**On the E**ff**iciency of K-Means Clustering: Evaluation,**

**Optimization, and Algorithm Selection**

Sheng Wang

New York University

swang@nyu.edu

Yuan Sun

RMIT University

yuan.sun@rmit.edu.au

Zhifeng Bao

RMIT University

zhifeng.bao@rmit.edu.au

**ABSTRACT**

This paper presents a thorough evaluation of the existing methods that accelerate Lloyd’s algorithm for fast k-means clustering. To do so, we analyze the pruning mechanisms of existing methods, and summarize their common pipeline into a unied evaluation framework UniK. UniK embraces a class of well-known methods and enables a ne-grained performance breakdown. Within UniK, we thoroughly evaluate the pros and cons of existing methods using multiple performance metrics on a number of datasets. Furthermore, we derive an optimized algorithm over UniK, which efectively hy- bridizes multiple existing methods for more aggressive pruning. To take this further, we investigate whether the most ecient method for a given clustering task can be automatically selected by machine learning, to benefit practitioners and researchers.

**PVLDB Reference Format:**

Sheng Wang, Yuan Sun, and Zhifeng Bao. On the Eciency of K-Means Clustering: Evaluation, Optimization, and Algorithm Selection. PVLDB, 14(2): 163 - 175, 2021.

[doi:10.14778/3425879.3425887](https://doi.org/10.14778/3425879.3425887)

**PVLDB Artifact Availability:**

The source code, data, and/or other artifacts have been made available at [https://github.com/tgbnhy/fast-kmeans.](https://github.com/tgbnhy/fast-kmeans)

**1 INTRODUCTION**

As one of the most widely-used clustering algorithms, k-means aims to partition a set of n points into k (k < n) clusters where each point is assigned to the cluster with the nearest centroid [[43,](#bookmark7)[70]](#bookmark8). Answering k-means is NP-hard and Lloyd’s algorithm [[48] is a](#bookmark9) standard approach. Essentially, it randomly initializes k centroids, then assigns each point to the cluster with the nearest centroid and renes each centroid iteratively. In each iteration, it needs to compute n · k distances in the assignment step and access n data points in the renement step. Such intensive computations make the Lloyd’s algorithm slow, especially in partitioning large datasets.

Accelerating the Lloyd’s algorithm for k-means clustering has been investigated for more than 20 years since the rst work was published [[58]](#bookmark10). Most of the existing acceleration methods focus on how to reduce intensive distance computations, which can be broadly divided into two categories: 1) the index-based methods

that group and prune points in batch [[27,](#bookmark12) [44,](#bookmark13) [50,](#bookmark14) [58], and 2) the](#bookmark15)

This work is licensed under the Creative Commons BY-NC-ND 4.0 International

License. Visit<https://creativecommons.org/licenses/by-nc-nd/4.0/>to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing[info@vldb.org. Copyright is held by the owner/author(s)](mailto:info@vldb.org). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 14, No. 2 ISSN 2150-8097. [doi:10.14778/3425879.3425887](https://doi.org/10.14778/3425879.3425887)

**k=10**

25

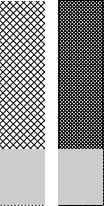
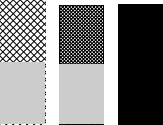
20

15

10

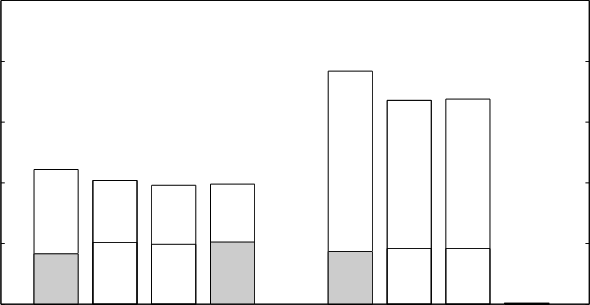
5

0



**Cluster ing t ime (s)**

**Full**  **Regroup**  **Yinyang**  **Index**  **Distance** 



19.2

16.816.9

11.110.2 9.8 9.9

0.1

**BigCross (d=57) NYC-Taxi (d=2)**

**Figure 1: The performances of representative methods, Regroup, Yinyang, Index and Full (usingmultiplepruning mech- anisms) on two clustering datasets, where** d **is the dimen- sionality of the dataset. The gray bar (“Distance”) denotes the time taken by each method to compute distance.**

sequential methods that scan each point one by one and utilize a bound based on triangle inequality to avoid calculating certain distance [[26,](#bookmark16)[34–38,](#bookmark17)[40,](#bookmark18)[41,](#bookmark19)[46,](#bookmark20)[53,](#bookmark21)[59,](#bookmark22)[61,](#bookmark23)[71]](#bookmark24).

**1.1 Motivations**

**Conducting a thorough evaluation of existing** k**-means algo- rithms.** Whilst a large body of methods have been proposed to accelerate the Lloyd’s algorithm for k-means clustering, there is still a lack of thorough evaluation on the effciency of these meth- ods. Moreover, there seems to be some misunderstanding on the performance of certain methods in the literature. For example, the index-based method [[44] was interpreted to be slower compared to](#bookmark25) the sequential methods (e.g., Yinyang [[35],](#bookmark26) Regroup [[61]) when the](#bookmark27) dimensionality of dataset is greater than 20 [[35], and hence was](#bookmark28) discarded by the machine learning (ML) community in its most recent studies [[35,](#bookmark29)[53,](#bookmark30)[61]](#bookmark31). However, we show in Figure [1](#bookmark3)that the

index-based method is in fact relatively fast and has the poten-

tial to signicantly accelerate large-scale clustering when using a proper data structure. This motivates us to conduct a fair and more thorough eciency evaluation on existing methods.

In fact, most existing studies considered reducing the number of distance computations as the main goal to improve the effciency of their methods. However, a method that computes fewer number of distances does not simply guarantee to have a shorter compu- tational time. For example in Figure [1,](#bookmark3) the Full method, which is armed with multiple pruning techniques, has the least number of distance computation, but overall is the slowest on the BigCross dataset. This is because other metrics, such as the number of data



accesses and the time taken to compute a bound for pruning, also contribute to the computational cost. To identify the key metrics, it is essential to analyse the pruning mechanisms of existing methods and extract a unied framework, such that existing methods can well t to enable a ne-grained performance breakdown of existing methods and in turn a more comprehensive evaluation.

**Selecting the best** k**-means algorithm for a given task.** Fast k-means clustering for an arbitrary dataset has attracted much at- tention [[49]](#bookmark32). Unfortunately, there is no single algorithm that is expected to be the “fastest” for clustering all datasets, which is also in line with the [“no free lunch" theorem in optimization [69]](#bookmark33). That calls for an efective approach that is able to select the best algorithm for a given clustering task. However, existing selection criteria are still based on simple rules, e.g., choosing the index-based method when the dimensionality of dataset is less than 20. Given the complex nature of clustering, they are unlikely to work well in practice. Given the large amount of data collected from our evalua- tions, it is natural to apply ML to learn an optimal mapping from a clustering task to the best performing algorithm. Note that the idea of using ML for algorithm selection [[52] has been explored before,](#bookmark34) e.g., meta-learning [[66] and auto-tuning in database management](#bookmark35) [[47,](#bookmark36)[65]](#bookmark37). However, as we will see shortly, it is nontrivial to apply this technique to k-means clustering because problem-specic features have to be carefully designed to describe datasets to be clustered.

**1.2 Our Contributions**

In this paper, we design a unied experimental framework to evalu- ate various existing k-means clustering methods, and design an ML approach to automatically select the best method for a given clus- tering task. More specically, we make the following contributions:

• **Hardware Acceleration.** Parallelization [[73], GPU](#bookmark61) [[72], and](#bookmark62) cache [[23] can accelerate it at physical level](#bookmark63).

• **Approximate Acceleration.** It aims to nd approximate clus- tering results within a bounded error w.r.t. the exact result of the Lloyd’s algorithm, by developing techniques like sampling [[19]](#bookmark64) and mini-batch [[54]](#bookmark65).

• **Fast Convergence.** It uses efcient initialization techniques

• We review the index-based and sequential methods and describe their pruning mechanisms in Sections [3](#bookmark38)and [4.](#bookmark39)

• Inspired by the common pruning pipeline of existing methods, we design a unied evaluation framework UniK in Section [5,](#bookmark40) that enables us to compare existing methods more fairly and com- prehensively. Some key insights obtained are: 1) the index-based method can be very fast even for high-dimensional data, when equipped with a proper data structure such as Ball-tree [[64]; and](#bookmark41)

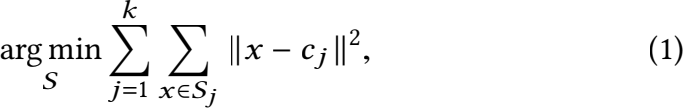
2) no single method can always perform the best across all cases. Detailed evaluations are in Section [7.2.](#bookmark42)

• The above further motivates us to design an adaptive setting for our UniK, which applies the bound-based pruning from sequential methods to assign points in batch without scanning all centroids. In Section [7.2.3, we evaluate our adaptive](#bookmark43) UniK and show that it outperforms the existing k-means algorithms when tested on various real-world datasets.

• To take it further, we adopt ML to automatically select the best method for a given clustering task in Section [6.](#bookmark44) This is achieved by learning from our evaluation records which contain the per- formance of all the methods on various datasets. An evaluation on multiple learning models is conducted in Section [7.3.](#bookmark45)

**2 PRELIMINARIES**

Given a dataset D = {x1, x2, ·· · , xn} ofn points, and each point has d dimensions, k-means aims to partition D into k mutually exclusive subsets S = {S1, S2, ·· · , Sk } to minimize the Sum of Squared Error,



where cj =  x, namely the centroid, is the mean of points

in Sj.

**2.1 Lloyd’s Algorithm**

With the initialized k centroids, the Lloyd’s algorithm for k-means [[48] conducts the assignment and re](#bookmark46)nement in an iterative manner until all the centroids do not change.

**Initialization.** It randomly chooses k points in D as the initial centroids. Normally, k-means++ [[20] is the default initialization](#bookmark47) method which aims to make k centroids far away from each other. **Assignment.** It needs to assign each of then data points to a cluster with the nearest centroid, and therefore requires n · k number of distance computations.

**Re**f**nement.** It needs to read every data point in a cluster to update the centroid. Hence n data accesses are conducted.

E 1. Figure[3](#bookmark48) shows two centroids c1, c2 (red) and ve data points (black) bounded by anode N. The assignment step in Lloyd’s algorithm computes the distance from every point xi to every centroid cj [to determine the nearest cluster.1](#bookmark49)

**2.2 Acceleration**

Given a dataset and k, there are four types of acceleration for fast k-means with the Lloyd’s algorithm:

such as k-means++ [[20,](#bookmark66)[22]](#bookmark67). As Celebi et al. [[28]](#bookmark68) have done an

evaluation on this, it will not be our focus.



• **Exact Lloyd’s Algorithm.** It focuses on reducing the number of distance computations in the Lloyd’s algorithm, and can be integrated with the above methods to reduce their running time.

Clustering has been evaluated from diferent perspectives. For example, Muller et al. [[51] evaluated clustering in subspace projec](#bookmark50)- tions of high-dimensional data; Hassanzadeh et al. [[42] proposed a](#bookmark51) framework to evaluate clustering for duplicate detection. In con- trast, this is the rst work that evaluates all accelerating tricks to reduce distance computations in the exact Lloyd’s algorithm. Figure [2](#bookmark52) summarizes a timeline of fast Lloyd’s algorithms.

**3 INDEX-BASED ALGORITHMS**

By assigning points to the nearest clusters in batch, index-based algorithms have been proposed to support fast k-means, such as kd-tree [[44,](#bookmark53)[58] and Ball-tree](#bookmark54) [[50]](#bookmark55). Intuitively, if an index node that covers m points is assigned to a cluster directly, then m · k number of distance computations and m data accesses can be reduced.

**3.1 Typical Indexes**

**kd-tree.** Indexing the dataset using kd-tree [[24] can accelerate](#bookmark57) k- means with batch-pruning for low dimensional data [[44,](#bookmark58)[58], where](#bookmark59) its intuition is presented in Figure [3(a): node](#bookmark60) N locates in the hy-

perplane H of c1 completely, thus all the points in N are closer

1By default, index i and j refer to data and cluster index respectively in the rest of this paper.

