

# Towards Explaining the Speed of $k$ -Means

Bodo Manthey

University of Twente, Department of Applied Mathematics  
P. O. Box 217, 7500 AE Enschede, The Netherlands  
`b.manthey@utwente.nl`

The  $k$ -means method is a popular algorithm for clustering, known for its speed in practice. This stands in contrast to its exponential worst-case running-time. To explain the speed of the  $k$ -means method, a smoothed analysis has been conducted. We sketch this smoothed analysis and a generalization to Bregman divergences.

## 1 $k$ -Means Clustering

The problem of clustering data into classes is ubiquitous in computer science, with applications ranging from computational biology over machine learning to image analysis. The  $k$ -means method is a very simple and implementation-friendly local improvement heuristic for clustering. It is used to partition a set  $X$  of  $n$   $d$ -dimensional data points into  $k$  clusters. (The number  $k$  of clusters is fixed in advance.) In  $k$ -means clustering, our goal is not only to get a clustering of the data points, but also to get a *center*  $c_i$  for each cluster  $X_i$  of the clustering  $X_1, \dots, X_k$ . A center can be viewed as a representative of its cluster. We do not require centers to be among the data points, but they can be arbitrary points. The goal is to find a “good” clustering, where “good” means that the clustering should minimize the objective function

$$\sum_{i=1}^k \sum_{x \in X_i} \delta(x, c_i).$$

Here,  $\delta$  denotes a distance measure. In the following, we will mainly use squared Euclidean distances, i.e.,  $\delta(x, c_i) = \|x - c_i\|^2$ .

Of course, given the cluster centers  $c_1, \dots, c_k \in \mathbb{R}^d$ , each point  $x \in X$  should be assigned to the cluster  $X_i$  whose center  $c_i$  is closest to  $x$ . On the other hand, given a clustering  $X_1, \dots, X_k$  of the data points, each center  $c_i$  should be chosen as the center of mass  $\frac{1}{|X_i|} \cdot \sum_{x \in X_i} x$  in order to minimize the objective function. (This holds in particular for squared Euclidean distances, but also for Bregman divergences, which we consider in Section 3.6.)

The *k-means method* (often called *k-means* for short or *Lloyd's method* because it is based on ideas by Lloyd [18]) exploits that the centers define the clustering and the clustering defines the centers. Thus, we optimize alternately centers and clustering until we find a local optimum:

1. Choose initial cluster centers  $c_1, \dots, c_k$ .
2. For each data point  $x$ : If  $c_i$  is the center closest to  $x$ , then assign  $x$  to  $X_i$ . (We ignore ties here for simplicity.)

