Tree Based Clustering On Large, High Dimensional Datasets[⋆]

Lee A. Carraher, Sayantan Dey, and Philip A. Wilsey[0000−0002−6562−8646]

Dept of EECS, University of Cincinnati, Cincinnati, OH 45221–0030 USA
leecarraher@gmail.com, deysn@mail.uc.edu, wilseypa@gmail.com

Abstract. Clustering continues to be an important tool for data engineering and analysis. While advances in deep learning tend to be at the forefront of machine learning, it is only useful for the supervised classification of data sets. Clustering is an essential tool for problems where labeling data sets is either too labor intensive or where there is no agreed upon ground truth. The well studied k-means problem partitions groups of similar vectors into k clusters by iteratively updating the cluster assignment such that it minimizes the within cluster sum of squares metric. Unfortunately k-means can become prohibitive for very large high dimensional data sets as iterative methods often rely on random access to, or multiple passes over, the data set — a requirement that is not often possible for large and potentially unbounded data sets. In this work we explore an randomized, approximate method for clustering called *Tree-Walk Random Projection Clustering* (TWRP) that is a fast, memory efficient method for finding cluster embedding in high dimensional spaces. TWRP combines random projection with a tree based partitioner to achieve a clustering method that forgoes storing the exhaustive representation of all vectors in the data space and instead performs a bounded search over the implied cluster bifurcation tree represented as approximate vector and count values. The TWRP algorithm is described and experimentally evaluated for scalability and accuracy in the presence of noise against several other well-known algorithms.

Keywords: Clustering · machine learning · dimensional reduction · locality sensitive hashing.

1 Introduction

The expanding needs for analysis on large data sets has increased as the amount and availability of data continues to grow. Concepts such as the Internet of Things (IoT), social media, digitized medical records, and the aggregation of complex high volume scientific measurements further exacerbated this need. The size and format of this data makes manual analysis infeasible and has motivated the drive for automated methods such as data clustering.

⋆ Support for this work was provided in part by the National Science Foundation under grant ACI–1440420.

Meanwhile big data solutions often find their greatest success on large fast moving data sets. By virtue of the sheer volume of these datasets, supervised labeling is often not possible. Unsupervised learning fills this gap by sacrificing a desired optimization based on a set of ground truths for a solution that, in the case of clustering, tries to optimize the co-similarity of objects in a partition or dissimilarity of object between partitions. Clustering is the process of partitioning data into grouping of related items. It is one of the most fundamental modes of learning and understanding data [21]. Clustering is an intuitive, data analysis method that can provide clear insights into the underlying structure and trends of a dataset. Given its interpretability at high applicability, clustering has been used in a wide variety of disciplines ranging from medical diagnostics and epidemiology to financial prediction and credit fraud detection. As the number of sources and volume of these types of datasets grow deeper insights are desired.

Among the most commonly used clustering algorithms, k-means has been proven as one of the most popular choices that delivers acceptable results in reasonable time [21]. k-means has proven to be statistically efficient and easy to implement. While k-means is widely used for clustering streaming data, it has performance issues when it comes to robustness with noise, parallelism, and working with very large, high-dimensional data sets. In particular, k-Means (and other conventional techniques) for data clustering do not parallelize or scale well with the increasing dimensionality of data.

This paper introduces and evaluates an approximate method for clustering called *Tree-Walk Random Projection Clustering (TWRP)* that is a fast, memory efficient method for finding clusters in high dimensional spaces. TWRP is a tree based clustering method that forgoes storing the exhaustive representation of all vectors of the data space and instead performs a bounded search over the implied cluster bifurcation tree represented as approximate vector and count values. Big data clustering algorithms are not new; in fact many classic algorithms are reasonably well suited to operate on big data with only minor pre-clustering tweaks [26]. Although successful, most methods optimize over the entire data embedding, while TWRP finds not only dense regions in the full embedding, but also allows for the identification of dense low-rank embeddings.

The remainder of this paper is organized as follows: Section 2 provides some background information. Section 3 surveys related work to develop solutions for high-performance big-data clustering. Section 4 describes the Tree Walk Random Projection (TWRP) cluster method and the original RPHash algorithm [8] (on which TWRP is based). Section 5 contains experimental results of our proposed solution against other common clustering methods. Finally, Section 6 gives a summary of our results and findings.

2 Preliminaries

An important tool for overcoming the Curse of Dimensionality *COD* is *dimensional reduction*. In the case of very large datasets, more robust data aware dimensional reduction techniques such as t-SNE and PCA begin to dominate the computational complexity. For ill-posed problems such as clustering, the optimal subspace embedding that is based on minimizing either the L_2-norm or Kullback-Leibler divergence is somewhat overkill. Instead, approximate dimensional reduction technique such as Random Projection using the method of Database Friendly Projection by Achlioptas [1] is sufficient.