UniK

Index Sequential >

1982 1999 2000 2002 2002 2003 2010 2012 2013 2014 2015 2015 2016 2016 2016 2017 2017 2020

人 人b人 人 人 人 人 人 人 人 b人 人 人 人 人 人 人 人



Lloyd Pelle Ball-tree kd-tree Philli Elka Hame Drak Annu Search Yinyang Heap Expo Drift Vector Regroup Cover Pami20

[[48]](#bookmark69) [[58]](#bookmark70) [[50]](#bookmark71) [[44]](#bookmark72) [[59]](#bookmark73) [[38]](#bookmark74) [[40]](#bookmark75) [[37]](#bookmark76) [[36]](#bookmark77) [[27]](#bookmark78) [[35]](#bookmark79) [[41]](#bookmark80) [[53]](#bookmark81) [[61]](#bookmark82) [[26]](#bookmark83) [[46]](#bookmark84) [[34]](#bookmark85) [[71]](#bookmark86)

**Figure 2: The research timeline of fast Lloyd’s algorithms for** k**-means, where the blue arrows show the successive relationship,** **and the underlines di****erentiate two types of sequential methods covered in Section** [**4.2**](#bookmark87)**(dash lines) and** [**4.3.**](#bookmark88)

|  |  |  |  |
| --- | --- | --- | --- |
| C  `、 C  2  `、 H   |  | | --- | |  |   N | P ; r -- | C 2 |

(a)pruning with kd-tree (b)pruning with Ball-tree

**Figure 3: Example of assignment of node** N **in kd-tree and Ball-tree, based on a hyperplane** H **and a ball with radius** r**.**

to c1 than c2 and then N cannot be assigned to c2 . Unfortunately, bisecting hyperplane in a high-dimensional space using kd-tree (Voronoi diagram is a common method) costs much time with a  [[30]](#bookmark89).

**Ball-tree.** Moore et al. [[50] proposed to use Ball-tree](#bookmark90) [[64] (a.k.a](#bookmark91). metric-tree) to group points using balls, which is a circle in a 2- dimensional space as shown in Figure [3(b)](#bookmark60). All the points in a ball N are closer to centroid c1 than c2 if

Ⅱp - c1 Ⅱ + r < Ⅱp - c2 Ⅱ - r, (2)

where p and r are the center and radius of the ball N. The left-hand side of Equation [2](#bookmark92) is an upper bound on the distance from the points in N to c1 , while the right-hand side is a lower bound on the distance from the points in N to c2 . However, data points in a high-dimensional space are dispersed, resulting in a much bigger radius r and degenerating its pruning power, i.e., it is unlikely to nd a ball N and two centroids c1 and c2 such that Equation [2](#bookmark92) holds. In this case, using Ball-tree can be even slower than directly assigning each point to a cluster, due to the computational overhead of traversing the Ball-tree.

**Other Indexes.** Apart from kd-tree and Ball-tree, there are other data structures in metric space that can be used to prune distance computations [[29,](#bookmark93)[62]. This includes Hierarchical k-means](#bookmark94) tree [[39,](#bookmark95) [67], M-tree](#bookmark96) [[32] and Cover-tree](#bookmark97) [[25,](#bookmark98)[34], to name a few.](#bookmark99) We will evaluate all these typical indexes in our experiment (Section [7.2.1)](#bookmark100).

**3.2 Pre-assignment Search**

Broder et al. [[27] proposed to search around a centroid cj](#bookmark101) the data points that lie within a distance threshold, and directly assign these points to cj, in order to avoid certain distance computations. For example in Figure [3(a), we can search for data points around](#bookmark60) c1 within a distance threshold , and the points found can be assigned to c1 directly, as they are closer to c1 than c2 . Here, we can use an index like kd-tree or Ball-tree to conduct a fast similarity search [[31] to obtain those points within this distance threshold](#bookmark102). Similar operation can be conducted on c2 . Then, we can use Lloyd’s algorithm to assign the rest of data points to a cluster. For those points that cannot be assigned by this method, we then sequentially

scan k centroids to nd a cluster to assign them. In this sense, it also belongs to the sequential algorithm to be introduced in Section [4.](#bookmark103) We denote this method as Search.

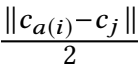
**4 SEQUENTIAL ALGORITHMS**

Essentially, sequential algorithms try to exploit the triangle inequal- ity to derive various forms of distance bounds, and use them to reduce the number of distance computations in the assignment step. However, it is inevitable to access the stored data and update the distance bound of each point. These extra costs actually account for a signicant portion of the total running time.

**4.1 Triangle Inequality**

To check whether a point xi belongs to a cluster of centroid cj[,2](#bookmark104) we can rst compare the lower bound on the distance between xi and cj, denoted as lb(i, j), against the upper bound on the distance from xi to its closest centroid ub(i). If lb(i, j) > ub(i), then xi does not belong to the cluster cj and we do not need to compute the distance between xi and cj. Formally, we denote this inequality as lb(i, j) > ub(i) → a(i) ,j, (3)

where a(i) ,j means that xi is not assigned to the cluster cj. Since the pruning is conducted in a pairwise fashion (point, centroid), we call it local pruning.

Elkan et al. [[38] obtained the lower boundlb(i](#bookmark106), j) in the following way: (1) the distance between centroids is used to derive the rst lower bound lb(i, j) =  [.3](#bookmark107)We call it as inter-bound. (2) The centroid drift from the previous cluster cj, to the current cluster cj is used to derive the second lower bound via the triangle inequality: lb(i, j) = Ⅱxi - cj, Ⅱ - Ⅱcj, - cjⅡ < Ⅱxi - cjⅡ . We call it as drift-bound.

(3) Between these two bounds, the bigger one is chosen as the nal lb(i, j). We denote the algorithm using both inter-bound and drift-bound as Elka.

The inter-bound requires to compute the pairwise distance be- tween centroids, and thus costs  number of computations. The drift-bound uses n · k units of memory, as it stores a bound for each (point, centroid) pair. The main issues of Elka are: 1) it uses much space; 2) the bounds derived might not be very tight. In what follows, we introduce the methods to address each of these issues.

**4.2 Methods with a Focus of Less Bounds**

4.2. 1 Hamerly Algorithm. Instead of storing a lower bound for each centroid, storing a global bound can save much space. Moti-

vated by this, Hamerly [[40] proposed to store the minimum lower](#bookmark108)

2To facilitate our illustration, each cluster is simply represented by its centroid if no ambiguity is caused.

3A similar idea was proposed by Phillips [[59] one year earlier.](#bookmark109)

bound as the global bound. Further, a global lower bound lb(i) for each point xi is used before scanning every centroid, i.e., lb(i) =

max (minj,a/(i)lb(i, j), minj,a/(i) Ⅱ ca/ () -cjⅡ ) where a/ (i) points to

previous cluster ofxi, xi can stay if:lb(i) > ub(i) → a(i) = a/ (i). We denote this algorithm as Hame, and it is known as the global pruning.

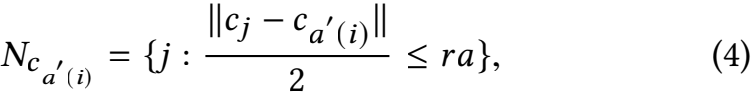
4.2.2 Sort-Centers. Beyond a single bound, Drake et al. [[37] pro](#bookmark110)- posed to store b < k bounds for each point, which can reduce both the storage space and the distance computation by bound. The b points that are selected based on their closeness to the assigned centroid, so it needs to be updated frequently in each iteration. Moreover, the selection of parameter b is also highly dependable on datasets. By default, we use a xed ratio suggested in [[37], i.e](#bookmark111)., b = 「 l. We denote this algorithm as Drak.

4.2.3 Yinyang: Group Pruning. Hame needs to conduct k - 1 local pruning if the global pruning fails, i.e., lb(i) ≤ ub(i), Ding et al. [[35] proposed to divide](#bookmark113) k centroids into t = 「 l groups when k is much bigger than 10, and add a pruning on a group between local pruning and global pruning. This is known as group pruning.

In the rst iteration, k centroids are divided into several groups based on k-means. Each group will maintain a bound rather than each point. However, the xed group may lead to a loose bound with the increase of iterations. Kwedlo et al. [[46] regrouped the](#bookmark114) k centroids in each iteration while [[35] only did it in the](#bookmark115) rst iteration, and the grouping used a more ecient way than k-means, and the group bounds become tighter. We denote these two algorithms as Yinyang and Regroup. An index such as Cover-tree [[25] can also be](#bookmark116) used to group k centroids [[34]](#bookmark117).

4.2.4 Storing Bound Gaps in a Heap. Hamerly et al. [[41] further](#bookmark118) proposed a combination of the global lower bound lb(i) and upper bound ub(i) in Hame, and used their gap lu(i) = lb(i) - ub(i). It can further reduce the space on storing bounds, but needs a heap for each cluster to hold it dynamically, and the heap update incurs extra costs. We denote this algorithm as Heap. Then each point will be attached with such a gap lu(i) and inserted into a queue, unless when lu(i) ≥ 0, it will stay in the current cluster and the distance between all other points and every centroid has to be computed.

4.2.5 Using Centroid Distance Only. Xia et al. [[71] proposed to use](#bookmark119) only the centroid distance to generate a subset of centroids Nca/ (i) as candidates for all the points inside. The main pruning idea is based on the radius ra of each cluster, where ra is the radius of the current assigned cluster ca/(i), i.e., the distance from the centroid to the farthest point in the cluster. In particular, each candidate centroid cj should have a distance of less than 2ra from ca/(i) ; otherwise, the points would choose ca/(i) rather than cj as the nearest centroid:



We name this method as Pami20, and it can save much space.

**4.3 Methods with a Focus of Tighter Bounds**

All the above bounds are based on the triangle inequality over a point and multiple centroids. To further tighten the bounds, L2- Norm is used by [[26,](#bookmark120) [36,](#bookmark121) [41,](#bookmark122) [61]](#bookmark123). We denote the following four algorithms as Annu, Expo, Drift, and Vector.

4.3. 1 Annular Algorithm: Sorting Centers by Norm. Drake et al. [[36,](#bookmark124)[41] proposed an annular algorithm to](#bookmark125) lter the centroids di- rectly. Through using the Norm of centroids, an o-line sorting can estimate a bound to determine the two closest centroids and tighten the upper and lower bounds. The basic idea is to pre-compute the distance from centroid to the origin (a.k.a. norm ⅡcⅡ), and further employ the triangle inequality to derive an annular area around the origin which all the candidate centroids are inside:

J(i) = {j : |ⅡcjⅡ - Ⅱxi Ⅱ| ≤ max (ub(i), Ⅱxi - cj// Ⅱ )}, (5)

where cj// is the second nearest centroid of xi .

4.3.2 Exponion Algorithm: Tighter Lower Bound. To further shrink the annular range of candidates [[36] around the origin, Newling et](#bookmark126) al. [[53] proposed to use a circle range around the assigned centroid](#bookmark127). It flters out the centroids which will not be the nearest, and returns a centroid set J/ (i) which is a subset of J(i):

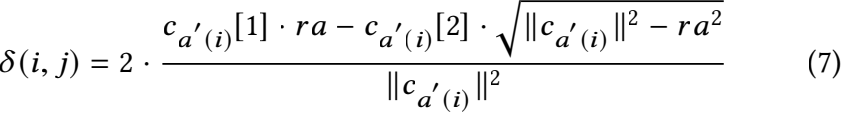
J/ (i) = {j : Ⅱcj - ca/(i)Ⅱ ≤ 2ub(i) + Ⅱ ca/ (i) - c/ (i) Ⅱ }, (6)

4.3.3 Tighter Centroid Dri Bound. In the Elka method, the cen-

troid drift (e.g., Ⅱc - cjⅡ has to be computed every time when the

centroid moves to a new one. It is used to update the lower bound of distance from every point to the new cluster Ⅱx1 - c1 Ⅱ in every iteration, while using triangle inequality among x1 , c1 , and c/ to update cannot get a tighter bound, and nding a smaller drift is crucial. By replacing this in Elka, Rysavy et al. [[61] proposed to](#bookmark129) use the distance between centroid and the origin point (e.g., [0, 0] in two dimension space), and compute a tighter drift δ . Note

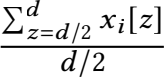
that the distance from ca/(i)to the origin point, i.e., Ⅱca/(i)Ⅱ, can be pre-computed. Here, we only show the drift computation in a 2-dimension case.



