Fast Exact k-Means, k-Medians and Bregman Divergence Clustering in 1D

Allan Grønlund* Kasper Green Larsen† Alexander Mathiasen‡
Jesper Sindahl Nielsen § Stefan Schneider ¶ Mingzhou Song ‖

Abstract

The k-Means clustering problem on n points is NP-Hard for any dimension $d \geq 2$, however, for the 1D case there exist exact polynomial time algorithms. The current state of the art is an $O(kn^2)$ dynamic programming algorithm that uses $O(kn)$ space. We present a new algorithm improving this to $O(n \lg n + kn)$ time and optimal $O(n)$ space. We generalize our algorithm to work for the absolute distance instead of squared distance and to work for any Bregman Divergence as well.

*Aarhus University. Email: jallan@cs.au.dk
†Aarhus University. Email: larsen@cs.au.dk
‡Aarhus University. Email: alexander.mathiasen@gmail.com
§Aarhus University. Email: jasn@cs.au.dk
¶University of California, San Diego. Email: stschnei@cs.ucsd.edu
‖New Mexico State University. Email: joemsong@cs.nmsu.edu

1 Introduction

Clustering is the problem of grouping elements into clusters such that each element is similar to the elements in the cluster assigned to it and not similar to elements in any other cluster. It is one of, if not the, primary problem in the area of machine learning known as Unsupervised Learning and no clustering problem is as famous and widely considered as the k-Means problem: Given a multiset $\mathcal{X} = \{x_1, ..., x_n\} \subset \mathbb{R}^d$ find k centroids $\mathcal{M} = \{\mu_1, ..., \mu_k\} \subset \mathbb{R}^d$ minimizing $\sum_{x \in \mathcal{X}} \min_{\mu \in M} ||x - \mu||^2$. Several NP-Hardness results exist for finding the optimal k-Means clustering in general, forcing one to turn towards heuristics. k-Means is NP-hard even for $k = 2$ and general dimension [3] and it is also NP-hard for $d = 2$ and general k [14]. Even hardness of approximation results exist [13, 7]. In [7] the authors show there exists a $\varepsilon > 0$ such that it is NP-hard to approximate k-Means to within a factor $1 + \varepsilon$ of optimal, and in [13] it is proved that $\varepsilon \geq 0.0013$. On the upper bound side the best known polynomial time approximation algorithm for k-Means has an approximation factor of 6.357 [2]. In practice, Lloyd's algorithm is a popular iterative local search heuristic that starts from some random or arbitrary clustering. The running time of Lloyd's algorithm is $O(tkn)$ where t is the number of rounds of the local search procedure. In theory, if Lloyd's algorithm is run to convergence to a local minimum, t could be exponential and there is no guarantee on how well the solution found approximates the optimal solution [5, 17]. Lloyd's algorithm is often combined with the effective seeding technique for selecting initial centroids due to [6] that gives an expected $O(\lg k)$ approximation ratio for the initial clustering, which can then improved further by Lloyd's algorithm.

For the one-dimensional case, the k-Means problem is not NP-hard. In particular, there is an $O(kn^2)$ time and $O(kn)$ space dynamic programming solution for the 1D case, due to work by [18]. The 1D k-Means problem is encountered surprisingly often in practice, some examples being in data analysis in social networks, bioinformatics and retail market [4, 12, 16].

It is only natural to try other reasonable distance measures for the data considered and define different clustering problems. There are many other choices than the sum of squares of the Euclidian distances that define k-Means. For instance, one could use any L_p norm instead. The special case of $p = 1$ is known as k-Medians clustering and has also received considerable attention. The k-Medians problems is also NP-hard and the best polynomial time approximation algorithms has an approximation factor of 2.633 [7].

In [8] the authors consider and define clustering with Bregman Divergences. Bregman Divergence generalize squared Euclidian distance and thus Bregman Clusterings include the k-Means problem, as well as a wide range of other clustering problems that can be defined from Bregman Divergences like e.g. clustering with KullbackLeibler divergence as the cost. Interestingly, the heuristic local search algorithm for Bregman Clustering [8] is basically the same approach as Lloyd's algorithm for k-Means. Clustering with Bregman Divergences is clearly NP-Hard as well since it includes k-Means clustering. We refer the reader to [8] for more about the general problem. For the 1D version of the problem, [15] generalized the algorithm from [18] to the k-Medians problems and Bregman Divergences achieving the same $O(kn^2)$ time and $O(kn)$ space bounds.

1.1 Our Results

In this paper we give theoretically and practically efficient algorithm for 1D clustering problems, in particular k-Means.