The JL-Lemma [32] states that the error bound for low dimensional embeddings, is exponentially proportional to the number of objects in the dataset. For large datasets, such as those we are interested in, this bound is still somewhat prohibitive, calling for subspace embeddings on the order of thousands. Thus for $1 - \epsilon = .90$, and $n > 10^9$, the reduced dimension D is bounded below by $D > \Omega(\frac{\log(n)}{\epsilon^2})$.

Locality Sensitive Hash (LSH) functions are employed as probabilistic representation of vector locality to improve the prohibitive subspace embedding dimensionality requirement of the JL-Lemma. An LSH function is any hash function with the property that hashed records with more similar components are more likely to be hashed to the same bucket than records with more fewer similarities. Formally:

Definition 1 (Locality Sensitive Hash Function [11]). *let $\mathbb{H} = \{h : S \to U\}$ is $(r_1, r_2, p_1, p_2)-$ sensitive if for any $u, v \in S$, where h is the hash function belonging to the hash family $\mathbb{H}$ that maps from the element set S to U and d is the distance metric. Thus, an LSH function operates so that:*

if $d(u, v) \leq r_1$ *then* $Pr_{\mathbb{H}}[h(u) = h(v)] \geq p_1$, *and*
if $d(u, v) > r_2$ *then* $Pr_{\mathbb{H}}[h(u) = h(v)] \leq p_2$.

While LSH functions are interesting approaches to quickly compute vector locality, they tend to have some difficulties separating distinct communities of vector data, especially when the vector data is not uniformly distributed throughout the subspace. While considerable work has gone into finding better and near optimal [4] functions for optimizing the signal to noise ratio for locality sensitive hash functions, the best solutions often require multiple passes over the data to build data aware functions.

To switch from the continuous spaces of random projections, discrete space partitioning lattices are also considered. The data space must be partitioned as evenly as possible for an optimal implementation of the clustering. Also to avoid expensive interprocess communication overhead, a universally generative naming scheme must be established. For many known datasets, the Voronoi partitioning [22] can be generated in $\Theta(nlog(n))$-time [16] for 2D space and produces perfect partitions. However as the dimension increases this algorithm have less favorable run time complexities [17], making them inefficient for partitioning higher dimensional data sets.

3 Related Work

In this section we present related work on clustering large scale datasets. A variety of clustering methods have been proposed in the past that take advantage of estimation techniques for machine learning, namely: *tree based clustering*, *dimensional reduction*, and *locality sensitive hashing (LSH).*

The DBScan [13], Clique [3], and CLARANS [28] algorithms represent a successful progression of *density scanning techniques*. Although density scan algorithms are often scalable, they often show weaknesses in accuracy when scaling the number of dimensions.

Proclus [2] explored random projection for clustering. The merits of random projection are discussed in [10]. They suggest that random projection not only compresses sparse data sets, making them computationally more tractable, but also may help overall accuracy by alleviated round-off issues caused by non-homoscedastic variance. This occurs because random projection generates more spherical clusters in a more dense subspace. In addition to Proclus, various other methods and analysis have been proposed for clustering with random projections that provide bounds on the convergence and limits of random projection clustering. Florescu provides bounds on the scaling and convergence of projected clustering [15]. Their results closely follow the logic of Urruty [31] who find that the number of orthogonal projections required is logarithmic in n (where n is the number of vectors to be clustered). Following that, the probability of a random projection plane offering a good partitioning increases exponentially as the number of dimensions in the projected subspace increases. Bingham *et al* provide examples of projected clustering well below the JL bound [7] and Bartal *et al* make these assertions mathematically rigorous showing that the projected subspace is independent of the data's original dimensionality [6].

4 Tree-Walk RPHash

RPHash [8] is an algorithm for dense region and microcluster identification suitable as a precursor to more robust clustering algorithms like k-Means and agglomerative hierarchical clustering or as a standalone approximate clustering algorithm. In *RPHash*, both approximate and randomized techniques are employed to provide a stochastic element to the clustering algorithm. Due to this nature it has the tendency to produces some variation in its outputs. The Tree Walk RPHash (TWRP) extension attempts to mitigate this instability and provide stability to the results.

The basis of the Tree Walk RPHash (TWRP) algorithm is the RPHash clustering algorithm which has capacity to be amenable to distributed and steaming settings [9]. The RPHash algorithm arose from a realization that the degenerative cases for LSH k-nearest neighbor search is a useful method for identifying candidate cluster centers. A problem arises in the query step in which a particular LSH hash is disproportionately over represented. The result is that the algorithm must linearly scan and order all members of that hash bucket, to decide which are the nearest neighbors of the query vector. While bad for LSH K-NN, RPHash uses these outlier buckets as candidates for cluster centers. The original RPHash algorithm is shown in Algorithm 1. Here x_k is a vector form the dataset X. This vector is projected multiple times by a projection matrix p

taken form the larger set P. Then the vector is mapped to a region or bucket following the LSH scheme and the bucket counts are recorded. In the second pass over the data (the second **forall** statement) only those vectors assigned to the buckets with high counts are considered as the probable candidate cluster centroids (c_i).

