# S-FINCH: An Optimized Streaming Adaptation to FINCH Clustering

James Cunningham*, Jim Davis*, Kyle Tarplee† and Juan Vasquez‡
*Department of Computer Science and Engineering,
Ohio State University, Columbus, OH 43210
Email: {cunningham.844, davis.1719}@osu.edu
†University of Dayton Research Institute, Dayton, OH 45469
Email: Kyle.Tarplee@udri.udayton.edu
‡Air Force Research Laboratory, Wright-Patterson AFB, OH 45433
Email: juan.vasquez.11@afresearchlab.com

***Abstract*—Real-world datasets are growing ever larger and more dynamic, motivating the need for efficient online data exploration algorithms. In this work, we present S-FINCH, a streaming domain optimization of the recent FINCH clustering method. The original FINCH approach demonstrated state-of-the-art offline performance while avoiding sensitive hyperparameters but was not designed for the online data streaming domain. The proposed S-FINCH method is an exact, incremental update to the FINCH approach which generates high-quality clusters while avoiding the potential sensitivities of other online clustering methods. We also provide alternative cluster tree representatives for faster empirical cluster times. Experiments are shown comparing the original FINCH approach to the proposed S-FINCH method in a streaming domain with multiple benchmark synthetic and real datasets. The S-FINCH method leverages the ability to make local changes for efficient, real time updates and can be applied to multiple data streaming scenarios.**

## I. Introduction

As the number of data sources and dynamic datasets grows, so does the need for efficient online algorithms to process them. Clustering is a very important data exploration tool used in various fields from data science to machine learning. Online clustering is the process of grouping samples in a data stream into meaningful collections as they appear over time. Unfortunately, available online clustering methods suffer from a sensitivity towards required data abstraction methods, various hyperparameters (i.e. number of clusters, density parameters, neighborhood sizes), outliers within the dataset, and poor scaling towards large numbers of clusters [1], [2].

The recent FINCH algorithm [3] presents an offline clustering algorithm which avoids sensitive hyperparameters, building a cluster graph using first-neighbor relationships as edges between samples and defining clusters as connected graph components. The FINCH algorithm aggregates these clusters to build a bottom-up cluster hierarchy quickly creating clusters of high purity with no prior knowledge of the data required. However, the FINCH algorithm is by design an *offline* clustering algorithm. It is not designed to efficiently handle the introduction of new data and thus requires a full re-clustering of all the data samples when even one new sample enters the space. As datasets become more dynamic with continuous streaming updates, it is important to consider how to efficiently update the cluster space over time to reflect the new data.

In this work we present an online modification to FINCH which we call Stream-FINCH (S-FINCH). We initially examine the stages of the FINCH algorithm and leverage key insights to produce an algorithm which reduces the online update complexity. We then compare the performance of S-FINCH and FINCH over multiple synthetic and real-world datasets. We show theoretically and empirically that the proposed S-FINCH algorithm is more efficient than the FINCH algorithm in the online domain and has reasonable real-time performance. We additionally present several different cluster representatives which can be used to efficiently construct the agglomerative cluster hierarchies used in FINCH. Lastly, we compare the cluster quality and clustering time performance using these new representatives.

## II. Related Work

Offline or batch clustering algorithms assume all relevant data is present at clustering time and can be processed at once. These algorithms are very well studied and employ a variety of strategies including partitioning, hierarchies, density models, subspace projection and graph theory [4], [5], [6], [7], [8], [9]. The most common among these strategies are partitioning and hierarchy-based methods.

Partitioning methods such as k-means or k-medoids predefine the number of desired partitions to efficiently cluster data, but suffers greatly from a strong sensitivity to outliers, the number of preset clusters, and the initial position of cluster representatives, as well as easily being drawn to local optima and having poor performance on non-convex data [2]. Hierarchical clustering methods aim to develop a hierarchical relationship among the data which is leveraged for clustering [6], [7], [10]. These methods are interpretable, scalable, and suitable for arbitrary data types, but can have high time complexities and are still sensitive to the preset number of clusters [2].

Unlike offline clustering, online algorithms assume non-stationary, unbounded data streams in which objects arrive continuously and in any order [1]. The nature of these data streams imposes more stringent requirements on time and

*International Conference on Pattern Recognition (ICPR), August 21-25, 2022*

memory complexity. For this reason, online clustering follows a two-phase approach: an *online* data abstraction phase which summarizes incoming blocks of data followed by an *offline* clustering phase which clusters sets of abstractions at defined intervals [1]. This paradigm improves online performance by reducing the number of objects through abstraction and moving the most inefficient phase, clustering, to an offline process. However, this approach introduces several inherent weaknesses that must be accounted for within both the data abstraction and clustering phases.