The k-Means clustering problem in 1D is defined as follows. Given $\mathcal{X} = \{x_1, ..., x_n\} \subset \mathbb{R}$ find centroids $\mathcal{M} = \{\mu_1, ..., \mu_k\} \subset \mathbb{R}$ minimizing the cost

$$\sum_{x \in \mathcal{X}} \min_{\mu \in \mathcal{M}} (x - \mu)^2$$

The main result of this paper is a new algorithm for 1D k-Means that run in $O(n \lg n + kn)$ time using optimal $O(n)$ space, or $O(kn)$ time if the input is already sorted. This in an improvement by a factor of n in time and k in space compared to the existing solution. As in [18] the algorithm computes the cost of the optimal clustering for all $k' \leq k$. The constant factors hidden by the O-notation are small and we

expect the algorithm to be very efficient in practice. In 1D Lloyd's algorithm can be implemented to run in $O(n \lg n + tk \lg n)$ time where t is the number of rounds, and we expect that our algorithm can compute the optimal clustering for reasonable k in essentially the same time as Lloyd's algorithm can approximate it.

The k-Medians problem is to compute a clustering that minimize the sum of absolute distances to the centroid, i.e. minimize

$$\sum_{x \in \mathcal{X}} \min_{\mu \in \mathcal{M}} |x - \mu|$$

Our algorithm generalize naturally to solve this problem in $O(n \lg n + kn)$ time using $O(n)$ space, or $O(kn)$ time if the input is sorted.

Let f be a differentiable real-valued strictly convex function. The Bregman Divergence D_f induced by f is defined as

$$D_f(x, y) = f(x) - f(y) - \nabla_f(y)(x - y)$$

Notice that the Bregman Divergence induced from $f(x) = x^2$, gives squared Euclidian Distance. Bregman divergences are not metrics since they are not necessarily symmetric and the triangle inequality is not necessarily satisfied. They do however have many redeeming qualities, for instance Bregman Divergences are convex in the first argument, albeit not the second, see [8, 9] for a more comprenhensive treatment.

The Bregman Clustering problem as defined in [8] is to find k centroids $\mathcal{M} = \{\mu_1, ..., \mu_k\}$ that minimize

$$\sum_{x \in \mathcal{X}} \min_{\mu \in \mathcal{M}} D_f(x, \mu)$$

where D_f is a Bregman Divergence. For our case, where the inputs $x, y \in \mathbb{R}$, we assume that computing a Bregman Divergence, i.e. evaluating f and its derivative, takes constant time. We show that our algorithm also naturally generalize to 1D clustering using any Bregman Divergence to define the cluster cost while still maintaing a running time of $O(n \lg n + kn)$ and $O(n)$ space, or $O(kn)$ time if the input is already sorted.

1.2 Outline

The algorithm we propose is a more efficicient version of the dynamic programming algorithm of [18]. In Section 2 we describe the existing $O(kn^2)$ time algorithm that uses $O(kn)$ space. In Section 3 we show how to compute the same values as the old algorithm using only $O(kn)$ time and $O(n)$ space. Finally, in Section 4 we show how our new algorithm generalizes to different cluster costs than squared Euclidian distance.

2 The $O(kn^2)$ Dynamic Programming Algorithm

In this section, we describe the previous $O(kn^2)$ time and $O(kn)$ space algorithm presented in [18]. We also introduce the definitions and notation we use in our new algorithm. We will always assume sorted input $x_1 \leq ... \leq x_n \in \mathbb{R}$. If the input is not sorted, we start by sorting it in $O(n \lg n)$ time. We also remark that there could be many ways of partitioning the point set and computing centroids that achieve the same cost. This is for instance the case if the input is n identical points. The task at hand is to find any optimal solution.

Let $CC(i, j) = \sum_{\ell=i}^{j} (x_\ell - \mu_{i,j})^2$ be the cost of grouping $x_i, ..., x_j$ into one cluster with the optimal choice of centroid, $\mu_{i,j} = \frac{1}{j-i+1} \sum_{\ell=i}^{j} x_\ell$, the arithmetic mean of the points.