In the original RPHash description, the LSH function is often chosen to be some generative function that uniformly covers a desired subspace. Ideally, an LSH function distributes vectors into buckets such that the underlying dense and sparse structures of the dataset are identifiable. One way to achieve this is to scale size of the LSH regions (often by the overall data's variance, and cardinality of the LSH mapping) to balance it between the two extremes: too dense or too sparse. To address this issue, a data aware LSH function was developed, called adaptive LSH (Algorithm 2), that attempts to optimize the distribution of hash buckets over the data-space. Adaptive LSH proceeds by taking a simple LSH function similar to the p-stable distribution LSH [20]. A key attribute to this particular type of LSH function is the immediate relationship between adjacent depths of the hash that we used in this algorithm. Composeable LSH allow us to balance the hash ID allocation.

Algorithm 1: 2-Pass RPHash

```
forall $x_k \in X$ do
    forall $p_i \in \mathbb{P}$ do
        $\tilde{x_k} \leftarrow \sqrt{\frac{m}{d}} p_i^\intercal x_k$
        $t = \mathbb{H}(\tilde{x_k})$
        $L[k][i] = t$
        $C$.add($t$)
forall $x_k \in X$ do
    forall $c_i \in C.top(K)$ do
        if $L[k] \cap M[i][0] \neq 0$ then
            $\Delta = x_k - M[k]$
            $M[k] =$
              $M[k] + \Delta/count$
            $L[k]$.add($M[i][0]$)
Result: Candidate Centroids
```

Algorithm 2: Adaptive LSH

```
$i = 1$
$ct, ct\_prev =$
  $C(\mathbb{H}^{i+1}(x)), C(\mathbb{H}^{i}(x))$
while $i < n$ and
 $2ct > ct\_prev$ do
    $ct\_prev, i = ct, i + 1$
    $ct = C(\mathbb{H}^{i}(x))$
Result: $\mathbb{H}^{i}(x)$
```

Definition 2 (LSH Composability). *An LSH function $\mathbb{H}^n(x)$ that maps $x \in \mathbb{R}^n \to \mathbb{Z}_2^n$, is composable if there is a related function $\mathbb{H}^{n-1}(x_{n-1})$ that maps $x_{n-1} \in \mathbb{R}^{n-1} \to \mathbb{Z}_2^{n-1}$ where $\mathbb{H}^{n-1}(x_{n-1}) = (\mathbb{H}^n(x) + 1) \bigcup (\mathbb{H}^n(x) + 0)$ for all $x_n \in \mathbb{R}^n$*

Although a variety of metrics are feasible, one of the simplest is to continue to extend the hash depth, so long as the subsequent hash count is greater than half of the parent hash count. This method comprises the adaptive LSH function (Algorithm 2).

We use the simple LSH function proposed in Indyk and Motwani's original work on LSH for approximate near neighbor search [20]. We apply the hamming space LSH function to projected euclidean space by observing the signs of the projected vectors with respect to the origin. Formally, a *Signed Based Projected LSH function* is defined as: $H(X) = \sum sign(P(X))2^n$.

The TWRP method begins with the standard RPHash algorithm, but instead of only updating buckets, TWRP also increments the counts of all sub-hashes as well. This bares a slight resemblance to Liu *et al* [23] while adapting their algorithm to work in a streaming and distributed setting and without using any supervised learning. TWRP can use a variety of metrics for the tree splitting condition. Unlike Liu *et al* TWRP does not concern itself with the more complicated calculation to compute C4.5 entropy method. TWRP avoids the splitting of clusters with random hyperplanes by the virtue of its application to high dimensional data that is, the probability of splitting a cluster goes to zero as the dimensionality grows. The theorem below is a consequence of the curse of dimensionality.

Theorem 1. ***(Hyperrectangle Splitting)*** *The probability of splitting a hyperrectangular region into two equal mass clusters where subsequent dimensional cuts contain the smaller of the two induced regions region approaches 0 exponentially in d.*

$$\lim_{d\to\infty} \frac{Vol(R) - Vol_{removed}(R)}{Vol(R)} = 0,\ R\ Rectangle\ \in \mathbb{R}^d. \tag{1}$$

Proof.