The data abstraction phase introduces sensitivity to both the specific data abstraction approach and approach-specific parameters such as window size and statistics about the data. Abstraction also introduces information loss and assumes a relationship among summarized objects which may not exist. Notably, the clustering phase in this paradigm is simply *offline* clustering on the abstractions. Therefore the online clustering algorithm inherits the weaknesses of the chosen offline clustering method, most often k-means [10], [11], [12], [13], [14], [15] or density-based clustering methods such as DBSCAN [8], [16], [17], [18] (chosen for their low time complexity given a preset number of clusters).

Recent work has also introduced "deep clustering" as a method of identifying partitions in datasets. Deep clustering uses trained deep neural networks to generate easily clusterable data representations or directly cluster input data [19], [20], [21], [22]. These methods can leverage deep learning frameworks to quickly process input data streams at speeds that can allow online clustering, but can be difficult to interpret, require large amounts of training resources, and struggle to handle data outside of the training distribution [23].

Our proposed S-FINCH algorithm addresses these issues by extending FINCH, a state-of-the-art hierarchical *offline* clustering method to the *online* domain. The high time complexity of hierarchical methods is avoided in S-FINCH by leveraging information in the existing hierarchy to perform efficient cluster updates when given new data. This method is a single phase, purely online approach that **does not** require data abstraction, a predefined number of clusters, or training and retains the scalability and flexibility of other hierarchical methods, making it suitable for data exploration with no prior assumptions of the dataset.

## III. Original FINCH Algorithm

The FINCH algorithm [3] provides a method of clustering by operating on the input dataset in three main stages: generate a graph from the dataset, label the connected components of the graph as clusters, and build a cluster hierarchy.

The first stage of FINCH generates a cluster graph using an adjacency matrix $A$ that computed using the following clustering equation from [3]:

$$A(i,j) = \begin{cases} 1 & \text{if } k_i^1 = j \text{ or } k_j^1 = i \text{ or } k_i^1 = k_j^1 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $k_i^1$ is the first nearest neighbor of sample $i$. This graph directly represents the first and shared neighbor relationships between samples in the input space. In the worst case this operation takes $\Theta(n^2)$ time for a true (non-approximate) nearest neighbor search. The second stage of FINCH labels the connected components produced in Stage 1 as clusters. In the worst case, all $n$ samples are connected via $n^2$ edges so this operation requires a $O(n^2)$ graph traversal.

FINCH then constructs a hierarchy of cluster partitions to avoid small, fractured clusters. Each level in this hierarchy is constructed by applying Stages 1 and 2 to the cluster means of the previous level, adding levels until the clusters converge. Since the minimum cluster size in FINCH is 2, the largest possible hierarchy is a full, balanced binary tree of height $log(n)$. This means the worst case time complexity of the FINCH algorithm can be expressed as:

$$(\Theta(n^2) + O(n^2)) \cdot O(log(n)) = O(n^2 log(n)) \quad (2)$$

The resulting hierarchy represents a fully unsupervised and parameter-free exploration of the input samples with each level representing a different set of possible cluster partitions. The levels can be examined and the most appropriate partition chosen based on the problem domain to give a final output clustering of the input samples.

However, since FINCH is an *offline* batch clustering algorithm, a naïve approach to online clustering requires a full re-clustering of the samples when any new data is introduced. Therefore the worst-case time complexity of processing $n$ samples from a continuous data stream using FINCH is

$$O(n \cdot O(n^2 log(n))) = O(n^3 log(n)) \quad (3)$$

We use this analysis of FINCH to motivate an online adaptation which reduces the single iteration update complexity.

## IV. Proposed S-FINCH Algorithm

We present Stream-FINCH (S-FINCH), an efficient online clustering variant of FINCH. S-FINCH leverages several key insights into the FINCH cluster graph that reveal the potential for efficient local updates to the cluster partitions given a new sample. The resulting cluster partitions will always be identical to the FINCH clusters produced using the same set of data points, regardless of the order in which points are processed.