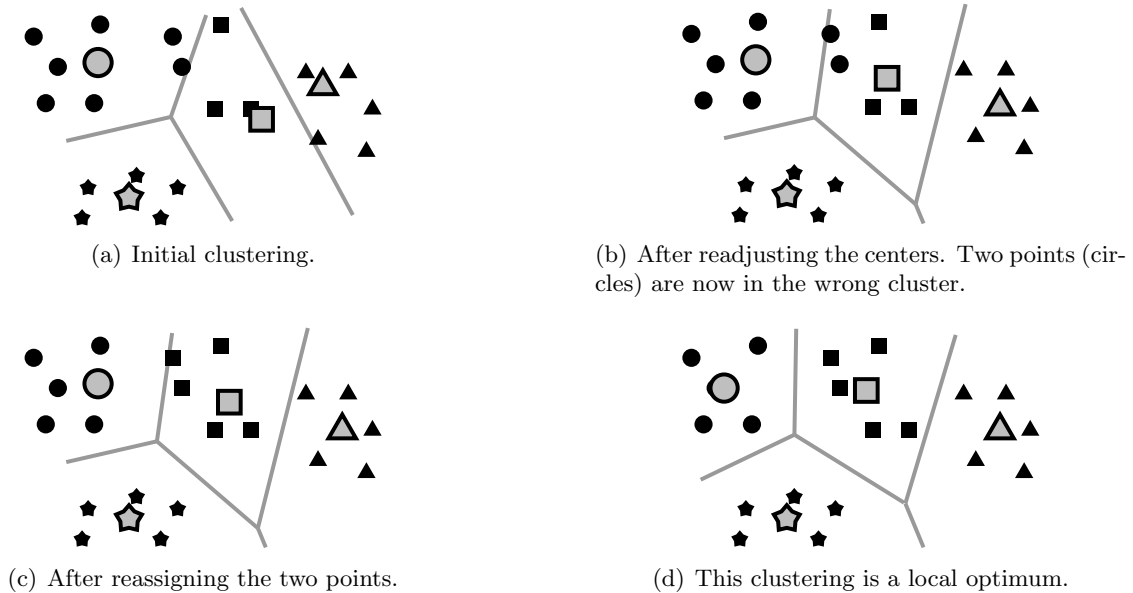


Figure 1: An example of  $k$ -means: (a) The initial centers (filled gray) induce an initial clustering. (b) We move the centers. Now some points prefer to be in a different cluster, whose center is closer to them. (c) The points are reassigned. (d) The centers are again adjusted. No further reassigning of points is needed, and  $k$ -means terminates.

3. For each  $i \in \{1, \dots, k\}$ : Set  $c_i = \frac{1}{|X_i|} \cdot \sum_{x \in X_i} x$ .

4. If anything has changed during the last iteration, then go back to Step 2.

Figure 1 shows an example. The gray lines are the Voronoi diagram of the cluster centers. This means that all points within one cell are closest to the cluster center in this cell.

The  $k$ -means method is one of the most popular clustering algorithms [9]. The main reason for its popularity is its speed: The number of iterations is often less than the number of data points [14]. This, however, is at stark contrast to its performance in theory: In the worst case,  $k$ -means requires  $2^{\Omega(k)}$  iterations [29]. By choosing  $k \in \Theta(n)$ , this can be made exponential in the number of data points. The only known upper bound for its running-time is  $\text{poly}(n^{kd})$  (we frequently use the short-hand  $\text{poly}(\dots)$  to denote an arbitrary but fixed polynomial), which is simply the number of possible *Voronoi-based clusterings* of  $n$  points in the  $d$ -dimensional space into  $k$  clusters [17]. A clustering is Voronoi-based if it can be represented by cluster centers such that every data point is assigned to its nearest cluster center. However, this bound is far from the observed speed of  $k$ -means. But what can we do in order to explain the performance of  $k$ -means theoretically? We can use the framework of smoothed analysis.

## 2 Smoothed Analysis

### 2.1 Worst-Case Analysis

Most widely used for the analysis of algorithms is the worst-case measure: The performance of an algorithm is measured by means of the most difficult instances. If the worst-case perfor-

mance of an algorithm is good, then this is a very strong statement: The algorithm has a good performance on every instance that it is given, no matter where this instance comes from.

Unfortunately, many algorithms do not have a good worst-case performance. But still, many of them show a remarkably good performance in practice. A famous example is the simplex method to solve linear optimization problems: Although the running-time of the simplex method is exponential in the worst-case, it often outperforms other algorithms (ellipsoid method or interior-point methods) that come with a polynomial worst-case running-time.

The reason for this discrepancy between worst-case and practical performance is that worst-case analysis is often dominated by pathological instances that do not even resemble realistic instances. In this sense, worst-case analysis' drawback is that it is often far too pessimistic.

## 2.2 Average-Case Analysis

A frequently used alternative to worst-case analysis is average-case analysis. Here, we do not measure by means of most difficult instances, thus avoiding the major drawback of worst-case analysis. Instead, we measure the expected performance on random inputs. But also average-case analysis has a crucial drawback: Instances from practice might have little in common with these random instances. Thus, even if an algorithm has a good average-case performance, this does not necessarily explain why or imply that this algorithm is efficient in practice.

The problem in fact is that random instances are often not typical instances, but they are instances that have very specific properties with overwhelming probability. And the fact that random inputs have specific properties does not mean that typical inputs share these properties.

## 2.3 Smoothed Analysis

To overcome the drawbacks of both average-case and worst-case analysis and in order to explain the speed of the simplex method, Spielman and Teng introduced the notion of smoothed analysis [27]. Smoothed analysis is a hybrid of average-case and worst-case analysis: First, an adversary chooses an instance. Second, this instance is slightly randomly perturbed. The smoothed performance is the expected performance, where the expectation is taken over the random perturbation. The adversary, trying to make the algorithm look as bad as possible, of course chooses an instance that maximizes the expected performance.