Let X s.t. $x_j = [0...c...0] \in \mathbb{R}^d$ is orthogonal, $c \in [0, 1)$, $\sum_i^n x_i = \mathbb{P}$, is a plane in $\mathbb{R}^d$, and $Vol(R) = 1$

Let: $S_1(p)$ be the volume of the projection of R on x_p Restrict $S_1(p)+\tilde{S_1}(p) = 1$, $S_1(p) \leq \tilde{S_1}(p)$ for all p

$V(R_{s(X)}) = \prod_p^n S_1(p)$ where $S_1(p) \in U[0, \frac{1}{2})$

$V(R_{s(X)}) \leq 2^{-n}$ for all n, $\lim_{n\to\infty} 2^{-n} = 0$

$\Rightarrow V(R_{S(X)}) + V(\tilde{R}_{S(X)}) = V(s)$, $V(\tilde{R}_{S(X)}) = V(s)$ as $d \to \infty$ ■

Algorithm 3 and Algorithm 4 are the core elements of TWRP. The algorithm is linear in the input vector size x. For each vector TWRP must compute the projection and update the counter (Algorithm 3). This algorithm introduces two operations the $+$ to mean population weighted addition, and $\gg$ for the bit shift operation. Projection using the approach of Achlioptas [1] can be performed in $dm/3$ operations where d is the original dimensionality and m is the projection sub-dimension. As each vector comes in it is projected onto the new lower dimension which the user inputs at runtime. There are m levels of hashing and as each level there are 2^{level} of buckets. Each projected vector is hashed onto a vector using the LSH scheme for every level. Thus all the vectors are present at all levels. We bound the memory as we only store a single vector in each bucket. This is done by storing only the mean of the vectors coming into a specific bucket. We also store the count for each bucket. Whenever a vector is to be inserted into a bucket, we update its vector using weighted merge and increase the count by one. It is to be remembered that at this point we store the vector means in their original dimension. Thus we diminish the dimension only to determine in which bucket each vector goes in and then store the vector means in its original dimension into the buckets.

Algorithm 3: Online Tree Generation

```
forall $x \in X$ do
    $\tilde{x} = \sqrt{\frac{m}{d}} p^{\intercal} x$ $h := \mathbb{H}(\tilde{x})$
    while $h > 0$ do
        $h = h \gg 1$
        $x' = C[h] + x$
        $C$.add$(h, x')$
```

Algorithm 4: Offline Tree Search

```
forall $H \in sort(C.ids)$ do
    if $2C[H] < C[H \gg 1]$ then
        $C[H \gg 1] = 0$
$L = []$
forall $h \in sort(C.counts)$ do
    $L \leftarrow$ medoid$(C[H])$
return $L$
```

The offline step consists of exploring and updating the count records (Algorithm 4). In general it follows a depth first search traversal for candidate clusters with a worse case complexity of exploring all non-leaf nodes, $\theta(2^{m-1})$. This is done by comparing the counts of each bucket at all levels. The hash values form a binary tree with each level having 2^{level} nodes. We compare the count of parent and child nodes/buckets and look for the possible centroids. We also remove the buckets with a threshold count value assuming these are possible noise. Thus only the dense part of the tree are kept for possible centroid values. We then return a overestimate of our weighted candidate centroids and use standard clustering algorithms such as k-means or Hierarchical Agglomerative to reduce these overestimated centroids to the desired number.

5 Experimental Results

TWRP has multiple configuration parameters, namely: projection distance, offline clustering algorithm, and sizes of the overestimated candidate cluster set. During preliminary testing of the TWRP, approximately 800 different configurations were evaluated using synthetically generated labeled data sets. Based on this testing, the best configuration that provided the consistently "best" WCSS result was selected. In particular, a configuration that projects data to 16 dimensions, using the offline k-means clusterer, and an overestimated centroid count of $10 \times k$ (where k is the number of clusters desired) was selected. This configuration is used to collect all the data reported in this section. Because of the stochastic nature of TWRP, the TWRP algorithm is configured to be run six times and the result with the best WCSS value is reported. The hardware used for testing is a 16 core Intel(R) Xeon(R) E5-2670 @ 2.6GHz, supporting 32 threads with 64GB of RAM.

5.1 Data Sets

The initial testing was performed with synthetic data generated as 10,000 vectors from dimensions 100 to 7,000. Each generated data set has 10 Gaussian clusters with labels recorded for all points. An accompanying noise test with synthetic data was performed. In particular, a 10,000 vector data set at dimension 1,000 was generated. We then replaced a percentage of vectors with randomly generated vectors but keeping the original label attached to the noise data. The noise was injected in percentages of 5-40 percent in steps of 5 percent. Finally, a set of scalability of the TWRP algorithm was performed also using synthetic data. In particular, we generated 8 synthetic datasets with 10 clusters and of 1,000 dimensions but varying the total number of vectors from 30,000 to 1,000,000.