We initially note that since shared nearest neighbor edges connect samples which share a nearest neighbor, removing them from the cluster graph *never* changes the graph's connected components. The samples in question are already connected via their nearest neighbor. For this reason we do not consider or compute the shared nearest neighbor edges for the S-FINCH algorithm and only keep the first neighbor edges. This reduces the worst-case number of edges in the cluster graph from $n^2$ to $n$.

We also note that cluster updates are necessarily local to the new sample and that each connected component in the cluster graph contains *exactly* one cycle. Storing cycle information allows for efficient clustering behavior when a new sample is introduced. Additionally, the FINCH algorithm builds successive hierarchy levels using the cluster means of the previous level. Cluster means will necessarily change

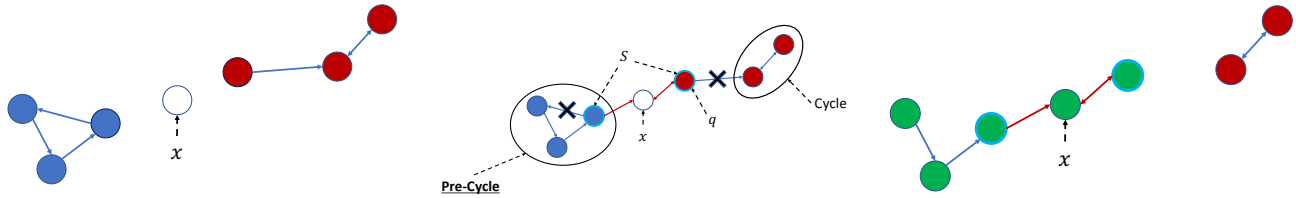


Fig. 1. S-FINCH Sample Relations. (a) Input cluster graph. (b) S-FINCH Stage 1: Cluster Graph Update. (c) S-FINCH Stage 2: Cluster Labeling.

during cluster updates, forcing the entire cluster hierarchy to be rebuilt. However, other cluster representatives can be employed to avoid this behavior and generate different cluster partitions.

### A. Proposed Method

The S-FINCH algorithm operates in three stages similar to the original FINCH algorithm (as presented in Algorithm 1). The first stage efficiently updates a given cluster graph given a new sample $x$ while finding both $x$'s nearest neighbor $q$, and the set of updated samples $S$. The second stage updates the cluster labels by performing a *local* search of the cluster graph from the new sample $x$. Finally, the third stage updates the cluster hierarchy given the new bottom-level clusters.

**Algorithm 1:** S-FINCH()

**Data:** $\mathcal{L}$: List of partitions, $x$: new sample
**begin**
  $C \longleftarrow$ first group of partitions in $\mathcal{L}$
  $U \longleftarrow \emptyset$ `// set of updated clusters`
  **while** $|C| > 1$ *or* $|U| > 0$ **do**
    `// Stage 1`
    Update edges in $C$, finding nearest neighbor $q$ of $x$ and set of samples $S$ linked to $x$
    `// Stage 2`
    Identify cluster label $x_L$ from $q$
    Label $x$'s cluster and find cluster's cycle
    `// Stage 3`
    $R \longleftarrow$ cluster representatives
    $C \longleftarrow$ next group of partitions in $\mathcal{L}$
    $U \longleftarrow R \setminus C$

The first stage of S-FINCH updates a given FINCH cluster graph from time $t-1$ with the new sample $x$ in a single pass through the dataset, finding $x$'s nearest neighbor $q$ and the set $S$ of any samples which have $x$ as their new nearest neighbor (as shown in Fig. 1) in $\Theta(n)$ time.

In the FINCH cluster graph, clusters are connected components. When a new sample is introduced, each cluster either grows by gaining edges, splits by losing edges, or remains the same. Since each edge represents a nearest neighbor relationship, all samples with changed edges are members of $S$ with the only other new edge belonging to $x$. Therefore, given $q$ and $S$, we can efficiently determine how the clusters of the FINCH cluster graph have changed post-update and find $x_L$, the cluster label of $x$, in **constant time**.

If $q$ is *not* a member of $S$ then it has not been updated, so $x_L = q_L$. If $q$ instead *is* a member of $S$, then it is possible $q$'s cluster has split. In this case, if $q$ was in a *cycle* prior to the update, then $q$'s cluster didn't split and $x_L = q_L$. Otherwise, $q$'s cluster has split and now points to $x$ so a new $x_L$ is assigned.