The perturbation reflects, for instance, that the input is subject to measurements or rounding errors. We might also think of the perturbation as modeling arbitrary circumstances, whose precise influence we do not know but where we have also no reason to believe that it is adversarial.

Let us now give a somewhat more formal definition of smoothed analysis. To do this, it helps to recapitulate worst-case and average-case analysis. Let  $t(X)$  denote the number of iterations that  $k$ -means needs on point set  $X$ . The worst-case running-time as a function of the number  $n$  of data points is the maximum running-time over all sets of  $n$  points:

$$T_{\text{worst}}(n) = \max_{X:|X|=n} t(X).$$

For the average-case analysis, we need a probability distribution according to which the instances are drawn. For instance, let  $P_n$  denote the probability distribution that draws  $n$  points in  $\mathbb{R}^d$  according to independent Gaussian distributions with mean 0 and standard deviation 1. Then we have

$$T_{\text{average}}(n) = \mathbb{E}_{X \sim P_n} t(X).$$

In smoothed analysis, as mentioned above, an adversary chooses an instance  $X$ . And then this instance is subject to a random perturbation. In our case, the adversary chooses  $X = \{x_1, \dots, x_n\}$ . Let  $Y = \{y_1, \dots, y_n\}$  be a set of  $n$  points drawn according to independent Gaussian distributions with mean 0 and standard deviation 1. We obtain our actual instance as  $Z = X + \sigma Y = \{x_1 + \sigma y_1, \dots, x_n + \sigma y_n\}$ . The parameter  $\sigma$ , which is called the *perturbation parameter*, measures the strength of the perturbation. It is the standard deviation of the perturbed points. Overall, we get the following definition for the smoothed running-time:

$$T_{\text{smoothed}}(n, \sigma) = \max_{X:|X|=n} \mathbb{E}_{Y \sim P_n}(t(X + \sigma Y)).$$

If the perturbation parameter  $\sigma$  is extremely small, then  $X + \sigma Y$  is approximately  $X$ . Thus, we obtain the worst case. If  $\sigma$  is extremely large, then the perturbation swamps out the original instances, and we obtain the average case. Smoothed analysis interpolates between these two extreme cases.

In order to get a realistic performance analysis, we are particularly interested in the case that  $\sigma$  is reasonably small. Here, this means that  $\sigma$  should be about  $1/\text{poly}(n)$ . Such perturbations take into account the aforementioned measurement or rounding errors or other circumstances. If the smoothed running-time of an algorithm is low, then this does not mean that bad inputs do not exist. But we must be very unlucky to get a bad input. Thus, it can explain the performance of the algorithm in practice.

Smoothed analysis has been invented by Spielman and Teng in order to explain the performance of the simplex method for linear programming [27]. Since its invention in 2001, smoothed analysis has been applied to a variety of different algorithms and problems. This includes linear programming [10, 26, 30], integer programming [6–8, 22–24], online algorithms [5, 25], searching and sorting [3, 13, 16, 19], game theory [11, 12], and local search [2, 15]. Spielman and Teng have written a comprehensive survey of smoothed analysis [28].

### 3 Smoothed Analysis of $k$ -Means

Now, we sketch the smoothed analysis of  $k$ -means. In a series of papers [1, 2, 20], it has been shown that the smoothed running-time of  $k$ -means is bounded by a polynomial in  $n$  and  $1/\sigma$  (we need  $k, d \leq n$ ; apart from that  $k$  and  $d$  are arbitrary and need not be constants). This explains why we do not see instances in practice, where  $k$ -means requires exponential time.

**Theorem 1 (Arthur et al. [1]).** *Let  $X \subseteq [0, 1]^d$  be an arbitrary set of  $n$  points, and assume that each point in  $X$  is perturbed by a Gaussian distribution with mean 0 and standard deviation  $\sigma$ , yielding a new set  $Z$  of points as described above. Then the expected running-time of  $k$ -means on  $Z$  is bounded by a polynomial in  $n$  and  $1/\sigma$ .*

In the following, we call the quantity analyzed, namely the expected running-time in the setting above, simply the *smoothed running-time of  $k$ -means*. Two remarks about this theorem are in order: First, instead of  $X \subseteq \mathbb{R}^d$ , we have  $X \subseteq [0, 1]^d$ . This is only a matter of scaling. What really matters is not the strength  $\sigma$  of the perturbation, but the ratio of  $\sigma$  to the size of the space from which the adversarial points come. Thus, we could replace  $[0, 1]^d$  by  $\mathbb{R}^d$  and consequently  $\sigma$  by  $\sigma \cdot \max_{x \in X} \|x\|$ . Second, the dependency of the bound on  $1/\sigma$  is quite natural: The smaller  $\sigma$ , the more powerful the adversary. Consequently, the bound on the running-time gets worse for smaller  $\sigma$ . For  $\sigma = 1/\text{poly}(n)$ , we get a polynomial expected running-time.