Lemma 1. *There is an $O(n)$ space data structure that can compute $CC(i, j)$ in $O(1)$ time for any $i \leq j$ using $O(n)$ time preprocessing.*

Proof. This is a standard application of prefix sums and works as follows. By definition,

$$CC(i,j) = \sum_{\ell=i}^{j}(x_\ell - \mu_{i,j})^2 = \sum_{\ell=i}^{j} x_\ell^2 + \mu_{i,j}^2 - 2x_\ell\mu_{i,j} = (j-i+1)\mu_{i,j}^2 + \mu_{i,j}\sum_{\ell=i}^{j} x_\ell + \sum_{\ell=i}^{j} x_\ell^2.$$

This cost can be computed in constant time with access to prefix sum arrays of $x_1, \ldots, x_n$ and $x_1^2, \ldots, x_n^2$. The centroid $\mu_{i,j}$ is also easily computed from the prefix sums. □

2.1 Algorithm Sketch

The algorithm computes the optimal clustering using i clusters for all prefixes of input points $x_1, \ldots, x_m$, for $m = 1, \ldots, n$, and for all $i = 1, \ldots, k$ using Dynamic Programming as follows.

Let $D[i][m]$ be the cost of optimally clustering $x_1, ..., x_m$ into i clusters. For $i = 1$ the cost of optimally clustering $x_1, ..., x_m$ into one cluster is the cluster cost $CC(1, m)$. That is, $D[1][m] = CC(1, m)$ for all m. This can be computed in $O(n)$ time by Lemma 1.

For $i > 1$

$$D[i][m] = \min_{j=1}^{m} D[i-1][j-1] + CC(j,m) \tag{1}$$

Notice that $D[i-1][j-1]$ is the cost of optimally clustering $x_1, ..., x_{j-1}$ into $i-1$ clusters and $CC(j, m)$ is the cost of clustering $x_j, ..., x_m$ into one cluster. This makes x_j the first point in the last and rightmost cluster. Let $T[i][m]$ be the argument that minimizes (1)

$$T[i][m] := \arg\min_{j=1}^{m} D[i-1][j-1] + CC(j,m) \tag{2}$$

It is possible there exists multiple j obtaining same minimal value for (1). To make the optimal clustering $\tilde{\mathcal{C}}^{(m)}$ unique, such ties are broken in favour of smaller j.

Notice $x_{T[i][m]}$ is the first point in the rightmost cluster of the optimal clustering. Thus, given T one can find the optimal solution by standard backtracking:

$$\begin{aligned}\tilde{X}_k &= \{x_{T[k][n]}, ..., x_n\},\\ \tilde{X}_{k-1} &= \{x_{T[k-1][T[k][n]-1]}, ..., x_{T[k][n]-1}\}\\ &\vdots\end{aligned}$$

Here $\tilde{X}_i$ is the i'th cluster in the optimal clustering. One can naively compute each entry of D and T using (1) and (2). This takes $O(n)$ time for each cell, thus D and T can be computed in $O(kn^2)$ time using $O(kn)$ space. This is exactly what is described in [18].

3 New Algorithm

The idea in our new algorithm is simply to compute the tables D and T faster, by reducing the time to compute each row of D and T to $O(n)$ time instead of $O(n^2)$ time. This improvement exploits a monotonicity property of the values stored in a row of T. This is explained in Section 3.1, resulting in an $O(kn)$ time and $O(kn)$ space solution, assuming sorted inputs. Section 3.2 then shows how to reduce the space usage to just $O(n)$ while retaining $O(kn)$ running time.

3.1 Fast Algorithm From Monotone Matrices

In this section we reduce the problem of computing a row of D and T to searching an implicitly defined $n \times n$ matrix of a special form, which allows us to compute each row of D and T in linear time.

Define $\mathrm{C}_i[m][j]$ as the cost of the optimal clustering of $x_1, \ldots, x_m$ using i clusters, restricted to having the rightmost cluster (largest cluster center) contain the elements $x_j, \ldots, x_m$. For convenience, we define $\mathrm{C}_i[m][j]$ for $j > m$ as the cost of clustering $x_1, \ldots, x_m$ into $i-1$ clusters, i.e. the last cluster is empty. This means that C_i satisfies:

$$\mathrm{C}_i[m][j] = D[i-1][\min\{j-1, m\}] + CC(j, m)$$

where by definition $CC(j, m) = 0$ when $j > m$ (which is consistent with the definition in Section 2). We have that $D[i][m]$ relates to C_i as follows:

$$D[i][m] = \min_j \mathrm{C}_i[m][j]$$

where ties are broken in favor of smaller j (as defined in Section 2.1).

This means that when we compute a row of D and T, we are actually computing $\min_j \mathrm{C}_i[m][j]$ for all $m = 1, \ldots, n$. We think of C_i as an $n \times n$ matrix with rows indexed by m and columns indexed by j. With this interpretation, computing the i'th row of D and T corresponds to computing for each row r in C_i, the column index c that corresponds to the smallest value in row r. In particular, the entries $D[i][m]$ and $T[i][m]$ correpond to the value and the index of the minimum entry in the m'th row of C_i respectively. The problem of finding the minimum value in every row of a matrix has been studied before [1]. First we need the definition of a monotone matrix.