Once $x_L$ is determined, $x$'s cluster is labeled using a $O(n)$ depth-first search on the undirected cluster graph from the members of $S$ (as shown in Fig. 1). Since efficient cluster labeling requires cycle information, once the cluster is labeled, its cycle is found using another $O(n)$ depth-first search from $x$. With this, the S-FINCH cluster graph is updated and each connected component is assigned a unique cluster label. These new clusters can now be agglomerated to form a hierarchical cluster tree.

Cluster agglomeration in S-FINCH is identical to the original FINCH algorithm, where the partition at a hierarchy level is computed from the set of cluster representatives from the previous level. This means the largest hierarchy has height $log(n)$. Given the complexity of clustering a level in S-FINCH, the total worst-case time complexity of S-FINCH can be expressed as

$$O((\Theta(n) + O(n)) \cdot log(n)) = O(nlog(n)) \tag{4}$$

This results in an overall reduction in the update time complexity of the S-FINCH algorithm compared to the original FINCH algorithm (Eqn. 2).

Furthermore, there is a reduction in overall *online* time complexity in S-FINCH when clustering $n$ samples from a continuous data stream:

$$O(n \cdot O(nlog(n))) = O(n^2 log(n)) \tag{5}$$

as compared to FINCH (Eqn. 3).

We note that for the online clustering domain, it is only necessary to update levels where a cluster representative has changed. "Virtual" representatives such as the cluster mean are not actual samples and therefore always change when a cluster is updated, changing the entire hierarchy. Using an *actual sample* as a prototype instead gives a discrete set of possible representatives, making it possible that no cluster representatives change after an update. This can significantly reduce necessary operations during agglomeration.

We offer four different cluster representatives. A **Mean** virtual representative is the mean of the original input samples included in the cluster partition as used in the original FINCH algorithm. A **Mean-of-Means** virtual representative is a mean calculated using the mean representatives of the previous level in the hierarchy for the current cluster partition. Representatives **Prototype 1** and **Prototype 2** are the actual samples in the cluster partition closest to the **Mean** and **Mean-of-Means**, respectively. We will compare each of these representatives empirically in our experiments.

## V. Experiments

We present our experiments in two phases, first examining the initial bottom level clustering results then observing the effect of different cluster representatives on cluster agglomeration and clustering performance.

### A. Datasets

We conduct our experiments using the following datasets used in the original FINCH experiments. The Aggregation dataset [24] is a 2-D point clustering dataset of 788 samples grouped into 7 clusters. Similarly, we use the Gestalt [25] cluster dataset of 399 2-D samples over 6 ground truth clusters[1]. For our larger-scale datasets, we first employ the MNIST handwritten digit 10K test set, using rasterized 784-D vectors of the pixels. Finally we examine the CIFAR-10 natural image dataset using 512-D feature vectors of the 10K test set. These feature vectors were extracted just before the fully connected layer from a modified ResNet-18 [26] CNN (first convolutional layer replaced by a 3x3 convolution of stride 1 with no max pooling).

### B. FINCH*

For additional comparison, we aim to provide a simple baseline optimization for FINCH. One option is to simply assign incoming samples to the nearest cluster partition on the first level of the hierarchy before updating the entire hierarchy. However, this approach does not allow for cluster separation and produces different clusters from FINCH.

Instead, we note that a simple streaming domain optimization to the FINCH algorithm can be used that does not recompute the *entire* pairwise distance matrix when a new sample is introduced. Instead, only the pairwise distance between the new sample and existing samples is computed. This optimization reduces the complexity of FINCH Stage 1 to $\Theta(n)$. However, this does not change the overall upper bound given in Eqn. 2. We refer to this variant as FINCH*. The update and online overall final time complexities of FINCH* are identical to FINCH.

[1]Note that the 2-D datasets include samples with more than two samples equidistant from each other. The original FINCH algorithm [3] does not include any details regarding choosing a nearest neighbor from a set of equidistant options. Therefore, we added an extremely small amount of Gaussian noise to the datasets so no sample is equidistant to more than one other sample.

### C. Bottom-Level Clustering Comparisons

We compare several metrics of the bottom-level cluster output from which the theoretical complexities were derived and relate these results to relative running times for FINCH, FINCH*, and S-FINCH.

**Cluster Graph Statistics:** The results in Table I suggest the bottom-level cluster output for these datasets consists of many small, fragmented clusters. These small clusters are actually more desirable for S-FINCH than FINCH since the number of clustering operations required for S-FINCH Stage 2 increases *linearly* with cluster size (FINCH is quadratic). Table I shows that cluster size remains small and nearly constant with respect to dataset size. These empirical observations of FINCH clustering behavior should translate into a small, nearly constant number of required operations for S-FINCH Stage 2, suggesting S-FINCH bottom-level clustering times far below the theoretical worst-case.