In the following, we will sketch a proof of the following weaker bound, which was the first bound on the smoothed running-time of  $k$ -means [2]. In order to simplify the proof sketch, we combine their ideas with ideas from Arthur et al. [1].

**Theorem 2 (Arthur, Vassilvitskii [2]).** *The smoothed running-time of  $k$ -means is bounded by  $\text{poly}(n^k, 1/\sigma)$ .*

### 3.1 High-Level Idea

The (very) high-level idea to prove a bound on the smoothed running-time of  $k$ -means is using a potential function argument: We use the objective function as the potential. If we can show that

1. initially the potential is at most  $P$  and
2. it decreases in every iteration of  $k$ -means by at least  $\Delta$ ,

then  $k$ -means can run for at most  $P/\Delta$  iterations. The reason is simply that the objective function cannot become negative. It is not too difficult to show that the potential is initially bounded by  $P = \text{poly}(n, 1/\sigma)$  with high probability (strictly speaking, this holds after the first iteration): All data points come from  $[0, 1]^d$ , and the perturbation is concentrated around 0. Thus, all perturbed points are within a reasonably small hypercube around the origin, and each point is not too far away from some center.

What remains to be done is to show that  $\Delta$  is at least  $\text{poly}(n^{-k}, \sigma)$  with high probability. If this holds, then we can bound the expected running-time by  $P/\Delta = \text{poly}(n^k, 1/\sigma)$ .

To show that the potential decreases in every iteration by at least  $\text{poly}(n^{-k}, \sigma)$ , we distinguish two cases: First, we consider iterations in which many points are reassigned to a different cluster. We call these iterations *dense*. Second, we consider the case that only few points are reassigned. These iterations are called *sparse*.

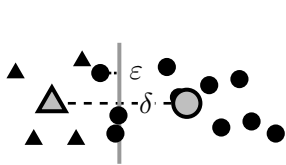
### 3.2 Dense Iterations

An iteration is called dense if there is one cluster that gains or loses at least  $2kd$  points during this iteration. Consider any dense iteration, and let  $X_1$  be the cluster that gains or loses at least  $2kd$  points. Since there are  $k - 1$  other clusters, there must be another cluster with which  $X_1$  exchanges at least  $2d + 1$  points. Let  $X_2$  be this other cluster.

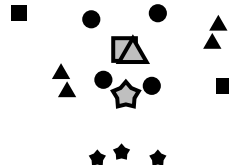
Now we observe the following (see Figure 2(a)): If there is one point  $x$  that is reassigned from  $X_1$  to  $X_2$  (or the other way round) and this point is much closer to  $c_2$  than to  $c_1$ , then reassigning this point to  $X_2$  decreases the potential significantly. More formally: Assume that  $x$  is at a distance of  $\varepsilon$  from the boundary between  $X_1$  and  $X_2$ . Let  $\delta$  be the distance between  $c_1$  and  $c_2$ . Then reassigning  $x$  from  $X_1$  to  $X_2$  decreases the potential by at least  $2\varepsilon\delta$ .

Why do we need  $2d + 1$  points that are exchanged between  $X_1$  and  $X_2$ ? It cannot be ruled out that a single point is almost exactly on the boundary, and just a tiny bit closer to  $c_2$  than to  $c_1$ . But: It is very unlikely that all  $2d + 1$  points are close to the boundary.

Fix any hyperplane  $H$ . The probability that a Gaussian point  $x$  is within a distance of  $\varepsilon$  from  $H$  is bounded by  $\varepsilon/\sigma$ . Thus, the probability that all  $2d + 1$  points are within distance  $\varepsilon$  from  $H$  is  $(\varepsilon/\sigma)^{2d+1}$ . However, the problem is now that  $H$  is not fixed. It is the hyperplane that bisects the clusters  $X_1$  and  $X_2$ . Its position depends on  $c_1$  and  $c_2$ , and  $c_1$  and  $c_2$  depend on the points in  $X_1$  and  $X_2$ . Thus, we cannot simply use the bound  $(\varepsilon/\sigma)^{2d+1}$ , but have to take into account these dependencies. This can be done according to the following lemma.