Definition 1. *[1] Let A be a matrix with real entries and let* $\mathrm{argmin}(i)$ *be the index of the leftmost column containing the minimum value in row i of A. A is said to be* monotone *if $a < b$ implies that* $\mathrm{argmin}(a) \leq \mathrm{argmin}(b)$*. A is* totally monotone *if all of its submatrices are monotone.*

In [1], the authors showed the following:

Theorem 1. *[1] Finding* $\mathrm{argmin}(i)$ *for each row i of an arbitrary $n \times m$ monotone matrix requires $\Theta(m \lg n)$ time, whereas if the matrix is totally monotone, the time is $O(m)$ when $m > n$ and is $O(m(1 + \lg(n/m)))$ when $m < n$.*

The fast algorithm for totally monotone matrices is known as the *SMAWK* algorithm and we will refer to it by that (cool) name.

Let's relate this to the 1D k-Means clustering problem. That C_i is monotone means that if we consider the optimal clustering of the points $x_1, \ldots, x_a$ with i clusters, then if we start adding more points $x_{a+1} \leq \cdots \leq x_b$ after x_a, then the first (smallest) point in the last of the i clusters can only increase (move right) in the new optimal clustering of $x_1, \ldots, x_b$. This sounds like it should be true for 1D k-Means and it turns out it is. Thus, applying the algorithm for monotone matrices, we can fill a row of D and T in $O(n \lg n)$ time leading to an $O(kn \lg n)$ time algorithm for 1D k-Means, which is already a great improvement.

However, as we show below, the matrix C_i induced by the 1D k-Means problem is in fact totally monotone:

Lemma 2. *The matrix C_i is totally monotone.*

Proof. As [1] remarks, a matrix A is totally monotone if all its 2×2 submatrices are monotone. To prove that C_i is totally monotone, we thus need to prove that for any two row indices a, b with $a < b$ and two column indices u, v with $u < v$, it holds that if $\mathrm{C}_i[a][v] < \mathrm{C}_i[a][u]$ then $\mathrm{C}_i[b][v] < \mathrm{C}_i[b][u]$.

Notice that these values correspond to the costs of clustering elements $x_1, \ldots, x_a$ and $x_1, \ldots, x_b$, starting the rightmost cluster with element x_u and x_v respectively. Since $\mathrm{C}_i[m][j] = D[i-1][\min\{j-1, m\}] + CC(j, m)$, this is the same as proving that

$$\begin{aligned} &D[i-1][\min\{v-1, m\}] + CC(v, a) < D[i-1][\min\{u-1, m\}] + CC(u, a) \Rightarrow \\ &D[i-1][\min\{v-1, m\}] + CC(v, b) < D[i-1][\min\{u-1, m\}] + CC(u, b) \end{aligned}$$

which is true if we can prove that $CC(v, b) - CC(v, a) \leq CC(u, b) - CC(u, a)$. Rearranging terms, what we need to prove is that for any $a < b$ and $u < v$, it holds that:

$$CC(v, b) + CC(u, a) \leq CC(u, b) + CC(v, a).$$

As a side note, this is called the *concave* property in early Dynamic Programming papers [20, 11, 19] and has been used to significantly speed up Dynamic Programming algorithms for other problems.

We start by handling the special case where $v > a$. In this case, we have by definition that $CC(v, a) = 0$, thus we need to show that $CC(v, b) + CC(u, a) \leq CC(u, b)$. This is the case since any point amongst $x_u, \ldots, x_b$ is included in at most one of $x_v, \ldots, x_b$ and $x_u, \ldots, x_a$ (since $a < v$). Thus $CC(v, b) + CC(u, a)$ is the cost of taking two disjoint and consecutive subsets of the points $x_u, \ldots, x_b$ and clustering the two sets using the optimal choice of centroid in each. Clearly this cost is less than clustering all the points using one centroid.