It has been proven that δ (i, j) < Ⅱc - cjⅡ [[61], then we can update](#bookmark130)

lb(i, j) = lb(i, j) - δ (i, j). For high-dimensional cases, ca/(i) [1] and ca/(i) [2] can be computed using a more complex conversion in Algorithm 2 of [[61], and we will not elaborate](#bookmark131).

4.3.4 Block Vector. Bottesch et al. [[26] calculated bounds based](#bookmark132) on norms and the Hölder’s inequality to have a tighter bound between the centroid and a point. The idea is to divide each data point into multiple blocks of equal size, similar to dimensionality reduction. By default, the data point is divided into two blocks, i.e.,

x = { Σi[z] ,  }. Then a tighter lower bound can be

obtained by using the pre-computed norm Ⅱxi Ⅱ and ⅡcjⅡ, and inner

product of the block vector x and c .



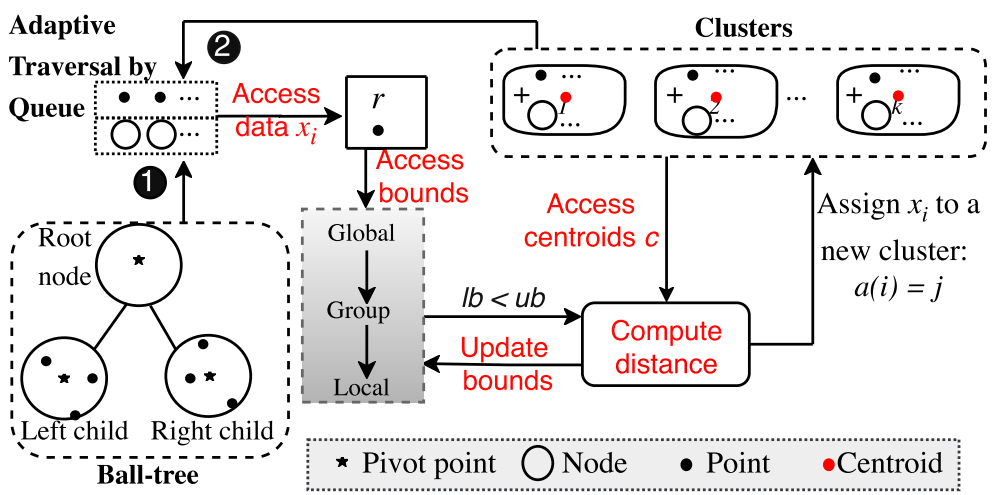
where 〈x , c〉 denotes the inner vector, i.e., 〈x , c〉 = x [1]

·

c [1] + x [2] · c [2].

**5 EVALUATION FRAMEWORK**

After reviewing the sequential methods, we conclude a common pruning pipeline in Figure [4.](#bookmark133) Five core operators (highlighted in red color) execute in the following order: (1) access the data point, (2) then access the global bound to see whether lb is bigger than



**Node Assignment.** Given anodeN, we will assign N to a clusterSj if the gap between the distance from pivot p to the nearest centroid cn1 (p) and the distance from p to the second nearest centroid cj// is larger than 2r, i.e.,

Ⅱp - cj// Ⅱ - r > Ⅱp - cn1 (p)Ⅱ + r → a(N) = n1 (p), (9)

**Figure 4: UniK framework with a common pruning pipeline.**

This is also a more general rule of Equation [2](#bookmark92) and Figure [3(b)](#bookmark60) where c1 and c2 are the two nearest centroids. To achieve the above condition, a straightforward way is to scan k centroids to nd the two nearest centroids forp, similar to [[50]. To further avoid distance](#bookmark138) computation, we use the idea of group pruning (Equation [6) based](#bookmark128) on the pivot point p to prune those centroids that cannot be the nearest two. Before the group pruning, we use a global pruning similar to Equation [9](#bookmark134)by combining r :

ub – (3) if yes, the point maintains in the current cluster; (4) other- wise, access group and local bounds to determine whether to read centroids and compute distances to nd the nearest cluster cj to assign, then it will be inserted into cj’s list of covered points, and (5) then update the bounds.

/

lb(p) - r > ub(i) + r → a(N) = a (N), (10)

After the global and group pruning, we use the local bounds in Equation [3](#bookmark105)by combining ragain to avoid the distance computation between the pivot point p and centroids:

lb(p, j) - r > ub(p) + r → a(N) ,j, (11)

By comparing the bounds for point and node, we combine them into a single pipeline by setting r = 0 when the checked object is a point. Node assignment difers from point assignment in the way that, the node needs to be further split and checked if it is not pruned by Equation [9](#bookmark134)and [10.](#bookmark139)

Before computing the distance when the child node or the point is scanned, we estimate the child’s pivot bounds to the centroids based on the parent-to-child distanceψ using triangle inequality, which will be further used to prune in a new round without computing the real distance. Specically, let Nc denote one child of N, then the upper bound and lower bound of N can be passed to Nc by:

Next, we will present an evaluation framework (UniK) based on the above pipeline, as shown in Algorithm[1.](#bookmark135) UniK supports multiple traversal mechanisms to smoothly switch between dierent index- based and sequential methods. Note that the above pruning pipeline mainly works for point data, and existing index-based methods need to scan k centroids withoutbound-related operations. Hence, before describing UniK, we will introduce an optimized assignment such that the bound-based prunings on both nodes and points can well ft into UniK in a unifed way, followed by an optimized refnement without any more data access. Our experiments show that UniK with these optimizations can further accelerate k-means.

**5.1 Optimizations in UniK**

lb(Nc.p, j) = lb(N.p, j) - Nc., ,

lb(Nc.p) = lb(N.p) - Nc., , ub(Nc.p) = ub(N.p) + Nc.,.

(12)

5.1.1 The Assignment Step. To scan all nodes and points simulta- neously, we dene an advanced node that shares exactly the same property with the point, and meanwhile is attached with more information that can help prune via bounds.

5.1.2 The Incremental Refinement Step. To update centroids after all the points are assigned, a traditional renement will read all the points in each center again, sum them up and get the mean vector. Hence, the whole dataset needs to be scanned again in each iteration. Ding et al. [[35] optimized this by updating the previous](#bookmark140) centroids with those points that changed clusters only, so only a subset of points will be re-accessed. However, we can save this cost if maintaining a sum vector in the assignment.

More specically, we maintain a sum vector so that will be up- dated when an object (node or point) moves in or out during the assignment step. Such an update will be rare when the iteration goes deeper, as only some points change their clusters across it- erations. Since index node is assigned in batch, then in order to avoid accessing all the points under it, each node’s sum vector so and the number of points num can be computed in advance by an iterative procedure [[50] once the index is built](#bookmark141). For example, as shown in the right part of Figure [4, when an object is assigned to a](#bookmark133) new cluster, we will update the sum vector (the symbol of “+”) of the two clusters which this object moves in and out accordingly.

**5.2 Detailed Framework Design**

Algorithm [1](#bookmark142) presents a general form of our unied framework under UniK, which is composed of a queue data structure and four core functions. The queue Q is used to hold both the points and the

**Advanced Index.** Given a tree-structured index T built from D, there are usually three kinds of nodes: root node Nrt, leaf node Nl , and internal node Ni . To enable the pruning capability for k-means, we enrich each node with some extra information as dened below:

DEFINITION 1. (**Node**) N = (p, r, so, ψ, LN, LP, num, h) covers its child nodes LN if N is an internal or root node, or a set of child points LP if N is a leaf node.

where pivot point p is the mean of all the points, radius r is the dis- tance from p to the furthest point inN, the sum vector so represents the sum of all the points under N, ψ = ⅡN/ .p -pⅡ is the distance from p to the pivot of its parent node N/ , num is the number of points covered in the range of N, and h is N’s height (or depth). We also analyze the space cost of this index in our technical report [[68]](#bookmark137) (see Section [A.2)](#bookmark2). To accommodate the assigned points and nodes, we further enrich the clusters as below:

DEFINITION 2. (**Cluster**) Sj = (cj, so, LN, LP, num) covers a list of nodes LN and points LP, and num is the number of points in Sj , i.e., num = |LP | +ΣN ∈LN N.num.

**Algorithm 1:** UniK-k-means (k, D)

**Input:** k: #clusters, D: dataset.

**Output:** k centroids: c = {c1 , . . . , ck }.

**1** Initialization(t, c, Q, S);

max **for** every clusterSj ∈ S **do**

**2 while** c changed or t < t **do**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

Update lb(i , j) with centroid drifts for xi ∈ Sj; Q . add(LN, LP);

**while** Q. isNotEmpty() **do**

o ← Q.poll(), r ← 0;

Update ub and lb ; **if** o is node **then**

 r ← o.#etRadius();

**if** lb(i) - r > ub(i) + r **then**

**11**

**12**

**13**

**14**

**15**

 o stays in current cluster: a(o) ← a, (o);

**else**

#ap =GroupLocalPruning(o , i , j) ; AssignObject(o, S , Q, #ap);

**16**

Refinement(S); t ← t + 1;

**17**

**18 return** {c1 , . . . , ck };

**19 Function** Initialization(t, c, Q, S )**:**

**20** t ← 0, Q ← 0, initialize centroids c = {c1 , · · · , ck };

**21 if** Nrt , null **then**

**22** S1. add(Nrt.so , Nrt); [// index [44]](#bookmark149)

**23 else**

**24** Search on each centroid; [// Search [27]](#bookmark151)

**25** S1. add(D);

**26 Function** GroupLocalPruning(o , i , j)**:**

**27** eJ ← J(i) or Ncj or J, (i); // Expo, Pami20, Annu

**28** min ← +∞, min2 ← +∞;

**29 for** every centroid cj ∈ J **do**

**30** Pass group pruning ; // Yinyang, Regroup

**31 if** ub(i) > lb(i , j) **then**

**32** lb(i , j) ← Ⅱo - cjⅡ;

**33 if** lb(i , j) < min **then**

**34** l a(i) ← j, min ← lb(i , j), min2 ← min;

**35 return** min2 - min;

**36 Function** AssignObject(o, S, Q, #ap)**:**

**37 if** o is node and Equation [9: #ap](#bookmark134) < 2r **then**

**38** Sa,(i) . remove(o.o , o);

**39 for** every child Nc ∈ o.LN **do**

**40** Update Nc’s bound by Equation[12;](#bookmark136)

**41** Sa,(i) . add(Nc.so , Nc), Q.push(Nc);

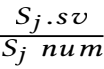
**42 else**

**43 if** a, (i) , a(i) **then**

**44** Sa,(i) . remove(o.so , o);

**45** S a(i). add(o.so , o);

**46 Function** Refinement(S)**:**

**48** cj ←  ;

**47 for** every clusterSj ∈ S **do**

nodes (called as object for unity purpose) to be assigned. When a node cannot be assigned by Equation[9, its child nodes are enqueued](#bookmark134). The four core functions are elaborated as follows.

**Initialization.** After randomly selecting k points as the cen- troids c (line [20), we assign the root node Nrt](#bookmark147) to the rst cluster S1 if an index has been built (line [22); otherwise, we temporally store](#bookmark148) the whole point set D in S1 (line [25)](#bookmark150). Then in each cluster, all the nodes and points are pushed into Q. An object o polled from the queue Q is pruned in the following manner: the global pruning is rst conducted in line [11, we assign o (denoted as a(o)) if the upper](#bookmark145) bound is smaller than the lower bound minus twice of the radius.

**GroupLocalPruning.** If the global pruning fails, we use the group pruning in line [27](#bookmark152)to lter partial centroids. Then we conduct the local pruning (line [31); if it fails, we compute the distance from the](#bookmark154) centroid to the pivot point, and update the two nearest neighbors’ distances (line [34)](#bookmark155).