**Cluster Update Times:** We see from Tables II and III that our empirical relative cluster time (results are computed and timed on a standard laptop) matches the theoretical expectations derived in the previous sections. Both the FINCH and FINCH* algorithms fail to be feasible online clustering options for the larger MNIST and CIFAR-10 datasets, taking over 3 hours to process less than 50% of each dataset with the remaining running time expected to increase quadratically. In contrast, the results of the S-FINCH algorithm in Tables II and III show an order of magnitude decrease in cluster time for both the final sample update and the clustering of the entire dataset stream.

**S-FINCH Scaling:** Here we examine the behavior of S-FINCH as the dataset size scales. For this we created a dataset of increasing size using random 2-D sample vectors drawn from a uniform distribution. The results in Table IV show that the update time and the distance calculation time increase linearly with input size while cluster labeling time stays relatively constant over increasing input sizes. This suggests that any further improvement to S-FINCH Stage 1 will dramatically reduce clustering times.

### D. Cluster Agglomeration

In this section we examine the effect of the cluster representative choice used in the hierarchical updates.

**Stopped Level:** Table V compares the average level in the bottom-up hierarchy at which computation stopped (lower is better) vs. the height of the cluster hierarchy. These results empirically validate that the virtual representatives (Mean, Mean-of-Means (MoM)) will always require every level in the tree to be re-clustered during an update and that the actual prototypes on average stopped within the first few levels of the hierarchy. We note that Prototype 2 typically stopped earlier than Prototype 1 across the datasets. This is likely due to the Mean-of-Means cluster representation (used in Prototype 2) being less sensitive than the Mean (used in Prototype 1) to changes in the cluster it represents.

**Agglomeration Time:** We report the relative average cluster agglomeration time in Table VI for the different cluster representatives across our datasets. The results in Table VI follow

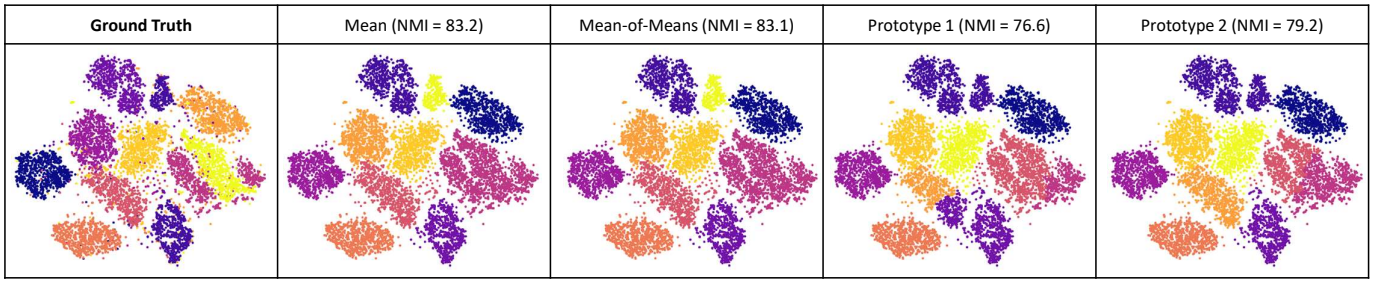


Fig. 2. CIFAR-10 Refined Clusters on 2-D t-SNE Reduction.

| | Aggr. | | | Gestalt | | | MNIST | | | CIFAR-10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Clusters** | 243 | | | 91 | | | 1951 | | | 1628 | | |
| **Cluster Size** | 3.24 | 2 | 7 | 4.39 | 2 | 28 | 5.13 | 2 | 39 | 6.14 | 2 | 52 |
| **Cycle Size** | 2.00 | 2 | 2 | 2.00 | 2 | 2 | 2.00 | 2 | 2 | 2.00 | 2 | 2 |

TABLE I
BOTTOM-LEVEL CLUSTER METRICS (MEAN/MODE/MAX). CLUSTERS: NUMBER OF CLUSTERS. CLUSTER SIZE: NUMBER OF SAMPLES PER CLUSTER. CYCLE SIZE: NUMBER OF SAMPLES IN A CLUSTER CYCLE.