Finally We used six real world datasets all available at the UCI machine learning repository. The *Human Activity Recognition Using Smartphones (HAR)* dataset is taken from the [5]. This dataset consists of 10,299 vectors containing 561 features consisting of 6 clusters. The *Smartphone Dataset for Human Activity Recognition (HAR) in Ambient Assisted Living (HARAAL)* dataset is taken from [12]. This data has 5744 vectors and 561 features. It has 6 clusters. The *gene expression cancer RNA-Seq Data Set (RNASEQ)* dataset is taken from [14]. This data has 801 vectors and 20531 features and 4 clusters. The *Smartphone-Based Recognition of Human Activities and Postural Transitions Data Set (HAPT)* dataset is taken from [29]. This data has 10929 vectors and 561 features. This dataset has 12 cluster groups. The *Indoor Location Data (INLOC)* dataset is taken from [30]. This data has 21048 vectors and 529 features. It consists of 3 clusters. The *Gas Sensor Array Drift (GSAD)* dataset is taken from [33]. This data has 13910 vectors and 128 features. It is composed of 6 clusters.

5.2 Algorithms Used for Comparison

We compared the performance of TWRP against six standard well known algorithms, specifically:

- *k-Means*: This is the algorithm of Hartigan and Wong [18] implemented with the function k-`means` in R [24].
- Four methods of *Agglomerative Hierarchical clustering*, namely: Single Linkage, Complete Linkage, Average Linkage and Ward's minimum variance method. At every stage the inter-cluster distances are recomputed by the Lance-Williams dissimilarity update formula according to the particular clustering method that is being used. For Ward's [27]) clustering method,the dissimilarities are squared before updating the cluster. We have used the function `hclust` in R for implementing these algorithms.
- *Self-organizing Tree Algorithm* (SOTA): This is relatively new algorithm based on neural network. It is implemented using the `sota` function in R (found in package `clValid`) [24].

These algorithms are selected due of their importance, popularity and availability in R statistical computing framework. These algorithms use FORTRAN, C and

C++ subroutines from R to make them run faster. The implementation language of TWRP is Java.

5.3 Comparison with Other Algorithms

The clustering performance and runtime for TWRP is compared to the six other algorithms described in Section 5.2. The clustering accuracy of TWRP is evaluated using 2 external clustering validation measures: Adjusted Rand Index (ARI) and Cluster Purity. We also use the internal measure WCSS for evaluation. ARI [19] measures the extent to which points from the same ground-truth partition appear in the same cluster, and the extent to which points from different ground-truth partitions are grouped in different clusters. ARI eliminates the chance of misjudging clustering outputs in cases where the output labels could be switched even if the clusters are well identified. The value of ARI lies between -1 and 1; an ARI of 1 denotes a perfect agreement between two partitions (and therefore a perfect clustering). Cluster purity [25] measures how many data points were correctly assigned to its original cluster. WCSS (also called WCSSE or SSE) is the within cluster sum of squared error. A lower value of WCSS indicates better clustering performance. It is basically the objective function that k-means algorithm tries to minimize in order to find suitable clusters.

The measured ARI, PURITY and WCSS for the synthetic datasets are plotted in Figure 1. The TWRP algorithm produces results on all measures are always better than that of k-means algorithm. TWRP has the value of 1 for ARI and PURITY for all these datasets,indicating perfect clustering. TWRP's WCSS also matches the baseline WCSS (*i.e.,* the actual WCSS for a dataset) for these synthetic data.

For the noise injected data, ARI, and WCSS are plotted in Figure2. The WCSS of TWRP are much lower than single, average and complete linkage and SOTA algorithms. The performance of k-means and TWRP are comparable. We see as the noise grows to 40 percent (when the signal itself is poor) all the algorithms tend to perform poorly. This is because we randomly generated noise vectors and kept the original label.

The results for the real world datasets are summarized in Table 3. The results show that TWRP, k-means, and Wards algorithm have a very similar performance. In contrast, single link perform poorly.

Scalability results are shown in Table 1 using synthetically generated data sets at a range of dimensions between 100 to 7000. TWRP outperforms the other algorithms by a large margin. TWRP is almost 56 faster than k-means and 358 times faster than Agglomerative Hierarchical for synthetic datasets having 7000 dimensions and 10,000 vectors. TWRP shows very little runtime performance degradation as the dimension increases. The same cannot be said of the other algorithms. The runtime at a fixed dimension as the number of vectors was increased of was also studied. In particular Table 2 shows scalabilty of TWRP on a data set of 1,000 features as the number of vectors is varied between 30K to 1M. This shows a growth in the runtimes that is linear to the number of vectors.