**AssignObject.** After the scanning of centroids, we obtain the gap between the distance to the two nearest clusters. Then, we check whether the object is a node, and compute the gap using Equa- tion [9](#bookmark134) to see whether we can assign the whole node (line [37)](#bookmark156). If not, we further split the node and push all its children into the queue (line [40–41)](#bookmark157). If it can be assigned or the object is a point, we will further check whether the object should stay in the current cluster (line [43); otherwise, we update the cluster (line](#bookmark159)[44)](#bookmark160).

**Refinement.** We divide the sum vector s" by the number of points inside to update the centroid in line [48.](#bookmark161) The renement can be done without accessing the dataset again, as we have updated the cluster’s sum vector in line [44](#bookmark160)when the object cannot remain in the current cluster.

**5.3 Multiple Traversal Mechanisms**

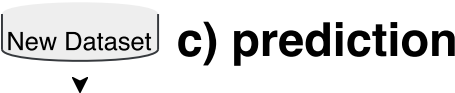
The symbols  in Algorithm [1](#bookmark142) are knobs that indicate whether to apply pruning techniques used in various index-based and se- quential methods which we evaluate. By turning certain knobs on, we can switch to a specic algorithm. For example, by turning on knobs at Lines [4,](#bookmark143)[8,](#bookmark144)[11,](#bookmark145)[14,](#bookmark146)[30,](#bookmark153)[31,](#bookmark154) and others o, the algorithm will run as the Yinyang [[35]](#bookmark163). Formally, we formulate the knobs as below:

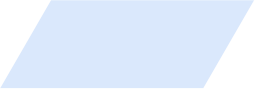
DEFINITION 3. **(Knob Con**f**guration)** A knob = {0,1} controls a setting (selection),e.g., whether to use index or not. A knob configura- tion θ ∈ Θ is a vector of all knobs in Algorithm [1, e.g.,](#bookmark142) [0, 1, 0, ·· · , 1], where Θ denotes the configuration space.

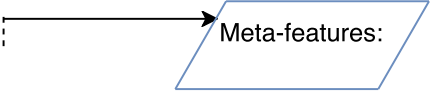
Most existing algorithms can t into UniK by projecting to a specic knob conguration[,4](#bookmark164) and our optimization proposed in Section [5.1](#bookmark136) can be activated by a new knob conguration[.5](#bookmark165) For example in Figure[4](#bookmark133)we present two traversal mechanisms (i.e. parts “!1 ” and “!2”) based on knobs, where we traverse from the root node by using!1 in the rst iteration, but in the following iterations we traverse from the nodes maintained in the current cluster by using !2 . This is because the pruning effect is not always good especially for high dimensional data, and it will still cost extra time in the tree traversal in next iteration if starting from the root

4They run in the same way as presented in the original papers, and will not incur any extra costs. Thus, it is a fair evaluation for any algorithm under comparison.

5 By enabling all bound knobs, we will get the Full method in Figure [1.](#bookmark3) We also argue that this formulation will be useful to cater for more new congurations in Θ to be explored by future studies, but it is not the focus of this paper.









 +





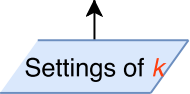
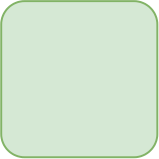




+ a) running



running unik with selective settings.



Data set 2



Index

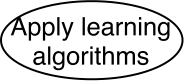


unik



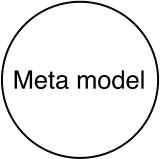
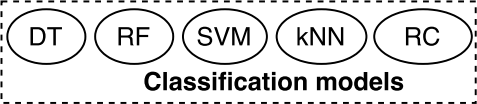
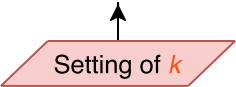








id



**Figure 5: A basic deci- sion tree (BDT) for sim-**

b) training

we train the model by applying various

model, we predict the

optimal algorithm for the

new data set and k.

|  |
| --- |
|  |

|  |
| --- |
|  |

**ple algorithm tuning. Figure 6: Three modules of UTune: a) selective running, b) meta-model training, and c) prediction.**

node. If most nodes in the index can be pruned, traversal from the root (i.e., !1) in each iteration can gain better performance than scanning nodes from clusters. In Algorithm [1, we implement such](#bookmark142) an adaptive mechanism as scanning both the points and nodes as an object o with a radius r, where r = 0 when o is a point.

To be more adaptive, in the rst and second iterations of UniK, we can traverse from the root !1 and current nodes in the clusters !2, respectively (as shown in line [22](#bookmark148)of Algorithm [1, where we put](#bookmark142) the root node Nrt into S1 before the rst iteration). Then we can compare the assignment time of these two iterations. If the time spent on the rst iteration !1 is bigger than that on the second iteration !2, we give up traversing from the root in subsequent iterations, and scan the current nodes and points maintained in each cluster; otherwise, we empty the point and node list of each cluster, and then push the root node at the end of each iteration to keep beneting from the index in multiple iterations. We denote these two traversal styles as index-single and index-multiple.

By default, we set the index as Ball-tree and set the bound con- guration same as Yinyang (Section [4.2.3)](#bookmark112). Our experiments in Section [7.2.3](#bookmark167)and [7.3](#bookmark168) show the the superiority of such a congura- tion over both Ball-tree and Yinyang in some datasets. When we disable all the bound congurations, it will be a pure index-based method [[50]](#bookmark169). Hence, besides multiple knobs of bound conguration, we also have four conguration knobs on the index traversal: 1) not using index; 2) pure; 3) index-single; 4) index-multiple. Next, we will further discuss how to choose the right knobs to turn on.

**6 CONFIGURATION AUTO-TUNING**

In this section, we study how to select a fast algorithm for a given clustering task. This is critical to both an evaluation framework (like this paper) and practitioners. Our literature review shows that existing studies [[34,](#bookmark170) [35] rely on simple yet fuzzy decision rules,](#bookmark171) e.g., not using index-based method for high-dimensional data or choosing Yinyang when k is big. In Figure [5,](#bookmark172) we illustrate a basic decision tree (BDT) based on these rules. However, our experiments show this BDT does not work very well.

Therefore, we choose to train an ML model based on our eval- uation data to automatically select a fast algorithm for a given clustering task. The algorithm selection is equivalent to a problem of tuning “knobs” in our evaluation framework UniK (Algorithm[1),](#bookmark142) in the sense that each of the existing algorithms corresponds to a unique knob conguration.

**An Overview.** We model the knob conguration (algorithm selec- tion) problem as a typical classication problem, where our goal is to predict the best knob conguration for a given clustering task.

**Table 1: A summary of features F** = {f 1, f 2, ·· · , fx}**.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Feature** | **Description** | **Normalize** | |
| Basic | n | The scale of dataset | - | |
| k | Number of clusters | - | |
| d | Dimensionality of dataset | - | |
| Tree | h(T)  |Ni |, |Nl | μ(h), σ (h) | Height of index tree T  #Internal & leaf nodes  Imbalance of tree | log2 n  f log2 | n f  n f |
| Leaf | μ(r), σ (r)  μ(ψ), σ (ψ)  μ(|Lp|), σ (|Lp|) | Radius of leaf nodes  Distance to parent node #Covered points in Ni | Nrt .r  Nrt .r  f | |

To this end, we extract meta-features to describe clustering datasets, and generate class labels (i.e., ground truth) from our evaluation logs that contain the records of which conguration performing the best for a particular dataset. We then feed the meta-features and class labels into an o-the-shelf classication algorithm to learn a mapping from a clustering dataset (with certain features) to the best performing algorithm conguration. We can then use the trained model to predict a good conguration for clustering a given dataset[.6](#bookmark173)We name our auto-tuning tool as UTune, and illustrate its three modules in Figure [6.](#bookmark174)

**6.1 Generating Training Data**

**Class Label Generation.** Since the knob configuration space is large, it is computationally intractable to try all possible knob con- gurations and label the dataset with the best-performing cong- uration. Thus, we only focus on a few knob congurations corre- sponding to the high-performing existing methods as our selection pool [[65]](#bookmark176). The pseudocode of our selective running can be found in Algorithm [2](#bookmark5) of our technical report [[68], and the main idea is](#bookmark177) three-fold: 1) we limit the number of iterations tmax as the run- ning time of each is similar after several iterations (see Figure [13);](#bookmark178) 2) we exclude those algorithms that have low rank during our evaluation, e.g., Search [[27], and our experiments (see Figure](#bookmark179)[12)](#bookmark180) show that only ve methods always have high ranks; 3) we test index optimizations if the pure index-based method outperforms sequential methods. Such a selective running enables us to generate more training data within a given amount of time, and thus further improves the prediction accuracy (see Table [5)](#bookmark181).

**Meta-Feature Extraction.** We extract a feature vector **F** to de- scribe (or represent) a dataset, such as dimensionality, scale, and k

6Note that the learning component in this paper will not tune any existing algorithm or parameter; its main task is to predict the best one among existing algorithms and our optimized algorithm. Moreover, our learning model is not limited to some specic learning methods such as deep learning; most classication models (such as decision tree, SVM, and kNN) can be trained to complete this prediction task.

**Table 2: An overview of datasets, the index construction time (second), and #nodes of Ball-tree.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ref.** | **Name** | n | d | **Time** | **#Nodes** | **Used by** |
| [[1]](#bookmark183) [[2]](#bookmark184) [[11]](#bookmark185) [[3]](#bookmark186) [[14]](#bookmark187) [[13]](#bookmark188) [[5]](#bookmark189) [[15]](#bookmark190) [[12]](#bookmark191) [[10]](#bookmark192) [[16]](#bookmark193) [[9]](#bookmark194) | BigCross  Conflong  Covtype  Europe  KeggDirect  KeggUndirect  NYC-Taxi  Skin  Power  RoadNetwork  US-Census  Mnist | 1.16M  165k 581k 169k 53.4k 65.5k 3.5M  245k  2.07M  434k  2.45M  60k | 57  3  55  2  24  29  2  4  9  4  68  784 | 10.8  0.26 3.87 0.27 0.17 0.31 8.7 0.33  4.3 0.55 204 4.8 | 183k 21.8k 88.3k 11.2k 2.8k 4.5k 228k 21.2k 43.7k 6.9k  135k 7.3k | [[63]](#bookmark195)  [[53]](#bookmark196)  [[38,](#bookmark197)[40,](#bookmark198)[41]](#bookmark199)  [[53]](#bookmark200) [[35,](#bookmark201)[53]](#bookmark202) [[35,](#bookmark203)[53]](#bookmark204)  -  [[53]](#bookmark205) [[34]](#bookmark206) [[35]](#bookmark207) [[53]](#bookmark208)  [[35,](#bookmark209)[40,](#bookmark210)[53]](#bookmark211) |

shown in Table [1.](#bookmark166) In addition to the basic features, we also extract novel and more complex features that can capture certain proper- ties of data distribution based on our built index. Recall the pruning mechanisms in Section [5,](#bookmark88) they work well when data shows good assembling distribution. The index construction actually conducts a more in-depth scanning of the data and reveals whether the data assembles well in the space. Specifically, the information we got includes the tree depth h(T), number of leaf nodes |Nl |, number of internal nodes |Ni |, and imbalance of tree (i.e., the mean value and standard deviation of all leaf nodes’ heights h).

Further, we extract more features of all the leaf nodes, includ- ing the radius of nodes r, the distance from child to parent nodes ψ , and the number of points in the leaf nodes |Lp|, during index construction. Specifically, we select all the leaf nodes N and extract their (r, ψ, |Lp|), then compute their mean μ and standard deviation σ , which are all normalized by their maximum values in the tree.

**6.2 Meta-Model Training and Prediction**

We model the algorithm selection as a multi-label classication problem in terms of using an index or not, and using which one of the ve bound congurations. The prediction needs to be conducted in two parts based on our two ground truth les which shows the rank of various index and bound congurations, respectively. Firstly, we predict the optimal bound conguration. Secondly, we predict the index conguration and whether the bound should be combined, as mentioned in Section[5.3. Then we combine the results from these](#bookmark162) two predictions and generate the nal conguration. For a new clustering task, we can extract the features based on our built index (e.g., Ball-tree), and use the learned model to predict a high-ranked

knob conguration of algorithms.

**7 EXPERIMENTAL EVALUATIONS**

Our evaluation seeks to answer the following questions:

• Which index structure is proper for index-based methods?

• How does the performance of sequential methods vary?

• Can our evaluation framework UniK enable further improvement on the performance of existing clustering methods?