We now turn to the general case where $u < v \leq a < b$. Let $\mu_{v,a}$ be the mean of $x_v, \ldots, x_a$ and $\mu_{u,b}$ be the mean of $x_u, \ldots, x_b$ and assume that $\mu_{v,a} \leq \mu_{u,b}$ (the other case is symmetric). Finally, let $CC(w, z)_\mu = \sum_{\ell=w}^{z} (x_\ell - \mu)^2$ denote the cost of grouping the elements $x_w, \ldots, x_z$ in a cluster with centroid μ. Split the cost $CC(u, b)$ into the cost of the elements $x_u, \ldots, x_{v-1}$ and the cost of the elements $x_v, \ldots, x_b$ as

$$CC(u, b) = \sum_{\ell=u}^{b} (x_\ell - \mu_{u,b})^2 = \sum_{\ell=u}^{v-1} (x_\ell - \mu_{u,b})^2 + \sum_{\ell=v}^{b} (x_\ell - \mu_{u,b})^2 = CC(u, v-1)_{\mu_{u,b}} + CC(v, b)_{\mu_{u,b}}.$$

We trivially get $CC(v, b)_{\mu_{u,b}} \geq CC(v, b)$ since $CC(v, b)$ is the cost using the optimal centroid. Secondly,

$$CC(u, v-1)_{\mu_{u,b}} + CC(v, a) \geq CC(u, v-1)_{\mu_{v,a}} + CC(v, a) = CC(u, a)_{\mu_{v,a}} \geq CC(u, a)$$

since $\mu_{v,a} \leq \mu_{u,b}$ and all elements $x_u, \ldots, x_{v-1}$ are less than or equal to $\mu_{v,a}$ (since $\mu_{v,a}$ is the mean of points $x_v, \ldots, x_a$ that all are greater than $x_u, \ldots, x_{v-1}$). Combining the results, we see that:

$$CC(v, b) + CC(u, a) \leq CC(v, b)_{\mu_{u,b}} + CC(u, v-1)_{\mu_{u,b}} + CC(v, a) = CC(u, b) + CC(v, a).$$

This completes the proof. □

Theorem 2. *Computing an optimal k-Means clustering of a sorted input of size n takes $O(kn)$ time.*

3.2 Reducing Space Usage

In the following, we show how to reduce the space usage to just $O(n)$ while maintaining $O(kn)$ running time using a space reduction technique of Hirschberg [10]. First observe that each row of T and D only refers to the previous row. Thus one can clearly "forget"row $i - 1$ when we are done computing row i. The problem is that if we don't store all of T, we cannot backtrack and find the optimal solution. In the following, we present an algorithm that avoids the table T entirely.

Our key observation is the following: Assume $k > 1$ and that for every prefix $x_1, \ldots, x_m$, we have computed the optimal cost of clustering $x_1, \ldots, x_m$ into $\lfloor k/2 \rfloor$ clusters. Note that this is precisely the set of values stored in the $\lfloor k/2 \rfloor$'th row of D. Assume furthermore that we have computed the optimal cost of clustering every suffix $x_m, \ldots, x_n$ into $k - \lfloor k/2 \rfloor$ clusters. Let us denote these costs by $\tilde{D}[k - \lfloor k/2 \rfloor][m]$ for $m = 1, \ldots, n$. Then clearly the optimal cost of clustering $x_1, \ldots, x_n$ into k clusters is given by:

$$D[k][n] = \min_{j=1}^{n} D[\lfloor k/2 \rfloor][j] + \tilde{D}[k - \lfloor k/2 \rfloor][j+1]. \tag{3}$$

Our main idea is to first compute row $\lfloor k/2 \rfloor$ of D and row $k - \lfloor k/2 \rfloor$ of $\tilde{D}$ using linear space. From these two, we can compute the argument j minimizing (3). We can then split the reporting of the optimal clustering into two recursive calls, one reporting the optimal clustering of points $x_1, \ldots, x_j$ into $\lfloor k/2 \rfloor$ clusters, and one call reporting the optimal clustering of $x_{j+1}, \ldots, x_n$ into $k - \lfloor k/2 \rfloor$ clusters. When the recursion bottoms out with $k = 1$, we can clearly report the optimal clustering using linear space and time as this is just the full set of points.

From Section 3.1 we already know how to compute row $\lfloor k/2 \rfloor$ of D using linear space: Simply call SMAWK to compute row i of D for $i = 1, \ldots, \lfloor k/2 \rfloor$, where we throw away row $i - 1$ of D (and don't

even store T) when we are done computing row i. Now observe that table $\tilde{D}$ can be computed by taking our points $x_1, \ldots, x_n$ and reversing their order by negating the values. This way we obtain a new ordered sequence of points $\tilde{X} = \tilde{x}_1 \leq \tilde{x}_2 \leq \cdots \leq \tilde{x}_n$ where $\tilde{x}_i = -x_{n-i+1}$. Running SMAWK repeatedly for $i = 1, \ldots, k - \lfloor k/2 \rfloor$ on the point set $\tilde{X}$ produces a table $\hat{D}$ such that $\hat{D}[i][m]$ is the optimal cost of clustering $\tilde{x}_1, \ldots, \tilde{x}_m = -x_n, \ldots, -x_{n-m+1}$ into i clusters. Since this cost is the same as clustering $x_{n-m+1}, \ldots, x_n$ into i clusters, we get that the $(k - \lfloor k/2 \rfloor)$'th row of $\hat{D}$ is identical to the i'th row of $\tilde{D}$ if we reverse the order of the entries.