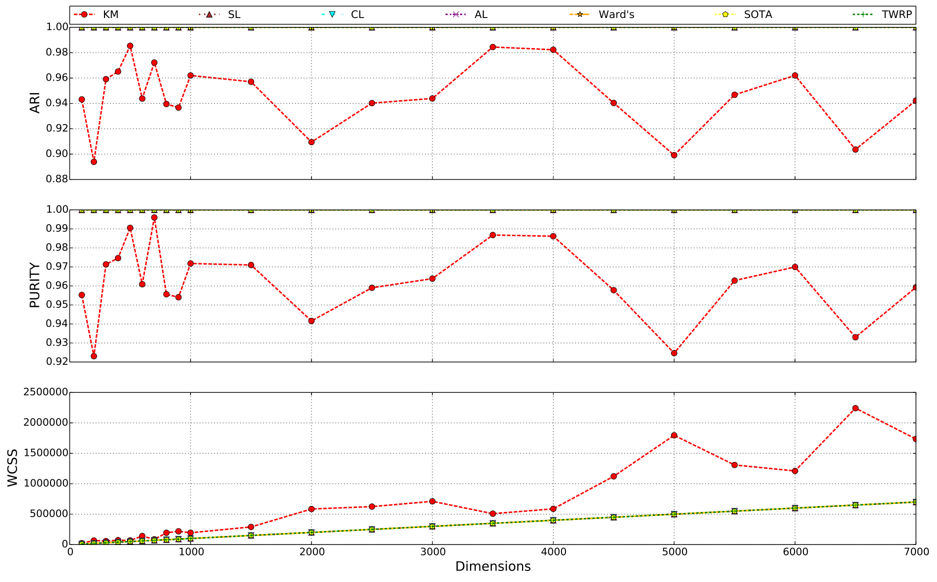

Fig. 1. ARI, Purity and WCSS Plotted Against Increasing dimension

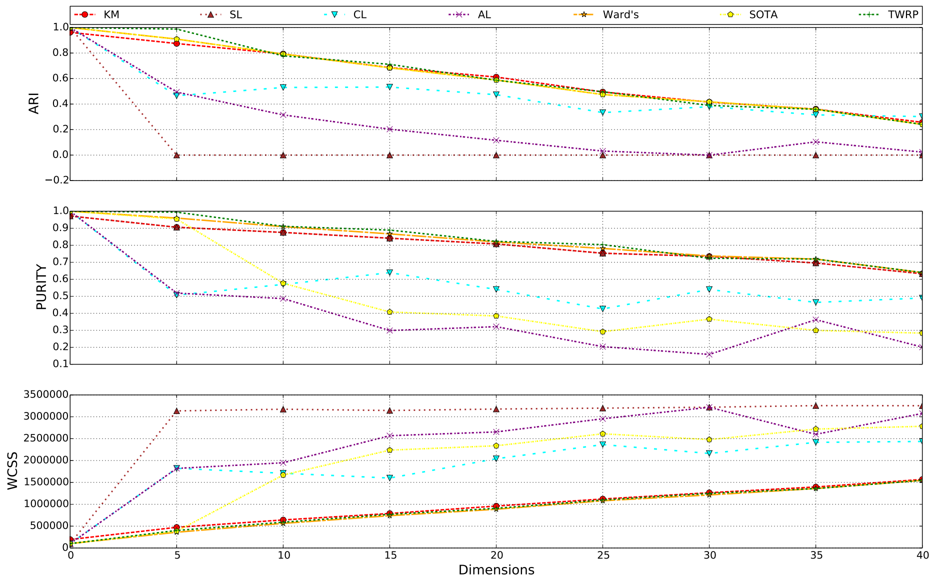

Fig. 2. ARI, Purity and WCSS Plotted Against Noise

Dataset	Measures	ARI	Purity	WCSS	Time
HAR	k-means	0.4610	0.6002	182 169	66.33
	SL	0.0000	0.1890	556 519	493.95
	CL	0.3270	0.3770	222 044	494.47
	AL	0.3321	0.3588	236 143	494.21
	Ward's	0.4909	0.6597	191 441	494.64
	SOTA	0.3143	0.3966	210 490	23.63
	TWRP	0.5125	0.5849	188 552	2.82
HAPT	k-means	0.3988	0.6498	2 498 381	182.90
	SL	0.0003	0.1821	6 023 519	601.98
	CL	0.0488	0.2505	4 584 352	602.42
	AL	0.0055	0.2046	5 491 388	602.04
	Ward's	0.4033	0.6624	2 617 769	602.68
	SOTA	0.3026	0.3848	2 990 195	31.13
	TWRP	0.3541	0.6257	2 593 802	6.54
HARAAL	k-means	0.2461	0.4240	1 618 089	23.55
	SL	0.0000	0.1964	3 166 056	148.43
	CL	0.0003	0.2002	3 043 579	148.53
	AL	0.0001	0.1972	3 097 976	148.45
	Ward's	0.2764	0.3929	1 653 179	148.58
	SOTA	0.2370	0.3785	1 814 593	12.30
	TWRP	0.2352	0.4076	1 636 495	2.64
INLOC	k-means	0.6048	0.8122	9 001 974 483	47.45
	SL	0.0000	0.4637	10 823 661 183	2266.78
	CL	0.7298	0.8413	9 257 876 517	2268.77
	AL	0.0000	0.4637	10 816 420 807	2267.82
	Ward's	0.7452	0.8469	9 040 839 873	2269.25
	SOTA	0.3954	0.7149	9 296 280 113	35.10
	TWRP	0.5322	0.7873	9 055 103 257	3.72
Gas-Sensor	k-means	0.1539	0.4427	27 714 236 160 297	9.05
	SL	0.0000	0.2165	192 076 751 899 323	42.61
	CL	0.0380	0.3474	47 050 045 285 192	42.57
	AL	0.0037	0.2865	75 622 509 139 114	42.45
	Ward's	0.2007	0.4378	31 162 058 051 998	42.87
	SOTA	0.0281	0.3435	46 727 103 818 336	11.93
	TWRP	0.2040	0.4709	31 000 000 000 000	1.86
RNASEQ	k-means	0.6438	0.8402	12 834 131	180.32
	SL	0.0007	0.3783	16 007 266	97.10
	CL	−0.0124	0.3758	15 692 260	97.10
	AL	0.0007	0.3783	16 007 266	97.08
	Ward's	0.5955	0.8202	12 916 461	97.09
	SOTA	0.3205	0.6317	13 632 923	87.72
	TWRP	0.5944	0.8077	20 357 469	7.45