• Can UTune predict a fast algorithm through learning?

**7.1 Experimental Setup**

**Implementation.** We implemented all the algorithms in Java 1.8.0\_201, and used Scikit-learn 0.22.2 [[4,](#bookmark213) [57] to train our classi](#bookmark214)- cation models. All experiments were performed on a server using

an Intel Xeon E5 CPU with 256 GB RAM running RHEL v6.3 Linux. Our code is available at [[6] for reproducibility.](#bookmark215)

**Parameter Settings.** The performance of index-based methods and three sequential algorithms (i.e., Yinyang [[35],](#bookmark216) Drak [[37], and](#bookmark217) Vector [[26]) will be a](#bookmark218)ected by parameters. To be fair, we follow the suggestions of those sequential methods and set xed parameters, detailed settings can be found in their description in Section [4.](#bookmark103) The eects on index-based methods will be studied in Section [7.2.1.](#bookmark219)

**Measurement.** Same as the traditional methods [[35,](#bookmark220)[55], we mea](#bookmark221)- sure the running time and the percentage of pruned distance com- putation (i.e., pruning power). In order to run more rounds of exper- iments, we record the total running time of the rst ten iterations (after which the running time usually becomes stable as shown in Figure [13)](#bookmark222). Moreover, we measure #data access, bound access, bound updates, and footprint. For each measurement above, we report the average value across ten sets of randomly initialized centroids using k-means++ [[20]](#bookmark223).

**Datasets.** We select a range of real-world datasets (Table [2), most](#bookmark182) of them are from the UCI repositories [[21], and also used in the](#bookmark224) state-of-the-art such as [[35,](#bookmark225)[53]](#bookmark226). Moreover, we introduce several recent new datasets, including pick-up locations of NYC taxi trips. The datasets’ hyperlinks can be found in the reference.

**7.2 Evaluation of Existing Methods in UniK**

7.2.1 Index-based Methods. We implemented ve indices: kd-tree, Hierarchical k-means tree (HKT) [[39], Ball-tree](#bookmark227) [[56], M-tree](#bookmark228) [[32],](#bookmark229) and Cover-tree [[25] covered in Section](#bookmark230) [3.1.](#bookmark56) The latter four that bound points by a radius can be easily extended to support UniK.

**Index Construction.** Figure [7](#bookmark231) compares these ve indices over BigCross, and shows the construction time and clustering time w.r.t. the dimension d and the data scale n, respectively. Here, we set n = 10, 000 when varying d, and M-tree is quite slow for a bigger n; that also explains why we ignore its performance.

Observations. (1) With the increase of d and n, the construction time increases and is more sensitive to n, but it is still tolerable for Ball-tree, Cover-tree, and kd-tree. (2) In average, Ball-tree is the fastest in clustering and 2nd fastest in index construction. (3) Even though kd-tree is the fastest in index construction, its leaf-nodes can only cover one point whereas Ball-tree can cover multiple points. Thus, kd-tree has many more nodes than Ball-tree (the ratio is around the capacity f = 30 according to our experiments),and Ball- tree is more applicable for large-scale clustering. (4) Columns 5 and 6 of Table [2](#bookmark182)also show the index construction time and the number of nodes of Ball-tree. We nd that the index of most datasets can be built within ten seconds, which means the delay of index building before clustering is tolerable.

**Clustering E**ff**ciency.** Observations. (1) With the increase of data scale, the cost of clustering rises dramatically for every index, but Ball-tree still beats other indices. (2) When the dimensionality in- creases, kd-tree’s performance degrades the most due to its complex pruning mechanism by using hyperplane; other indices are not im- pacted much as they use a radius to prune without any extra cost. (3) A bigger k makes the clustering slower, in a linear manner.

Our Choice. Thus, we choose Ball-tree [[56] as](#bookmark232) UniK’s default index structure and conduct a further comparison with sequential meth- ods and our proposed UniK and UTune. The space cost of the index

kd-tree  Ball-tree 

M-tree 

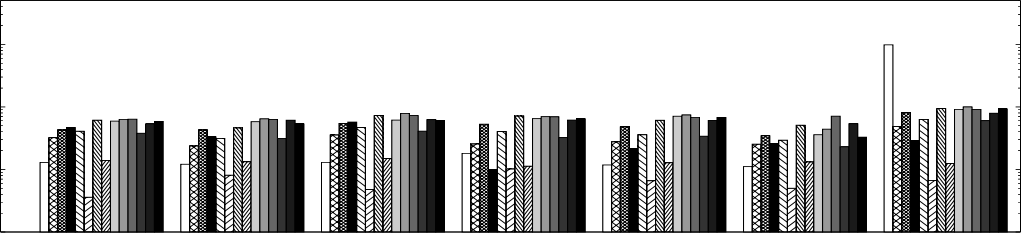
HKT  Cover-tree

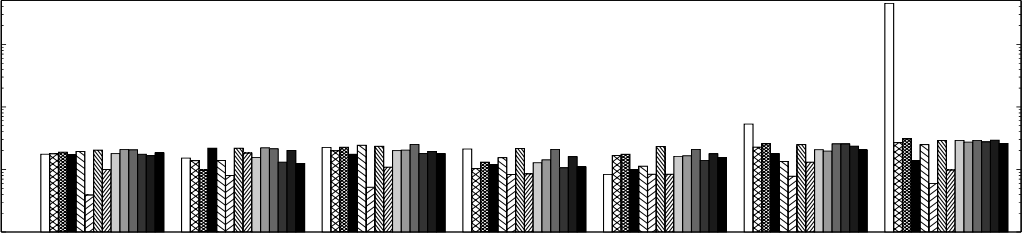


0

**Figure 7: Construction and clustering performance of** **ve index structures.**

INDE  Elka  Hame  Drak  Annu  Search  Yinyang  Heap  Expo  Vector  Regroup  Drift  Pami20  Full 

**k=10 k=100**

100

**Speedup**

**Speedup**

100

10

10

1

1

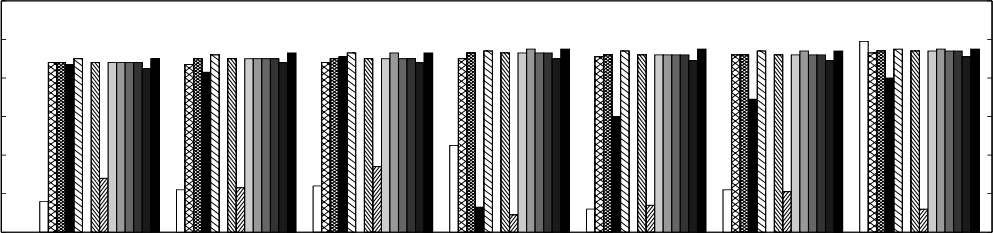
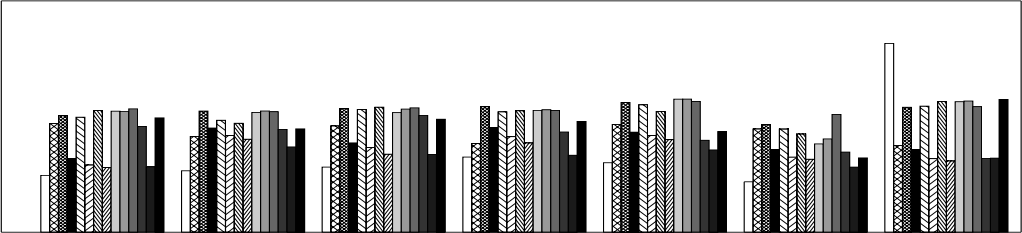
0.1

0.1

Cross Conf Covt Euro KeggD Kegg NYC Cross Conf Covt Euro KeggD Kegg NYC

**Figure 8: Overall speedup in various datasets when setting** k **as 10 and 100, respectively.**

**Ratio of Pruned Computation**

1.2 1 0.8 0.6 0.4 0.2 0

**Speedup of Refinement**

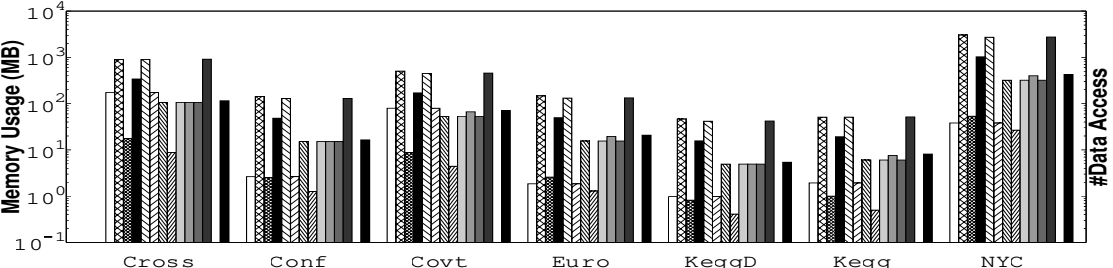
100

10

1

0.1

Cross Conf Covt Euro KeggD Kegg NYC Cross Conf Covt Euro KeggD Kegg NYC

**computation pruning ratio (**k = 100**).**

**Figure 9: Re****nement’s speedup**

**and distance**

10

10

10

10

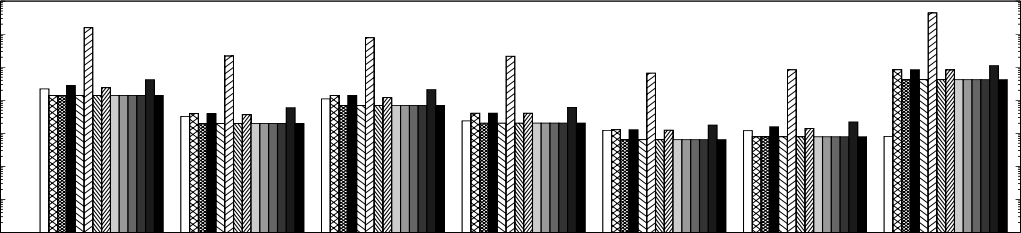
10

10

10

10

10



9

8

7

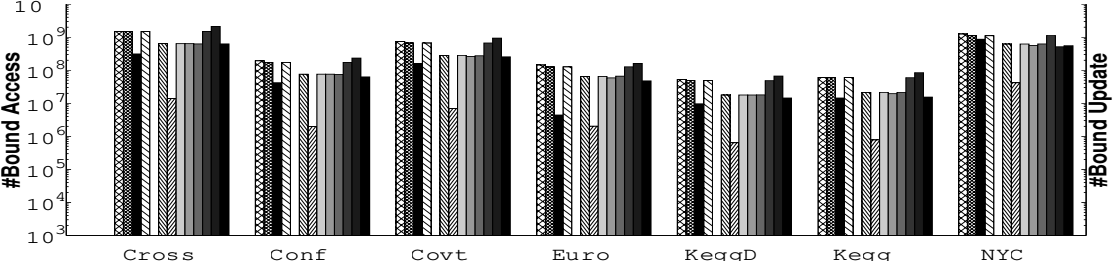
6

5

4

3

Cross Conf Covt Euro KeggD Kegg NYC

**(index) and data accesses (**k = 100**).**

**Figure 10: Statistics on the footprint of bound**

10 10

10

10

10

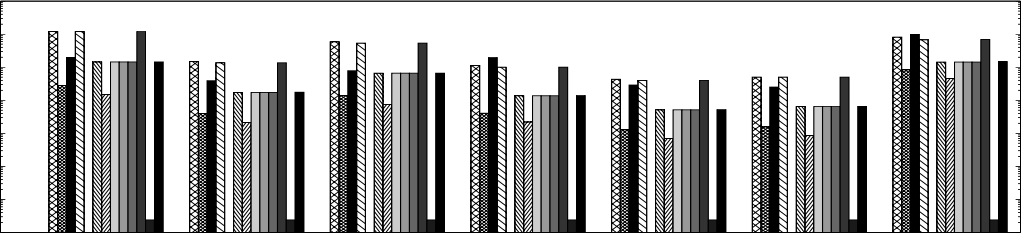
10

10

10

10

10



9

8

7

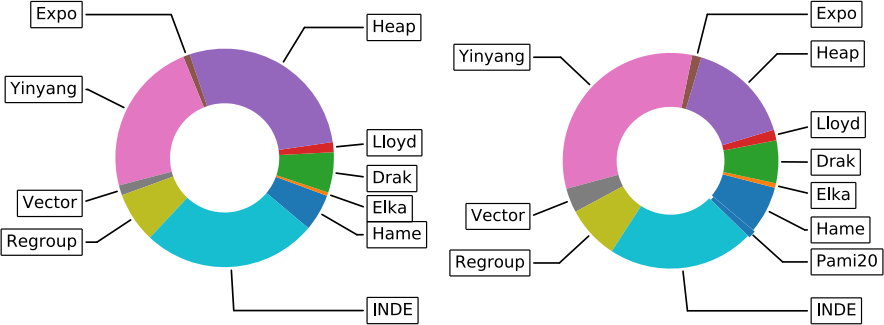
6

5

4

3

Cross Conf Covt Euro KeggD Kegg NYC **Figure 11: Statistics on the bound accesses and updates (**k = 100**).**



**Figure 12: Leaderboard of sequential methods as top 1 & 3.**

is not high compared with most sequential methods, which will be presented in Figure [10. Moreover, setting a proper capacity f](#bookmark231) for leaf nodes in M-tree, HKT, and Ball-tree is crucial[,7](#bookmark233) and a larger f can save more space as the number of nodes will decrease. To balance the space and eciency, we set a relatively small capacity (i.e. 30), and a sensitivity test of capacity will be conducted later in Figure[14, together with](#bookmark234) UniK. It shows that the clustering eciency is not aected much by the capacity.