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **FINCH** | 0.015 | 5.9e-3 | 3.14 | 2.26 |
| **FINCH*** | 0.67x | 1x | 0.61x | 0.38x |
| **S-FINCH** | **0.01x** | **0.02x** | **0.02x** | **0.02x** |

TABLE II
FINAL SAMPLE RELATIVE ONLINE UPDATE TIME (SEC).

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **FINCH** | 5.31 | 1.44 | $>$ 3 hours | $>$ 3 hours |
| **FINCH*** | 0.87x | 0.92x | $>$ 3 hours | $>$ 3 hours |
| **S-FINCH** | **0.11x** | **0.13x** | **508.75** | **180.27** |

TABLE III
ALL SAMPLES RELATIVE ONLINE UPDATE TIME (SEC).

| Size | Update Time | Distance | Labeling |
|---|---|---|---|
| **10000** | 3.2e-4 | 2.2e-4 | 2.0e-6 |
| **20000** | 1.66x | 1.86x | 0.5x |
| **50000** | 4.69x | 5x | 1.5x |
| **80000** | 7.19x | 7.73x | 2.0x |
| **100000** | 8.75x | 9.55x | 2.0x |

TABLE IV
S-FINCH RELATIVE UPDATE TIMES OVER INCREASING $n$ (SEC).

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **Mean** | 7 / 7 | 6 / 6 | 7 / 7 | 7 / 7 |
| **MoM** | 6 / 6 | 6 / 6 | 8 / 8 | 7 / 7 |
| **Proto. 1** | 2.3 / 7 | 2.3 / 6 | 1.7 / 7 | 1.7 / 7 |
| **Proto. 2** | **1.8** / 7 | **1.6** / 6 | **1.3** / 8 | **1.3** / 7 |

TABLE V
AVERAGE STOPPED LEVEL OF AGGLOMERATION VS. TREE HEIGHT.

the expectations from Table V, showing that actual prototypes result in lower computational cost. Stopping earlier in the hierarchy results in fewer operations during agglomeration and shorter agglomeration times for our prototype representatives. Among the prototype representatives, use of Prototype 2 was faster than Prototype 1 across the datasets.

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **FINCH** | 1.1e-2 | 6.0e-3 | 0.49 | 0.46 |
| **Mean** | 0.38x | 0.3x | 0.24x | 0.15x |
| **MoM** | 0.33x | 0.28x | 0.24x | 0.15x |
| **Proto. 1** | 0.21x | 0.18x | 0.06x | 0.04x |
| **Proto. 2** | **0.10x** | **0.09x** | **0.03x** | **0.02x** |

TABLE VI
AVERAGE RELATIVE AGGLOMERATION TIME AFTER FINAL SAMPLE UPDATE (SEC).

### *E. Clustering Quality*

Many evaluation methods have been proposed to measure cluster quality [27], [28], [29], [30], [31]. We choose to follow [3] and quantitatively measure cluster quality using the Normalized Mutual Information (NMI) [32] between the clustering algorithm output and ground truth (larger values are desired). We supplement these results with a qualitative visual representation of the cluster output using different cluster representatives.

We compare clustering quality using different cluster representatives by calculating the NMI between the *best* cluster partition in the cluster hierarchy and the given dataset's ground truth. The best cluster partition was chosen as the level in the cluster hierarchy with a number of clusters closest to the actual number of ground truth classes. From the NMI results in Table VII, the clustering quality of the different cluster representatives is similar within each dataset except for MNIST (to be discussed). We note the Mean representative results do not match [3] as their results were generated using an "early exit" strategy in their code not mentioned in their discussion of the algorithm. Our results are consistent with the algorithm presented in [3].

In [3] they also present a simple method for *refining* a cluster partition to a given number of clusters. First, the level in the hierarchy with the smallest number of clusters *greater* than the target number of ground truth classes is chosen. The closest clusters in this level are iteratively merged until the target

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **Mean** | 81.5 | 68.6 | 58.8 | **64.8** |
| **MoM** | 81.2 | **69.6** | **67.0** | 64.5 |
| **Proto. 1** | 77.6 | 66.3 | 40.2 | 60.4 |
| **Proto. 2** | **81.7** | 66.0 | 33.8 | 62.9 |

TABLE VII
NMI AT GROUND TRUTH.

number of clusters is reached. We report the corresponding refined NMI scores in Table VIII.