To summarize our algorithm for reporting the optimal clustering, do as follows: Let L be an initially empty output list of clusters. If $k = 1$, append to L a cluster containing all points. Otherwise ($k > 1$), use SMAWK on $x_1, \ldots, x_n$ and $-x_n, \ldots, -x_1$ to compute row $\lfloor k/2 \rfloor$ of D and row $k - \lfloor k/2 \rfloor$ of $\tilde{D}$ using linear space (by evicting row $i - 1$ from memory when we have finished computing row i) and $O(kn)$ time. Compute the argument j minimizing (3) in $O(n)$ time. Evict row $\lfloor k/2 \rfloor$ of D and row $k - \lfloor k/2 \rfloor$ of $\tilde{D}$ from memory. Recursively report the optimal clustering of points $x_1, \ldots, x_j$ into $\lfloor k/2 \rfloor$ clusters (which appends the output to L). When this terminates, recursively report the optimal clustering of points $x_{j+1}, \ldots, x_n$ into $k - \lfloor k/2 \rfloor$ clusters. When the algorithm terminates, L contains the optimal clustering of $x_1, \ldots, x_n$ into k clusters.

At any given time, our algorithm uses only $O(n)$ space. To see this, first note that we evict all memory used to compute the value j minimizing (3) before recursing. Furthermore, we complete the first recursive call (and evict all memory used) before starting the second. Finally, for the recursion, we don't need to make a copy of points $x_1, \ldots, x_j$. It suffices to remember that we are only working on the subset of inputs $x_1, \ldots, x_j$.

Now let $F(n, k)$ denote the time used by the above algorithm to compute the optimal clustering of n sorted points into k clusters. Then there is some constant $C > 0$ such that $F(n, k)$ satisfies the recurrence:

$$F(n, 1) \leq Cn,$$

and for $k > 1$:

$$F(n, k) \leq \max_{j=1}^{n} F(j, \lfloor k/2 \rfloor) + F(n - j, k - \lfloor k/2 \rfloor) + Cnk.$$

We claim that $F(n, k)$ satisfies $F(n, k) \leq 3Ckn$. We prove the claim by induction in k. The base case $k = 1$ follows trivially by inspection of the formula for $F(n, 1)$. For the inductive step $k > 1$, we use the induction hypothesis to conclude:

$$\begin{aligned} F(n, k) &\leq \max_{j=1}^{n} 3Cj\lfloor k/2 \rfloor + 3C(n - j)(k - \lfloor k/2 \rfloor) + Cnk \\ &\leq \max_{j=1}^{n} 3Cj\lceil k/2 \rceil + 3C(n - j)\lceil k/2 \rceil + Cnk \\ &= 3Cn\lceil k/2 \rceil + Ckn. \end{aligned}$$

For $k > 1$, we have that $\lceil k/2 \rceil \leq (2/3)k$, therefore:

$$\begin{aligned} F(n, k) &\leq 3Cn(2/3)k + Ckn \\ &= 3Ckn. \end{aligned}$$

Which is what we needed to prove.

Theorem 3. *Computing an optimal k-Means clustering of a sorted input of size n takes $O(kn)$ time and uses $O(n)$ space.*

4 Extending to More Distance Measures

In the following we show how to generalize our algorithm to Bregman Divergences and sum of absolute distances while retaining the same running time and space usage.

4.1 Bregman Divergence and Bregman Clustering

In this section we show how our algorithm generalizes to any Bregman Divergence. First, let us remind ourselves what a Bregman Divergence and a Bregman Clustering is. Let f be a differentiable real-valued strictly convex function. The Bregman Divergence D_f defined by f is defined as

$$D_f(x, y) = f(x) - f(y) - \nabla_f(y)(x - y)$$

Bregman Clustering. The Bregman Clustering problem as defined in [8], is to find a clustering that minimize

$$\sum_{x \in \mathcal{X}} \min_{\mu} D_f(x, \mu)$$

Notice that the cluster center is the second argument of the Bregman Divergence. This is important since Bregman Divergences are not in general symmetric.