7.2.2 Sequential Methods. **Learderboard.** Figure [12](#bookmark180) shows a

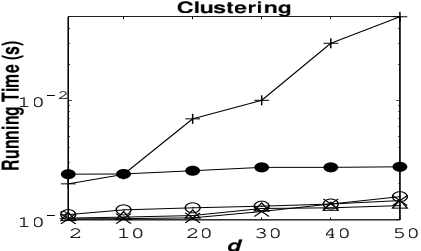
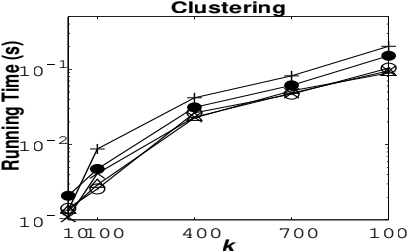
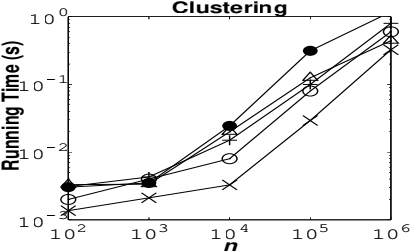
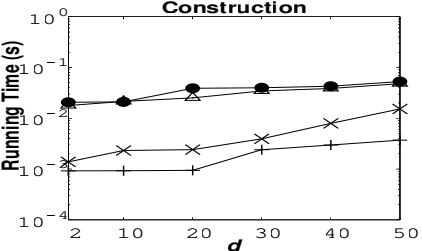
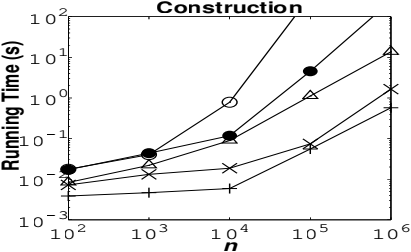
7 Cover-tree and kd-tree do not have a parameter on capacity.

leaderboard of various sequential methods by setting dierent pa- rameter settings (k, n, and d) across all the datasets in Table[2.](#bookmark182) We can observe that ve sequential methods have competitive perfor- mance: Hame, Drak, Heap, Yinyang, and Regroup.

Our Choice. Thus, we will use these ve sequential algorithms as the selection pool in our auto-tuning model UTune.

**Speedup and Pruning Ratio.** We rst investigate the speedup and the pruning ratio in distance computation. Figure [8](#bookmark231)shows the overall speedup over the Lloyd’s algorithm, compared with the rep- resentative index-based method (INDE): Ball-tree. Since assignment occupies most time of the clustering time and it shows a similar trend with the overall speedup, we ignore it here, and it can be found in our technical report [[68] (see Section](#bookmark235)[A.3)](#bookmark11). Interestingly, the improvement of renement (Figure [9) using our incremental](#bookmark231) method signicantly improves the eciency for all algorithms.

On the low-dimensional NYC dataset, we observe that the index-based method can beat all existing sequential methods in term of running time. This also occurs in several relatively high- dimensional datasets when k = 10, such as KeggD and Kegg. Among all the sequential methods, the Regroup and Yinyang are two fastest



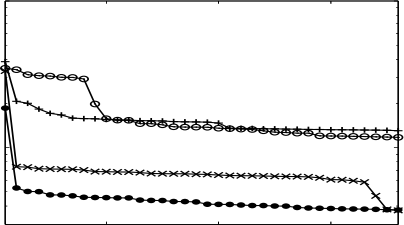
171

LLOY  SEQU  INDE  UniK 

**BigCross**

**KeggUndirected**

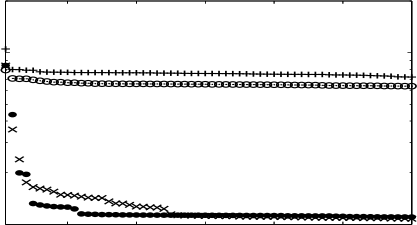
1



**Running Time (s)**

**Running Time (s)**

0.1



10

1

1 10 20 30 40 50 60

1 10 20 30

***iteration***

***iteration***

**Figure 13: Running time of each iteration over the Keg- gUndirected (**d = 29**) and BigCross dataset (**d = 57**).**

**Speedup**

**Speedup**

8

7

6

5

4

3

2

1

0

5

4

3

2

1

0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

|  |
| --- |
|  |

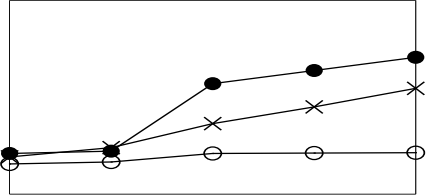
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |

|  |  |
| --- | --- |
|  |  |
|  | |
|  | |

10 20 30 40 50 60

1 5 10 50100 500

***n (x1000)***

***f***

**Speedup**

**Speedup**

5

4

3

2

1

0

10

8

6

4

2

0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |

10 200 100

10 20 30 40 50

***d***

***k***

**Figure 14: Sensitivity test on capacity** f **, data scale** n**,** k**, and dimension** d **over the BigCross dataset.**

**Table 3: Bound and data accesses in the** **rst iteration.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset** | **Criteria** | Lloyd | SEQU | INDE | UniK |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cross k = 100 | Time (s)  Pruned  Bound  Point  Node | 96.0  0  0  100M  0 | 33.1  84%  1.5B  30M  0 | 55.3  45%  0  15.8M  931k | 16.3 91% 0.9B 9.8M 108k |

methods on most datasets. We also observe that the speedup is not consistent with the pruning ratio, e.g., the index is almost 150 times (it is 400 when k = 10) faster on NYC, while its pruning ratio is only 10% (30% when k = 10) more than others. A reverse trend happens in the fully optimized algorithm (Full) which has the highest pruning ratio but very low eciency.

The above observations verify our claims that **index-based method can be very fast, and a higher pruning ratio cannot simply guarantee better performance**.

**Role of Data Access & Bound Access/Update.** Figure [10](#bookmark231) presents the memory usage of each algorithm. We nd: among all the space-saving sequential methods, Heap can save the most, and index-based method’s overhead is also much less than those sequential methods, and it will not increase with the rise of k. Moreover in Figure [11,](#bookmark231) the index-based method has much less data access, which can explain why it is much faster; Yinyang has much less bound access and update which can explain why it is faster than other methods that have very similar pruning ratio. This also reveals that **data access, bound access and bound updates are three crucial factors that should also be considered in the fu-**

**ture algorithm design and evaluation**. A similar performance breakdown of setting k = 10 and more analysis can be found in our technical report [[68] (see Section](#bookmark237)[A.3)](#bookmark11).

7.2.3 Our Unified Method UniK. Next, we compare UniK with (ar- guably the best) index-based method (INDE): Ball-tree and sequen- tial method (SEQU): Yinyang, and conduct a series of ne-grained

evaluations on specic datasets. Detailed results on each dataset are in Table [6](#bookmark238)when comparing with UTune later.

**Running Time Per Iteration.** Figure [13](#bookmark239)shows the running time on each iteration. We observe that the time spent per iteration decreases sharply in the rst few iterations and then becomes stable. UniK is the fastest because of our adaptive traversal mechanism, and INDE and SEQU dominate each other in two datasets respectively.

**Access on Data and Bound.** Specically, we count the number of bound accesses, data accesses, and distance computations for BigCross, as shown in Table [3.](#bookmark236) The number of bound accesses and data accesses of SEQU is much higher than INDE and UniK, and UniK has the minimum number of accesses. For example, SEQU needs 1.5 billion number of bound accesses, and UniK only needs 0.9 billion, as most points have been pruned by index.

**Robustness to Parameters.** We test the robustness of UniK tovari- ous parameters in Figure[14, especially the capacity of the leaf nodes](#bookmark234). By increasing the capacity, the performance decreases slightly but does not uctuate much. Like other two methods, UniK’s speedup rises slightly when increasing n, d, and k.

7.2.4 Summary of Lessons Learned. We rst summarize some key insights that might dier from existing studies:

• Ball-tree is very fast in both construction and clustering. For low-dimensional spatial datasets such as NYC, Ball-tree can beat all sequential methods to a great extent. Moreover, Ball-tree also wins in high-dimensional datasets such as BigCross and Kegg when k equals 10. This is because the data points assemble well, and having a small r helps the pruning in batch. Hence, by choos- ing the right index structure (Ball-tree), the index-based method can also perform well in high-dimensional data, which breaks the traditional conclusions towards the index-based methods [[53]](#bookmark240).

• Among all the sequential methods, there are ve methods alter- nately being the fastest in various clustering tasks. They are: Hame, Drak, Heap, Yinyang, and Regroup. As a common target on reducing space on bounds, they also reduce the overhead to maintain and update those bounds, which further leads to faster performance.

• Several tight bounds proposed do not work well and their space consumption is high (e.g. [[26,](#bookmark241)[61]), probably because frequent](#bookmark242) bound access, comparison, and expensive updates are needed, and large amount of bounds have to be stored which further increases maintenance costs. Recall Section [4.3,](#bookmark88) we can also see that it is complex to compute and update these bounds.

We also have several new and deeper insights:

• By integrating the index-based and sequential methods, UniK achieves an average of 53% performance improvement in our tested clustering tasks, and even up to 73% when k ≥ 100 (details in Table [6). Compared with index-based methods, it avoids many](#bookmark243) distance computations when assigning nodes by bound-based pruning. Compared with sequential methods, it prevents inten- sive data accesses and distance computations simultaneously using batch pruning.

• We have shown the aggregate rank of each method over all datasets in term of eciency using pie charts in Figure [12.](#bookmark180) Fur- ther, we rate them in Table [4](#bookmark244)based on multiple metrics we used, and more experiments results are not shown due to space limit. Here we do not list other index-based methods as Ball-tree clearly

**Table 4: Evaluation Summary of Section** [**7.2.**](#bookmark219) **The darker the circle** 。**, the higher degree of its corresponding criteria.**

**Users**  **Beginners**  **Researchers**  **Domain**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Criteria**  **Algorithm** | Leaderboard | Space-saving | Parameter-free | Fewer | data access | Fewer | bound access | Fewer | distance | Robustness | Ease of | implementation | Analysts |

Ball-tree [[50]](#bookmark245)

O● ● ● O●

● ● O● ● ● O● O● ● O●

O● O● O●

O.

O O. O. O●

O● O● O● O. O. O. O● O● ●

O● O.

O● O● O. O● O● O. O. O. O● O● O●

O● O O●

O● O● ● O● O● O. O. O● ● O●

Elka [[38]](#bookmark246)

O.

O O● O●

O ● ● O O.

O.

O. O.

O● O● O● O● O. O. O. O● O● O●

Hame [[40]](#bookmark247)

O● O O. O● O● O. O. O. O● O● O●

Drak [[37]](#bookmark248)

Annu [[36]](#bookmark249)

O●

Heap [[41]](#bookmark250)

O O

O●

Yinyang [[35]](#bookmark251)

Expo [[53]](#bookmark252)

Drift [[61]](#bookmark253)

O.