Fig. 3. Performance for real data sets

Dimension	*k*-**means**	**Average Link**	**SOTA**	**TWRP**
100	3.9656	47.46	13.758	1.5930
300	13.3796	194.121	23.797	2.2386
500	32.92	454.777	36.046	2.7942
700	77.7298	672.759	48.963	3.7974
900	117.3675	929.6	61.759	4.2552
1000	142.2341	1064.178	68.86	4.6368
1500	237.0007	1789.142	96.795	6.3204
2000	366.0743	2450.813	127.972	7.9176
2500	431.5876	3108.654	155.968	9.6270
3000	542.0223	3771.886	185.23	11.1174
3500	631.8423	4435.349	220.562	12.9984
4000	741.915	5080.802	248.669	14.3292
4500	811.3911	5725.061	274.983	15.3030
5000	909.223	6362.574	301.314	17.2584
5500	975.2703	7006.91	324.314	19.3518
6000	1076.882	7645.155	359.911	20.2080
6500	1187.6062	8278.964	400.767	21.4596
7000	1340.8115	8520.85	428.15	23.7648

Table 1. Scalability with respect to Dimension(seconds)

size	**30K**	**50K**	**70K**	**90K**	**100K**	**300K**	**600K**	**1M**
TWRP	10.62	16.53	21.17	27.20	31.52	84.53	157.58	278.29

Table 2. Scalability with respect to size of dataset (seconds)

6 Conclusions

In this work we introduced the Tree-Walk Random Projection (TWRP) algorithm for clustering large high dimensional datasets with log-linear processing complexity. We applied the TWRP algorithm to real data and found that it performs comparably to other commonly used clustering methods. In tests with synthetic data, where clusters are well defined and spherical, we find that TWRP accuracy outperforms k-means and is equivalent to other standard techniques. In addition, TWRP performance is at par with k-means and Ward's algorithm for noise injected into a dataset. We suspect this is a result of our simplified clustering metric favoring higher support partitions which is unable to distinguish a partition of noise from the true cluster ground truth. We believe choosing a different clustering metric such as C4.5 or within cluster sum of squared error, for the cluster tree bifurcation could potentially extend the effectiveness of TWRP to noisier data sets, however with an additional cost in computation and storage.

The complexity analysis of our method finds that it scales very well both in time and space complexity. The overall scalability is predictable and does not hide large constants.

As TWRP was designed to process very large high dimensional clustering problems, a key requirement is that it be fast. The results show quite clearly that TWRP achieves this goal by comprehensively beating other well known algorithm in its running time and scalability.

References

1. Achlioptas, D.: Database-friendly random projections. In: Proc of the 20th Symp on Principles of Database Systems. pp. 274–281 (2001)
2. Aggarwal, C.C., Wolf, J.L., Yu, P.S., Procopiuc, C., Park, J.S.: Fast algorithms for projected clustering. In: Proc of the 1999 ACM SIGMOD Int Conf on Management of Data (SIGMOD '99). pp. 61–72 (1999)
3. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc of the 1998 ACM SIGMOD Int Conf on Management of Data. pp. 94–105 (1998)
4. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on. pp. 459–468. IEEE (2006)
5. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: UCI machine learning repository (2012), `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`
6. Bartal, Y., Recht, B., Schulman, L.J.: Dimensionality reduction: beyond the johnson-lindenstrauss bound. In: Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms. pp. 868–887 (2011)
7. Bingham, E., Mannila, H.: Random projection in dimensionality reduction: Applications to image and text data. In: in Knowledge Discovery and Data Mining. pp. 245–250. ACM Press (2001)
8. Carraher, L.A., Wilsey, P.A., Moitra, A., , Dey, S.: Multi-probe random projection clustering to secure very large distributed datasets. In: 2nd International Workshop on Privacy and Security of Big Data (Oct 2015)