For the purpose of 1D clustering, we mention two important properties of Bregman Divergences. For any Bregman Divergence, the unique element that minimizes the summed distance to a multiset of elements is the mean of the elements, exactly as it was for squared Euclidian distance. This is in one sense the defining property of Bregman Divergences [8].

The second important is the linear separator property, which is very important for clustering with Bregman Divergences but also very relevavant to Bregman Voronoi Diagrams [8, 9].

Linear Separators For Bregman Divergences. For all Bregman divergences, the locus of points that are equidistant to two fixed points μ_1, μ_2 in terms of a Bregman divergence is given by

$$\{x \in \mathcal{X} \mid D_f(x, p) = D_f(x, q)\} = \{x \in \mathcal{X} \mid x(\nabla_f(\mu_1) - \nabla_f(\mu_2)) = f(\mu_1) - \mu_1 \nabla_f(\mu_1) - f(\mu_2) + \mu_2 \nabla_f(\mu_2)\}$$

which corresponds to a hyperplane. Also, the points μ_1, μ_2 sits on either side of the hyperplane and the Voronoi cells defined using Bregman divergences are connected.

This means, in particular, that between any two points in 1D, $\mu_1 < \mu_2$, there is a hyperplane (point) h with $\mu_1 < h < \mu_2$ and all points smaller than h are closer to μ_1 and all points larger than h are closer to μ_2. We capture what we need from this observation in a simple "distance" lemma:

Lemma 3. *Given two fixed real numbers $\mu_1 < \mu_2$, then for any point $x_r \geq \mu_2$, we have $D_f(x_r, \mu_1) > D_f(x_r, \mu_2)$, and for any point $x_l \leq \mu_1$ we have $D_f(x_l, \mu_1) < D_f(x_l, \mu_2)$*

Computing Cluster Costs for Bregman Divergences. Since the mean minizes Bregman Divergences, the centroids used in optimal clusterings are unchanged compared to the k-Means case. The prefix sums idea used to implement the data structure used for Lemma 1 generalizes to Bregman Divergences as observed in [15] (under the name Summed Area Tables). The formula for computing the cost of grouping the points $x_i, \ldots, x_j$ in one cluster is as follows. Let $\mu_{i,j} = \frac{1}{j-i+1} \sum_{\ell=i}^{j} x_\ell$ be the arithmetic mean of the points $x_i, \ldots, x_j$, then

$$\begin{aligned} CC(i, j) &= \sum_{\ell=i}^{j} D_f(x_\ell, \mu_{i,j}) = \sum_{\ell=i}^{j} f(x_\ell) - f(\mu_{i,j}) - \nabla_f(\mu_{i,j})(x_\ell - \mu_{i,j}) \\ &= \left(\sum_{\ell=i}^{j} f(x_\ell)\right) - (j - i + 1) f(\mu_{i,j}) - \nabla_f(\mu_{i,j}) \left(\left(\sum_{\ell=i}^{j} x_\ell\right) - (j - i + 1)\mu_{i,j}\right) \end{aligned}$$

It follows that the Bregman Divergence cost of a consecutive subset of input points and the centroid can be computed in in constant time with stored prefix sums for $x_1, \ldots, x_n$ and $f(x_1), \ldots, f(x_n)$.

Totally Monotone Matrix. The only properties we used in Section 3.1 to prove that the matrix C_i is totally monotone, is that the mean is the minimizer of the sum of distances to a multiset of points, and that

$$CC(u, v-1)_{\mu_{u,b}} + CC(v, a) \geq CC(u, v-1)_{\mu_{v,a}} + CC(v, a) = CC(u, a)_{\mu_{v,a}}$$

when $\mu_{v,a} \leq \mu_{u,b}$ and all elements in $x_u, \ldots, x_{v-1} \leq \mu_{v,a}$. This is clearly still true by Lemma 3.

Theorem 4. *Computing an optimal Bregman Clustering of a sorted input of size n takes $O(kn)$ time and $O(n)$ space.*

4.2 k-Medians - Clustering with sum of absolute values

For the k-Medians problem we replace the the sum of squared Euclidian distances with the sum of absolute distances. Formally, the k-Medians problem is to compute a clustering minimizing

$$\sum_{x \in \mathcal{X}} \min_{\mu \in \mathcal{M}} |x - \mu|$$

Note that in 1D, all L_p norms are the same and reduce to this case. Also note that the minimizing centroid for a cluster is no longer the mean of the points in that cluster, but the median. To solve this problem, we change the centroid to be the median, and if there an even number of points, we fix the median to be the exact middle point between the two middle elements, making the choice of centroid unique.