Vector [[26]](#bookmark254)

O

O

Regroup [[61]](#bookmark255)

O● O ●

Pami20 [[71]](#bookmark256)

O● O

UniK

dominates the rest. We divided the metrics into two groups: be- ginners and researchers, where the former is for general data mining users, and the latter is for research purposes on acceler- ating k-means over existing algorithms. With our rates, users can choose a proper algorithm based on their demands.

• Besides the metrics we have evaluated, Table [4](#bookmark244)also includes the robustness to various clustering tasks and ease of implementation that system engineers can refer to. For domain-specific analysts, we highly suggest spatial analysts () to adapt Ball-tree as it can bring signicant acceleration for low-dimensional spatial data. Based on users’ computing devices, we recommend that analysts with servers () can choose Yinyang or our UniK, and Hame and Pami20 will be proper for laptops () with limited memory.

**7.3 Evaluation of Auto-tuning**

Based on the above comprehensive evaluations on index-based and sequential methods, we choose two representative algorithms, one for each class, denoted as INDE (Ball-tree) and SEQU (Yinyang)[.8](#bookmark257)Next we compare them with UniK (with our default index and bound confgurations), and our UTune with various learning models.

7.3.1 Model Training. **Ground Truth Generation.** Since avail- able public datasets for clustering are not that many, when run- ning clustering methods to generate the ground truth, we alter k = {10, 100, 200, 400, 600, 800, 1000}, n = {103, 104, 105, 106}, and d = {10, 20, 30, 40, 50} over all the datasets in Table [2.](#bookmark182) In the appen- dix of technical report [[68] (see Figure](#bookmark258)[18), we present the e](#bookmark6)ciency of the full running and selective running (proposed in Section [6.1](#bookmark175) and we used ve methods) on the datasets chosen based on our leaderboards (see Figure [12). We can observe that selective running](#bookmark180) is much faster, as it skips those slow methods and saves much time to run more parameter settings and get more training data, which can further improve the precision, as later exhibited in Table [5.](#bookmark259)

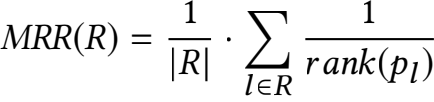
**Adopted ML Models.** With the ground truth obtained, we adopt most classical classication models [[45] for training and prediction:](#bookmark260) decision trees (**DT**),random forests (**RF**),k nearest neighbor (**kNN**),

support vector machine (**SVM**), and Ridge linear classier (**RC**)[.9](#bookmark261)

8These two algorithms also compose the BDT in Figure[5,](#bookmark174) where Yinyang is same as Hame by setting t = 1 when k < 50.

9We divide the ground truth into two parts: 70% for training and 30% for testing.

**Prediction Accuracy.** We adopt a rank-aware quality metric called [mean reciprocal rank (**MRR**) [33] to measure the precision](#bookmark262). Given a set of testing records R, our loss function is dened as below:

 (13)

where rank(pl ) denotes the ranking of prediction pl in the labeled ground truth.

Table [5](#bookmark263)shows the MRR of dierent models (the BDT in Figure [5,](#bookmark174) DT, RF, SVM, kNN, RC) in predicting the index conguration (i.e. Index@MRR) and the choice of bound to be used (i.e. Bound@MRR). The MRR result can be further interpreted from two dimensions: (i) For each model, we distinguish the MRR precision over the training data obtained from the full running and the selective running (high- lighted as a prex “S-”), respectively; (ii) For each model, we keep adding three groups of features in the training phase, namely basic features, index features (Tree), and advanced features on leaf level (Leaf ), to verify their effectiveness. Details on these features are in Table [1.](#bookmark166) For completeness purpose, we also report the training and prediction time in our technical report [[68] (see Table](#bookmark264)[7)](#bookmark1).

Observations. (1) Within the same limited time, selective running has higher precision and can achieve 92% if using decision tree (with a depth of 10) or SVM, while BDT that relies on fuzzy rules only achieves 43%. This is because selective running manages to generate more training records than full running (e.g., 1600 vs. 436 in this case). (2) With more index and leaf features, we can have a higher precision than using basic features only when using the selective running ground-truth. (3) Among all the classier models, the decision tree has the highest precision and it is also very fast for both training and prediction.

7.3.2 Verification. Among all the prediction models of high accu- racy, we select the decision tree (DT) to support UTune. Then, we compare UTune with the representatives: INDE, SEQU, UniK, to verify whether our predicted conguration works well. Table [6](#bookmark265)presents the running time of Lloyd’s algorithm, and the speedup brought by INDE, SEQU, UniK, and UTune over Lloyd’s. The percentage of the pruned distance computations is shown below the speedup. In the appendix our of technical report [[68], we also show the correspond](#bookmark266)- ing assignment and renement time.

Observations. (1) On average, both UniK and UTune outperform the index-based and sequential methods in multiple cases, especially when k is big (see bold numbers for significant improvements), and the integration of bound and indexing further improves the pruning ratio. (2) UniK cannot always be fast over high-dimensional data such as Power, which is also consistent with the no free lunch theorem [[69]](#bookmark267). The main reason is that the index cannot prune well in the rst iteration for the datasets that do not assemble well, while UniK can alter to sequential method and avoid being slow in the following iterations. (3) By applying an auto-tuning in predicting the optimal conguration, UTune achieves the best performance across all datasets. The performance gap is even larger on low- dimensional datasets such as NYC (up to 389 times speedup), where UTune rightly predicts the conguration and improves the pruning percentage over SEQU.

To investigate whether our adopted ML models can generalize well to datasets that have not been seen during training, we further test UTune on three new datasets: Spam [[7], Shuttle](#bookmark268) [[8], and MSD](#bookmark269)

**Accuracy** BDT  **Basic features + Tree-features + Leaf-features**

DT RF SVM kNN RC DT RF SVM kNN RC DT RF SVM kNN RC

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bound@MRR | 0.41 | 0.70 | 0.68 | 0.62 | 0.63 | 0.57 | 0.67 | 0.70 | 0.64 | 0.63 | 0.59 | 0.69 | 0.68 | 0.63 | 0.63 | 0.60 |
| Index@MRR | 0.37 | 0.80 | 0.82 | 0.84 | 0.74 | 0.68 | 0.83 | 0.77 | 0.83 | 0.74 | 0.70 | 0.74 | 0.77 | 0.83 | 0.74 | 0.74 |
| S-Bound@MRR | 0.42 | 0.84 | 0.83 | 0.81 | 0.82 | 0.74 | 0.86 | 0.87 | 0.81 | 0.82 | 0.74 | 0.89 | 0.87 | 0.88 | 0.88 | 0.80 |
| S-Index@MRR | **0.43** | 0.89 | 0.87 | 0.86 | 0.83 | 0.83 | 0.91 | 0.90 | 0.87 | 0.83 | 0.85 | **0.92** | 0.92 | 0.92 | 0.86 | 0.84 |

**Table 6: Overall speedup over the running time (second) of Lloyd’s algorithm (the gray column) on various datasets.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data** | k = 10 k = 100 k = 1000 | | | | | | | | | | | | | | |
| Lloyd | ×Speedup | | | | Lloyd | ×Speedup | | | | Lloyd | ×Speedup | | | |
| SEQU | INDE | UniK | UTune | SEQU | INDE | UniK | UTune | SEQU | INDE | UniK | UTune |
| Cross | 262 | 1.64 71% | 1.76 67% | 1.36 78% | 1.76 84% | 1463 | 2.83 86% | 2.16 59% | **3.24**  90% | **4.70**  90% | 13530 | 3.31 90% | 1.81 46% | **4.04**  94% | **7.73**  93% |
| Conf | 2.45 | 1.32 68% | 1.30 61% | 1.30 74% | 1.32 68% | 9.00 | 1.53 90% | 1.63 25% | 2.22 90% | 2.50 90% | 50.75 | 2.83 90% | 1.47 6% | 2.88 93% | 2.93 93% |
| Covt | 0.65 | 1.89 74% | 2.18 72% | 1.68 87% | 2.18 72% | 2.53 | 5.62 90% | 1.43 23% | 5.61 94% | 5.67 93% | 10.39 | 7.47 92% | 1.04 4% | 6.66 92% | 7.47 92% |
| Euro | 15.3 | 1.38 75% | 1.42 67% | 1.39 84% | 1.48 35% | 111 | 3.24 92% | 2.53 45% | 3.79 90% | 4.03 95% | 381.9 | 2.65 94% | 0.63 11% | 3.12 95% | 3.13 95% |
| KeggD | 0.45 | 2.93 83% | 3.59 79% | **4.22**  84% | **4.3**  95% | 1.16 | 2.61 92% | 1.21 11% | 4.00 71% | 5.8 95% | 8.50 | 6.58 93% | 1.23 11% | **7.01**  89% | **7.57**  95% |
| Kegg | 0.49 | 1.98 78% | 2.83 83% | 2.40 94% | 2.83 83% | 2.49 | 4.69 93% | 1.79 31% | **5.87**  95% | **6.15**  96% | 18.64 | 6.67 93% | 0.94 51% | 6.52 95% | 6.67 93% |
| NYC | 15.3 | 1.39 84% | 389 99% | 31.4 99% | **389**  99% | 75.6 | 4.19 94% | 153 99% | 55.6 99% | 153 99% | 229.8 | 1.69 93% | 11.05 93% | 7.53 95% | 13.3 96% |
| Skin | 0.56 | 1.30 79% | 2.54 87% | 2.40 88% | 2.54 87% | 2.92 | 2.35 92% | 2.60 56% | **4.09**  96% | **4.13**  96% | 21.41 | 2.70 93% | 1.38 27% | **3.28**  94% | **3.54**  95% |
| Power | 6.38 | 1.43 78% | 0.77 53% | 0.87 82% | 1.43 78% | 32.9 | 2.39 91% | 1.02 18% | 2.53 93% | 2.60 91% | 223.9 | 2.17 92% | 0.96 2% | 2.26 92% | 2.5 92% |
| Road | 6.02 | 1.36 84% | 8.64 96% | 8.19 98% | 8.64 96% | 21.2 | 2.57 93% | 3.68 69% | **4.60**  93% | **4.93**  97% | 132.8 | 2.40 94% | 1.58 27% | 2.69 93% | 2.87 95% |
| Census | 11.9 | 1.31 62% | 0.82 26% | 1.14 67% | 1.55 69% | 94.7 | 3.65 84% | 1.14 15% | 3.51 85% | 3.67 84% | 791 | 5.85 91% | 1.05 9% | 5.87 91% | 5.87 91% |
| Mnist | 7.44 | 1.13 1% | 0.91 0% | 0.98 1% | 1.36 27% | 67.3 | 1.21 17% | 0.98 15% | 1.22 18% | 3.94 77% | 709 | 1.69 37% | 1.04 2% | 1.54 38% | 5.13 83% |
| Spam | 0.12 | 1.13 | 1.42 | 1.15 | 1.62 | 0.69 | 5.80 | 2.12 | 12.59 | 12.59 | 4.87 | 4.35 | 2.87 | 8.87 | 8.87 |
| Shuttle | 0.20 | 3.65 | 0.72 | 0.57 | 3.65 | 1.15 | 5.62 | 3.67 | 5.47 | 6.53 | 4.61 | 4.85 | 1.94 | 5.17 | 5.17 |
| MSD | 8.93 | 1.17 | 0.72 | 0.92 | 1.17 | 21.2 | 2.04 | 1.21 | 2.17 | 2.17 | 592 | 2.33 | 1.17 | 2.57 | 2.57 |

[[17]](#bookmark270). The results are consistent with our earlier nding, i.e., UTune beats other methods in most times, and is always in the leaderboard.

This conrms a good generalization capability of our ML model.

7.3.3 Summary of Lessons Learned. Through an evaluation on multiple learning models for auto-conguration, we further learn:

• Automatic algorithm selection for fast k-means is feasible by using a meta-learning model. Through UTune, we can predict an algorithm conguration that leads to better performance than state-of-the-art methods which we have reported in last section, also including our UniK. Moreover, the learning cost is low if using the oine evaluation logs.