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **Mean** | 85.5 | 68.6 | 70.0 | **64.8** |
| **Mean-of-Means** | **98.7** | **69.6** | **70.5** | 64.5 |
| **Prototype 1** | 80.6 | 66.0 | 45.7 | 62.2 |
| **Prototype 2** | 86.9 | 66.4 | 50.7 | 62.8 |

TABLE VIII
REFINED NMI AT GROUND TRUTH.

### *F. Sparse Data*

The previous experiments demonstrated a severe degradation in NMI when using the actual cluster representatives for the MNIST dataset. This suggests that a limitation exists in using these prototypes as cluster representatives. The MNIST dataset consists of 28x28 nearly binary images which were rasterized into vectors for clustering. We note that these vectors are very sparse and therefore mostly exist on the corners a 784-D unit hypercube. Thus the mean of multiple FINCH clusters will not be near *any* of the actual samples, and therefore can explain the performance gap for MNIST.

We demonstrate this sparsity effect by instead using a *dense* embedding of the MNIST dataset obtained using features from the penultimate layer of a simple 2-layer CNN trained on MNIST to 98.7% test accuracy. We ran S-FINCH clustering on these 128-D MNIST CNN feature vectors across all cluster representatives and report the new NMI scores in Table IX. The Prototype NMI scores of the CNN features in Table IX are more consistent with the results in Tables VII for the other datasets.

| | Rasterized Pixels | | CNN Features | |
|---|---|---|---|---|
| | Raw | Refined | Raw | Refined |
| **Mean** | 58.8 | 70.0 | 94.3 | 94.3 |
| **MoM** | 67.0 | 70.5 | **94.5** | **94.5** |
| **Proto. 1** | 40.2 | 45.7 | 85.1 | 92.3 |
| **Proto. 2** | 33.8 | 50.7 | 88.5 | 88.5 |

TABLE IX
RAW VS. REFINED NMI OVER MNIST FEATURE REPRESENTATIONS.

### *G. Comparison to Baseline Clustering Algorithms*

We also compare these refined S-FINCH NMI scores (using Mean-of-Means cluster representatives) in Table X to several baseline clustering methods used for the offline clustering phase of many online clustering algorithms [3], [33], [34], [35], [9], [36], [10]. Note that the reported values for FINCH differ from [3] as we do not consider scores obtained using any additional early stoppage techniques (as was used in [3]).

Unlike S-FINCH's single phase approach, all major online clustering algorithms operate in a two-stage approach of online data abstraction clustered using an offline clustering backbone. As S-FINCH avoids the data abstraction step, we only compare cluster quality between S-FINCH and the offline clustering backbones using raw samples with no data abstraction. We expect these backbones to produce better clusters using no data abstraction as this is their intended offline domain. The results in Table X demonstrate that S-FINCH largely outperforms all other baselines on the Aggregation dataset and MNIST CNN dataset while having comparable performance on the Gestalt and CIFAR-10 datasets.

| | Aggr. | Gestalt | MNIST | CIFAR-10 |
|---|---|---|---|---|
| **S-FINCH (MoM)** | 98.7 | 69.6 | 94.5 | 64.5 |
| **FINCH** | -13.2 | -1.0 | -0.2 | -0.3 |
| **K-means** | -11.0 | 1.9 | -6.5 | 5.1 |
| **Spectral** | -17.7 | 5.4 | -92.6 | -0.3 |
| **HAC** | -6.7 | 3.7 | -2.6 | 2.2 |
| **SSC** | -24.7 | -2.6 | -12.2 | -72.0 |
| **EN-SSC** | -54.2 | -27.6 | -8.0 | 0.4 |
| **BIRCH** | -14.0 | 12.8 | -2.6 | 2.3 |

TABLE X
RELATIVE BASELINE COMPARISON (NMI).

We supplement the quantitative cluster quality results given by NMI scores with a qualitative visual clustering in Fig. 2 of CIFAR-10 using the different cluster representatives. We first reduced the CIFAR-10 dataset to 2-D using t-SNE [37] and then colored the data using the refined clustering results. We note that qualitatively all of the cluster representatives are comparable to ground truth, each finding visually separable groups within the dataset.