(a) Dense iteration: Three points want to switch from the right to the left cluster. The top-most point is significantly far away from the boundary. Thus, the potential decreases significantly.



(b) Three sets for the same cluster. Circles plus squares form  $X'_i$ , circles plus triangles form  $X''_i$ . Their centers are close together. Circles plus stars form  $X'''_i$ , and its center is far away from the other centers.

Figure 2: Dense and sparse iterations.

**Lemma 3 (Arthur, Vassilvitskii [2]).** *The probability that there exists a hyperplane  $H$  and  $2d + 1$  points of  $Z$  such that all these points are  $\text{poly}(n^{-k}, \sigma)$ -close to  $H$  is bounded by  $\text{poly}(n^{-kd})$ .*

Now there is no more dependency on  $H$ , as the lemma is simultaneously about all hyperplanes. If there is no hyperplane and set of  $2d + 1$  points that are all  $\text{poly}(n^{-k}, \sigma)$ -close to this hyperplane, then every dense iteration decreases the potential by at least  $\text{poly}(n^{-k}, \sigma)$ .

### 3.3 Sparse Iterations

Sparse iterations are a bit more complicated to handle. In a sparse iteration, any cluster gains or loses at most  $2kd$  data points. In contrast to dense iterations, we cannot guarantee that one of the points that switches to another cluster is significantly far away from the boundary that separates the two clusters. Thus, it might happen that reassigning the data points does not decrease the potential significantly.

However, we have not taken into account the other step of  $k$ -means: adjusting the cluster centers (Step 3). Assume that some cluster center  $c_i$  is at position  $c'_i$  before executing Step 3 and at position  $c''_i = \frac{1}{|X_i|} \cdot \sum_{x \in X_i} x$  afterwards. Then this decreases the potential function by  $\|c'_i - c''_i\|^2 \cdot |X_i|$ .

Now one might want to argue along the following lines: Assume that cluster  $X_i$  changes during an iteration. Let  $X'_i$  be the set of points in cluster  $X_i$  before the iteration and  $X''_i$  be the set of points of  $X_i$  afterwards. If  $X'_i$  and  $X''_i$  do not differ by too many points, then the corresponding centers  $c'_i$  and  $c''_i$  must be at least a certain distance apart (again, this should hold with high probability).

Unfortunately, this cannot be shown. But something a bit weaker holds: It is very unlikely that the centers of three sets  $X'_i, X''_i, X'''_i$  are very close to each other if the sets differ by not too many elements. Figure 2(b) illustrates the situation.

**Lemma 4 (Arthur et al. [1]).** *The probability that there exist sets  $X'_i, X''_i, X'''_i$  of points with centers  $c'_i, c''_i, c'''_i$  such that  $X'_i$  and  $X''_i$  as well as  $X''_i$  and  $X'''_i$  differ by at most  $O(kd)$  points and  $\|c'_i - c''_i\| \leq \text{poly}(n^{-k}, \sigma)$  and  $\|c''_i - c'''_i\| \leq \text{poly}(n^{-k}, \sigma)$  is at most  $\text{poly}(n^{-kd})$ .*

Now we have to adjust our high-level idea from Section 3.1 a bit:

1. We have a dense iteration. We already saw how to deal with this case.
2. We have a sparse iteration. In this case, we consider a sequence of iterations until one of the following two events happens:

- a) We encounter a dense iteration. This is fine as we then jump to case 1.
- b) One of the clusters assumes a third set of points.

Case 2b remains to be analyzed. We have already said that the center of the cluster that assumes three different sets is likely to make a significant move: If there are no three sets as described in Lemma 4, then the move of the center if a cluster assumes a third set decreases the potential by  $\text{poly}(n^{-k}, \sigma)$ .

How long can a sequence of iterations be until one center assumes a third set? The straightforward answer is  $2^k$  iterations: Each of the  $k$  clusters is either in its first or second set (it can jump back and forth between these). This gives  $2^k$  configurations. None of these configurations can repeat. Thus, after  $2^k$  iterations, either  $k$ -means terminates or one cluster assumes a third set. The bound of  $2^k$  suffices here, but it can even be brought down to a constant [1].