• There are several ways to improve the precision of models. Firstly, our selective running based on evaluation can save much time to generate more logs for training. Secondly, building indexes can provide more features that help improve the prediction accuracy.

• By using very basic machine learning models without ne-tuning, our algorithmselection approach already achieves 92% prediction

accuracy. Note that nding the best choice of learning models is orthogonal to this work, though.

**8 CONCLUSIONS**

We evaluated existing accelerating algorithms for fast k-means, in a unied framework which enables a ne-grained performance break- down. To auto-congure the pipeline for the optimal performance, we trained a meta-model to predict the optimal conguration. Ex- periments on real datasets showed that our unied framework with autotuning can accelerate k-means efectively. In the future, we will work on learning with a rank-aware loss function like MRR, i.e., designing specic machine learning models to further improve precision. Recalling the parameter settings in Section [7.1,](#bookmark212) it will also be interesting to study the tuning of three sequential methods’ parameters. More discussions on these future opportunities can be found in our technical report [[68] (see Section](#bookmark271)[A.5)](#bookmark3).

**Acknowledgment.** Zhifeng Bao is supported by ARC DP200102611, DP180102050 and a Google Faculty Award.

**REFERENCES**

[1] BigCross Dataset. [https://s3.amazonaws.com/h2o-training/clustering/BigCross.](https://s3.amazonaws.com/h2o-training/clustering/BigCross.data.gz) [data.gz.](https://s3.amazonaws.com/h2o-training/clustering/BigCross.data.gz)

[2] ConfLongDemo-JSI | Machine Learning Data. [http://networkrepository.com/](http://networkrepository.com/ConfLongDemo-JSI.php) [ConfLongDemo-JSI.php.](http://networkrepository.com/ConfLongDemo-JSI.php)

[3] Europedif dataset. [http://cs.joensuu.f/sipu/datasets/europedi.txt.](http://cs.joensuu.fi/sipu/datasets/europediff.txt)

[4] K-Means clustering in Scikit-learn. [https://scikit-learn.org/stable/modules/](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) [generated/sklearn.cluster.KMeans.html.](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)

[5] NYC Taxi Limousine Commission - Trip Record Data). [http://www.nyc.gov/html/](http://www.nyc.gov/html/tlc/html/technology/data.shtml) [tlc/html/technology/data.shtml.](http://www.nyc.gov/html/tlc/html/technology/data.shtml)

[6] Repository of UniK. [https://github.com/tgbnhy/fast-kmeans.](https://github.com/tgbnhy/fast-kmeans)

[7] Spambase Data Set. [https://archive.ics.uci.edu/ml/datasets/spambase.](https://archive.ics.uci.edu/ml/datasets/spambase)

[8] Statlog (Shuttle) Data Set. [https://archive.ics.uci.edu/ml/datasets/Statlog+](https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)) [%28Shuttle%29.](https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle))

[9] THE MNIST DATABASE. [http://yann.lecun.com/exdb/mnist/.](http://yann.lecun.com/exdb/mnist/)

[10] UCI Machine Learning Repository: 3D Road Network (North Jutland, Denmark) Data Set. [https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+](https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+Jutland,+Denmark)) [Jutland,+Denmark).](https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+Jutland,+Denmark))

[11] UCI Machine Learning Repository: Covertype Data Set. [https://archive.ics.uci.](https://archive.ics.uci.edu/ml/datasets/covertype) [edu/ml/datasets/covertype.](https://archive.ics.uci.edu/ml/datasets/covertype)

[12] UCI Machine Learning Repository: Individual household electric power con- sumption Data Set.[https://archive.ics.uci.edu/ml/datasets/Individual+household+](https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption) [electric+power+consumption.](https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption)

[13] UCI Machine Learning Repository: KEGG Metabolic Reaction Network (Undi- rected) Data Set. [https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+](https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Reaction+Network+(Undirected)) [Reaction+Network+(Undirected).](https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Reaction+Network+(Undirected))

[14] UCI Machine Learning Repository: KEGG Metabolic Relation Network (Directed) Data Set. [https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Relation+](https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Relation+Network+(Directed)) [Network+(Directed).](https://archive.ics.uci.edu/ml/datasets/KEGG+Metabolic+Relation+Network+(Directed))

[15] UCI Machine Learning Repository: Skin Segmentation Data Set. [https://archive.](https://archive.ics.uci.edu/ml/datasets/skin+segmentation) [ics.uci.edu/ml/datasets/skin+segmentation.](https://archive.ics.uci.edu/ml/datasets/skin+segmentation)

[16] UCI Machine Learning Repository: US Census Data (1990) Data Set. [https:](https://archive.ics.uci.edu/ml/machine-learning-databases/census1990-mld/) [//archive.ics.uci.edu/ml/machine-learning-databases/census1990-mld/.](https://archive.ics.uci.edu/ml/machine-learning-databases/census1990-mld/)

[17] YearPredictionMSD Data Set. [https://archive.ics.uci.edu/ml/datasets/](https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD) [YearPredictionMSD.](https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD)

[18] Z. Abedjan, L. Golab, and F. Naumann. Proling relational data: a survey. VLDB Journal, 24(4):557–581, 2015.

[19] A. Aggarwal, A. Deshpande, and R. Kannan. Adaptive Sampling for k-Means Clustering. In APPROX and RANDOM, pages 15–28, 2009.

[20] D. Arthur and S. Vassilvitskii. K-Means++: The advantages of careful seeding. In SODA, pages 1027–1025, 2007.

[21] K. Bache and M. Lichman. UCI Machine Learning Repository, 2013.

[22] B. Bahmani,R. Kumar, and S. Vassilvitskii. Scalable K-Means++. PVLDB, 5(7):622– 633, 2012.

[23] M. A. Bender, J. Berry, S. D. Hammond, B. Moore, B. Moseley, and C. A. Phillips.

k-Means Clustering on Two-Level Memory Systems. In MEMSYS, pages 197–205, 2015.

[24] J. L. Bentley. Multidimensional Binary SearchTrees Used for Associative Search- ing. Commun. ACM, 18(9):509–517, 1975.

[25] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In ICML, pages 97–104, 2006.

[26] T. Bottesch, T. Bühler, and M. Kächele. Speeding up k -means by approximating Euclidean distances via block vectors. In ICML, pages 2578–2586, 2016.

[27] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, and S. Venkatesan. Scalable K-Means by ranked retrieval. In WSDM, pages 233–242, 2014.

[28] M. E. Celebi, H. A. Kingravi, and P. A. Vela. A comparative study of effcient initialization methods for the k-means clustering algorithm. Expert Systems With Applications, 40:200–210, 2013.

[29] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in Metric Spaces. ACM Computing Surveys, 33(3):273–321, 2001.

[30] B. Chazelle. An Optimal Convex Hull Algorithm in Any Fixed Dimension. Technical report, 1993.

[31] L. Chen, Y. Gao, B. Zheng, C. S. Jensen, H. Yang, and K. Yang. Pivot-based metric indexing. PVLDB, 10(10):1058–1069, 2017.

[32] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An ecient access method for similarity search in metric spaces. In VLDB, pages 426–435, 1997.

[33] N. Craswell. Mean Reciprocal Rank. Encyclopedia of database systems, 2009.

[34] R. R. Curtin. A Dual-Tree Algorithm for Fast k-means Clustering With Large k. In SDM, pages 300–308, 2017.

[35] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz. Yinyang K-means: A drop-in replacement of the classic K-means with consistent speedup. In ICML, pages 579–587, 2015.

[36] J. Drake. Faster k-means Clustering. In MS Thesis, 2013.

[37] J. Drake and G. Hamerly. Accelerated k-means with adaptive distance bounds. In OPT, pages 2–5, 2012.

[38] C. Elkan. Using the triangle inequality to accelerate k-means. In ICML, pages 147–153, 2003.

[39] K. Fukunaga and P. M. Narendra. A Branch and Bound Algorithm for Computing k-Nearest Neighbors. IEEE Trans. Computers, 24(7):750–753, 1975.

[40] G. Hamerly. Making k-means even faster. In SDM, pages 130–140, 2010.

[41] G. Hamerly and J. Drake. Accelerating Lloyd’s Algorithm for k-Means Clustering. 2015.

[42] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. PVLDB, 2(1):1282–1293, 2009.

[43] A. K. Jain. Data clustering: 50 years beyond K-means. Pattern Recognition Letters, 31(8):651–666, 2010.

[44] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An ecient k-means clustering algorithm: Analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(7):881–892, 2002.

[45] S. B. Kotsiantis. Supervised Machine Learning: A Review of Classication Tech- niques. Informatica, 31:249–268, 2007.

[46] W. Kwedlo. Two Modications of Yinyang K-means Algorithm. In ICAISC, pages 94–103, 2017.

[47] G. Li, X. Zhou, S. Li, and B. Gao. QTune: A Query-Aware database tuning system with deep reinforcement learning. PVLDB, 12(12):2118–2130, 2019.

[48] S. P. Lloyd. Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2):129–137, 1982.

[49] A. Lulli, M. Dell’Amico, P. Michiardi, and L. Ricci. Ng-Dbscan: Scalable Density- Based Clustering for Arbitrary Data. PVLDB, 10(3):157–168, 2016.

[50] A. W. Moore. The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In UAI, pages 397–405, 2000.

[51] E. Müller, S. Günnemann,I. Assent, and T. Seidl. Evaluating clustering in subspace projections of high dimensional data. PVLDB, 2(1):1270–1281, 2009.

[52] M. A. Muñoz, Y. Sun, M. Kirley, and S. K. Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. Information Sciences, 317:224–245, 2015.

[53] J. Newling and F. Fleuret. Fast k-means with accurate bounds. In ICML, pages 936–944, 2016.

[54] J. Newling and F. Fleuret. Nested Mini-Batch K-Means. In NIPS, pages 1352–1360, 2016.

[55] J. Newling and F. Fleuret. K-Medoids For K-Means Seeding. In NIPS, pages 5201–5209, 2017.

[56] S. M. Omohundro. Five Balltree Construction Algorithms. Technical report, 1989.

[57] F. Pedregosa, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[58] D. Pelleg and A. Moore. Accelerating exact k -means algorithms with geometric reasoning. In KDD, pages 277–281, 1999.

[59] S. J. Phillips. Acceleration of k-means and related clustering algorithms. In ALENEX, pages 166–177, 2002.

[60] A. Rivolli, L. P. F. Garcia, C. Soares,J. Vanschoren, and A. C. P. L. F. de Carvalho. Towards Reproducible Empirical Research in Meta-learning. Technical report, 2018.

[61] P. Ryšavý and G. Hamerly. Geometric methods to accelerate k -means algorithms. In SDM, pages 324–332, 2016.

[62] H. Samet. Foundations of multidimensional and metric data structures. Morgan Kaufmann, 2006.

[63] M. Shindler, A. Wong, and A. Meyerson. Fast and Accurate k-means For Large Datasets. In NIPS, pages 2375–2383, 2011.

[64] J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. Information Processing Letters, 40(4):175–179, 1991.

[65] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic Database Manage-

ment System Tuning Through Large-scale Machine Learning. In SIGMOD, pages 1009–1024, 2017.

[66] J. Vanschoren. Meta-Learning: A Survey. Technical report, 2018.

[67] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin. Fast large-scale trajectory clustering. PVLDB, 13(1):29–42, 2019.

[68] S. Wang, Y. Sun, and Z. Bao. On the effciency of k-means clustering: Evaluation, optimization, and algorithm selection (technical report). arXiv preprint arXiv: 2010.06654, 2020.

[69] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1):67–82, 1997.

[70] X. Wu, V. Kumar, J. R. Quinlan,J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan,

A. Ng, B. Liu,P. S. Yu, Z.-H. Zhou,M. Steinbach, D.J. Hand, and D. Steinberg. Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1):1–37, 2008.

[71] S. Xia, D. Peng, D. Meng, C. Zhang, G. Wang, E. Giem, W. Wei, and Z. Chen. A Fast Adaptive k-means with No Bounds. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 1–1, 2020.

[72] M. Zechner and M. Granitzer. Accelerating K-Means on the Graphics Processor via CUDA. In INTENSIVE, pages 7–15, 2009.

[73] W. Zhao, H. Ma, and Q. He. Parallel K -Means Clustering Based on MapReduce. In CloudCom, pages 674–679, 2009.