## VI. CONCLUSION

We presented S-FINCH, an online streaming domain optimization to the offline FINCH algorithm. We showed that S-FINCH provides theoretically and empirically more efficient online updates than the original FINCH algorithm, producing exact bottom-level cluster output that is provably identical to the original FINCH algorithm's cluster results. We also introduced alternative cluster representatives for agglomeration that can reduce the necessary agglomeration computation. The current bottleneck of S-FINCH is the search for updated vertices in the cluster graph and is recommended for future work. We also suggest further experimentation of different cluster representatives that can assist with Stage 1 search and/or increase cluster quality. As data sources grow even larger and become more numerous, we expect efficient online clustering algorithms such as S-FINCH to become key tools for data science. Our Python implementation of S-FINCH is available at https://github.com/jamie-cunningham/sfinch.

## VII. ACKNOWLEDGMENT

This research was supported by the US Air Force Research Laboratory under contract #GRT00054740.

## REFERENCES


[1] J. Silva, E. Faria, R. Barros, E. Hruschka, A. de Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, 03 2014.

[2] D. Xu and Y. jie Tian, "A comprehensive survey of clustering algorithms," *Annals of Data Science*, vol. 2, pp. 165–193, 2015.

[3] S. Sarfraz, V. Sharma, and R. Stiefelhagen, "Efficient parameter-free clustering using first neighbor relations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[4] U. V. Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, 2007.

[5] L. Kaufman and P. J. Roussew, *Finding Groups in Data - An Introduction to Cluster Analysis*. A Wiley-Science Publication John Wiley & Sons, 1990.

[6] R. Sibson, "SLINK: an optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–34, 1973.

[7] D. Defays, "An efficient algorithm for a complete-link method," *The Computer Journal*, vol. 20, no. 4, pp. 364–366, 1977.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, vol. KDD-96. AAAI Press, 1996, pp. 226–231.

[9] E. Elhamifar and R. Vidal, "Sparse subspace clustering: algorithm, theory, and applications," *IEEE transactions on pattern analysis and machine intelligence*, November 2013.

[10] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 1996, pp. 103–114.

[11] A. Kumar, A. Singh, and R. Singh, "An efficient hybrid-clustream algorithm for stream mining," in *2017 13th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, 2017, pp. 430–437.

[12] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "The clustree: indexing micro-clusters for anytime stream mining," *Knowledge and Information Systems*, vol. 29, no. 2, pp. 249–272, Nov 2011.

[13] J. Gama, P. Rodrigues, and L. Lopes, "Clustering distributed sensor data streams using local processing and reduced communication," *Intell. Data Anal.*, vol. 15, pp. 3–28, 01 2011.

[14] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, no. 3, p. 515–528, Mar. 2003.

[15] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "Streamkm++: A clustering algorithm for data streams," *ACM J. Exp. Algorithmics*, vol. 17, May 2012.

[16] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 133–142.

[17] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *In 2006 SIAM Conference on Data Mining*, 2006, pp. 328–339.

[18] R. Ahmed, G. Dalkılıç, and Y. Erten, "Dgstream: High quality and efficiency stream clustering algorithm," *Expert Systems with Applications*, vol. 141, p. 112947, 2020.

[19] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 132–149.

[20] J. Cai, J. Fan, W. Guo, S. Wang, Y. Zhang, and Z. Zhang, "Efficient deep embedded subspace clustering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 1–10.

[21] Y. Li, P. Hu, Z. Liu, D. Peng, J. T. Zhou, and X. Peng, "Contrastive clustering," in *2021 AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[22] X. Zhan, J. Xie, Z. Liu, Y.-S. Ong, and C. C. Loy, "Online deep clustering for unsupervised representation learning," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 6687–6696.

[23] L. Alzubaidi, J. Zhang, and A. Humaidi, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, p. 53, 2021.

[24] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation." *ACM TKDD*, 2007.

[25] C. Zahn, "Graph-theoretical methods for detecting and describing Gestalt clusters," *IEEE Trans. on Computers*, vol. C-20, no. 1, pp. 68–86, Jan. 1971.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, p. 770–778.

[27] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979.

[28] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, Sep. 1973.

[29] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, p. 553–569, 1983.

[30] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[32] A. Fred and A. Jain, "Robust data clustering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[33] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007, p. 1027–1035.

[34] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, ser. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, p. 849–856.

[35] J. H. W. Jr., "Hierarchical grouping to optimize an objective function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

[36] C. You, C. Li, D. Robinson, and R. Vidal, "Oracle based active set algorithm for scalable elastic net subspace clustering," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3928–3937, 2016.

[37] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-SNE," *Journal of Machine Learning Research*, vol. 9: 2579–2605, Nov 2008.