As for Bregman Divergences, we need to show that we can compute the cost $CC(i, j)$ with this new cost in constant time. Also, we need to compute the centroid in constant time and we need to argue that the implicit matrix C_i is totally monotone. The arguments are essentially the same, but for completeness we briefly cover them below.

Computing Cluster Costs for Absolute Distances. Not surprisingly, using prefix sums still allow constant time computation of $CC(i, j)$. Let $m_{i,j} = \frac{j+i}{2}$, and compute the centroid as $\mu_{i,j} = \frac{x_{\lfloor m_{i,j} \rfloor} + x_{\lceil m_{i,j} \rceil}}{2}$

$$CC(i, j) = \sum_{\ell=i}^{j} |x_\ell - \mu_{i,j}| = \sum_{\ell=i}^{\lfloor m_{i,j} \rfloor} \mu_{i,j} - x_\ell + \sum_{\ell=1+\lfloor m_{i,j} \rfloor}^{j} x_\ell - \mu_{i,j}$$

which can be computed in constant time with access to a prefix sum table of $x_1, \ldots, x_n$. This was also observed in [15].

Totally Monotone Matrix. The totally monotone matrix argument above for Bregman Divergences (and for squared Euclidian distance) is still valid since first of all, we still have $x_u, \ldots, x_{v-1} \leq \mu_{v,a}$ as $\mu_{v,a}$ is the median of points all greater than $x_u, \ldots, x_{v-1}$. Furthermore, it still holds that when $\mu_{v,a} \leq \mu_{u,b}$ and all elements $x_u, \ldots, x_{v-1}$ are less than or equal to $\mu_{v,a}$, then:

$$CC(u, v-1)_{\mu_{u,b}} + CC(v, a) \geq CC(u, v-1)_{\mu_{v,a}} + CC(v, a) = CC(u, a)_{\mu_{v,a}}$$

Theorem 5. *Computing an optimal k-Medians Clustering of a sorted input of size n takes $O(kn)$ time and $O(n)$ space.*

References

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

[2] S. Ahmadian, A. Norouzi-Fard, O. Svensson, and J. Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *CoRR*, abs/1612.07925, 2016.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[4] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni. Analysis of ego network structure in online social networks. In *Privacy, security, risk and trust (PASSAT), 2012 international conference on and 2012 international confernece on social computing (SocialCom)*, pages 31–40. IEEE, 2012.

[5] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, SCG '06, pages 144–153, New York, NY, USA, 2006. ACM.

[6] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[7] P. Awasthi, M. Charikar, R. Krishnaswamy, and A. K. Sinop. The hardness of approximation of euclidean k-means. In L. Arge and J. Pach, editors, *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, volume 34 of *LIPIcs*, pages 754–767. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[8] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *J. Mach. Learn. Res.*, 6:1705–1749, Dec. 2005.

[9] J.-D. Boissonnat, F. Nielsen, and R. Nock. Bregman voronoi diagrams. *Discrete & Computational Geometry*, 44(2):281–307, 2010.

[10] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[11] D. S. Hirschberg and L. L. Larmore. The least weight subsequence problem. *SIAM Journal on Computing*, 16(4):628–638, 1987.

[12] O. Jeske, M. Jogler, J. Petersen, J. Sikorski, and C. Jogler. From genome mining to phenotypic microarrays: Planctomycetes as source for novel bioactive molecules. *Antonie Van Leeuwenhoek*, 104(4):551–567, 2013.

[13] E. Lee, M. Schmidt, and J. Wright. Improved and simplified inapproximability for k-means. *Information Processing Letters*, 120:40–43, 2017.

[14] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. *The Planar k-Means Problem is NP-Hard*, pages 274–285. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[15] F. Nielsen and R. Nock. Optimal interval clustering: Application to bregman clustering and statistical mixture learning. *IEEE Signal Process. Lett.*, 21:1289–1292, 2014.

[16] D. Pennacchioli, M. Coscia, S. Rinzivillo, F. Giannotti, and D. Pedreschi. The retail market as a complex system. *EPJ Data Science*, 3(1):1, 2014.

[17] A. Vattani. k-means requires exponentially many iterations even in the plane. *Discrete & Computational Geometry*, 45(4):596–616, 2011.

[18] H. Wang and M. Song. Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, 2011.

[19] R. Wilber. The concave least-weight subsequence problem revisited. *Journal of Algorithms*, 9(3):418 – 425, 1988.

[20] F. F. Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 429–435, New York, NY, USA, 1980. ACM.