vol. 1. 577–584.

[46] Xiaochun Wang, Xiaoli Wang, and D Mitchell Wilkes. 2009. A divide-and-conquer approach for minimum spanning tree-based clustering. *IEEE Transactions on Knowledge and Data Engineering* 21, 7 (2009), 945–958.

[47] Wikipedia. 2020. Global city. Website. https://en.wikipedia.org/wiki/Global_Cities_Index.

[48] Hailong Yang, Zhongzhi Luan, Wenjun Li, and Depei Qian. 2012. MapReduce workload modeling with statistical approach. *Journal of grid computing* 10, 2 (2012), 279–310.

[49] Yang You. 2020. *Fast and Accurate Machine Learning on Distributed Systems and Supercomputers*. (2020).

[50] Yang You, James Demmel, Cho-Jui Hsieh, and Richard Vuduc. 2018. Accurate, fast and scalable kernel ridge regression on parallel and distributed systems. In *Proceedings of the 2018 International Conference on Supercomputing*. 307–317.

[51] Charles T. Zahn. 1971. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on computers* 100, 1 (1971), 68–86.

[52] Yuchen Zhang, John Duchi, and Martin Wainwright. 2013. Divide and conquer kernel ridge regression. In *Conference on learning theory*. 592–617.

[53] Yuchen Zhang, John Duchi, and Martin Wainwright. 2015. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research* 16, 1 (2015), 3299–3340.

[54] Caiming Zhong, Mikko Malinen, Duoqian Miao, and Pasi Fränti. 2015. A fast minimum spanning tree algorithm based on K-means. *Information Sciences* 295 (2015), 1–17.