9. Carraher, L.A., Wilsey, P.A., Moitra, A., , Dey, S.: Random projection clustering on streaming data. In: IEEE 16th International Conference on Data Mining Workshops (ICDMW). pp. 708–715 (Dec 2016)
10. Dasgupta, S.: Experiments with random projection. In: Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence. pp. 143–151. UAI'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
11. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the twentieth annual symposium on Computational geometry. pp. 253–262. SCG '04, ACM, New York, NY, USA (2004). https://doi.org/http://doi.acm.org/10.1145/997817.997857, http://doi.acm.org/10.1145/997817.997857
12. Davis, K.A., Owusu, E.B.: UCI machine learning repository (2016), https://archive.ics.uci.edu/ml/datasets/Smartphone+Dataset+for+Human+Activity+Recognition+%28HAR%29+in+Ambient+Assisted+Living+%28AAL%29
13. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD '96. vol. 96, pp. 226–231 (1996)
14. Fiorini, S.: UCI machine learning repository (2016), https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq
15. Florescu, I., Molyboha, A., Myasnikov, A.: Scaling and convergence of projection sampling. Tech. rep., Stevens Institute of Technology (2009)
16. Fortune, S.: A sweepline algorithm for voronoi diagrams. In: Proceedings of the Second Annual Symposium on Computational Geometry. pp. 313–322. SCG '86, ACM, New York, NY, USA (1986)
17. Gavrilova, M.L.: Lecture notes in computer science. In: Kumar, V., Gavrilova, M., Tan, C., L'Ecuyer, P. (eds.) Comp. Sci. and Its Applications –- ICCSA 2003, vol. 2669, chap. An Explicit Solution for Computing the Euclidean d-dimensional Voronoi Diagram of Spheres in a FP Arithmetic, pp. 827–835. Springer Berlin Heidelberg (2003)
18. Hartigan, J.A., Wong, M.A.: A k-means clustering algorithm. JSTOR: Applied Statistics **28**(1), 100–108 (1979)
19. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification **2**(1), 193–218 (1985)
20. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc of the 13th Annual ACM Symp on Theory of Computing. pp. 604–613. STOC '98, ACM, New York, NY, USA (1998). https://doi.org/http://doi.acm.org/10.1145/276698.276876
21. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recogn. Lett. **31**(8), 651–666 (Jun 2010). https://doi.org/10.1016/j.patrec.2009.09.011
22. Klein, R.: Abstract voronoi diagrams and their applications. In: Computational Geometry and its Applications, vol. 333, pp. 148–157. Springer Berlin Heidelberg (1988)
23. Liu, B., Xia, Y., Yu, P.S.: Clustering through decision tree construction. In: Proceedings of the Ninth International Conference on Information and Knowledge Management. pp. 20–29. CIKM '00, ACM, New York, NY, USA (2000). https://doi.org/10.1145/354756.354775, http://doi.acm.org/10.1145/354756.354775
24. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.: cluster: Cluster Analysis Basics and Extensions (2013), r package version 1.14.4
25. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)

26. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 169–178. KDD '00, ACM, New York, NY, USA (2000). https://doi.org/10.1145/347090.347123
27. Murtagh, F., Legendre, P.: Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion? J. Classif. **31**(3), 274–295 (Oct 2014). https://doi.org/10.1007/s00357-014-9161-z
28. Ng, R.T., Han, J.: Clarans: A method for clustering objects for spatial data mining. IEEE Trans on Knowledge and Data Engineering **14**(5), 1003–1016 (2002)
29. Reyes-Ortiz, J.L., Oneto, L., SamÃ, A., Parra, X., Anguita, D.: UCI machine learning repository (2015), `https://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions`
30. Torres-Sospedra, J., Montoliu, R., Martínez-Usó, A., Arnau, T.J., Avariento, J.P., Benedito-Bordonau, M., Huerta, J.: UCI machine learning repository (2014), `https://archive.ics.uci.edu/ml/datasets/UJIIndoorLoc`
31. Urruty, T., Djeraba, C., Simovici, D.: Clustering by random projections. In: Perner, P. (ed.) Adv. in Data Mining. Theoretical Aspects and Applications, vol. 4597, chap. Lecture Notes in Computer Science, pp. 107–119. Springer (2007). https://doi.org/10.1007/978-3-540-73435-2_9
32. Vempala, S.S.: The Random Projection Method. DIMACS Series, American Mathematical Society (2004)
33. Vergara, A., Fonollosa, J., Rodriguez-Lujan, I., Huerta, R.: UCI machine learning repository (2013), `https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset+at+Different+Concentrations`