### 3.4 Putting Things Together

From Sections 3.2 and 3.3, we obtain the following situation: With high probability, every dense iteration and every sequence of  $2^k$  sparse iterations decreases the potential by at least  $\text{poly}(n^{-k}, \sigma)$ . We combine this with the fact that, with high probability, the potential is initially at most  $\text{poly}(n, 1/\sigma)$ . Thus, the number of iterations is at most  $2^k \cdot \frac{\text{poly}(n, 1/\sigma)}{\text{poly}(n^{-k}, \sigma)} = \text{poly}(n^k, 1/\sigma)$  with high probability. If this does not hold, then we use the worst-case bound of  $\text{poly}(n^{kd})$ . However, the probability that we have to use the worst-case bound is at most  $\text{poly}(n^{-kd})$ , which makes this contribution to the expected running-time negligible. Thus, we get an expected running-time of  $\text{poly}(n^k, 1/\sigma)$ , which proves Theorem 2.

### 3.5 Towards the Polynomial Bound

The above distinction into two different types of iterations is a bit crude and does not suffice to prove Theorem 1. In this section, we will give a very brief idea how Theorem 1 can be proved.

We have to take into account the positive correlations between similar iterations: Assume that we have two similar clusterings, and we perform one step of  $k$ -means on them. Assume further that this yields a significant decrease of the potential with respect to the first one. Then it is likely that also in the second case, the potential decreases significantly. Thus, we divide the possible iterations of  $k$ -means into classes of similar transitions. This is the purpose of a transition blueprint.

Consider the situation in Figure 3(a). For every point that wants to switch to another cluster, we draw an edge between the respective cluster centers. This gives us the *transition graph* shown in Figure 3(b). A *transition blueprint* is a combination of a transition graph together with a rough estimate of where the  $k$  cluster centers are. The idea is as follows: If a transition blueprint describes an iteration of  $k$ -means, but a cluster center is far away from its estimate in this transition, then this cluster center must make a significant move. This decreases the potential. Otherwise, i.e., if all cluster centers are close to their estimates, then we know (at least roughly) where all the centers are. This means that the hyperplanes are (almost) fixed, as they depend only on the positions of the centers. Thus, we can (almost) exploit what we already know from Section 3.2: The probability that a Gaussian point is  $\varepsilon$ -close to a hyperplane is at most  $\varepsilon/\sigma$ . By taking into account the dependencies that are caused by conditioning on the event that all cluster centers are close to their estimates, this can be turned into a proof for Theorem 1.

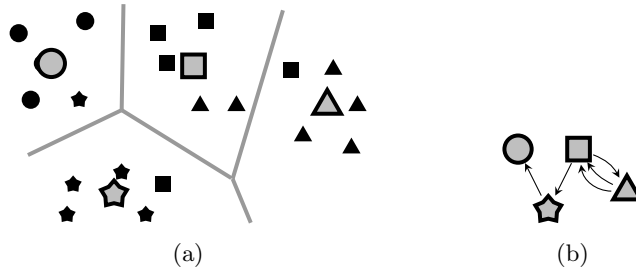


Figure 3: A transition of  $k$ -means (a) and its transition graph (b).

### 3.6 Bregman Divergences

Squared Euclidean distance is the most obvious distance measure for  $k$ -means. (The square guarantees that the center should be the center of mass.) However, for many applications other distance measures are used. One example is the Kullback-Leibler divergence (KLD) or relative entropy: We view  $c$  and  $x$  as probability distributions on the set  $\{1, \dots, d\}$ , i.e.,  $c_i, x_i \geq 0$  for all  $i$  and  $\sum_{i=1}^d c_i = \sum_{i=1}^d x_i = 1$ . KLD measures the expected number of extra bits if we code samples from  $x$  with a code based on  $c$  rather than a code based on  $x$  itself. Thus,  $\delta_{\text{KLD}}(x, c) = \sum_{i=1}^d x_i \cdot \log(x_i/c_i)$ .

KLD and squared Euclidean distance are two examples of Bregman divergences. A Bregman divergence is a distance measure based on the following observation: Consider a strictly convex function  $\Phi : D \rightarrow \mathbb{R}$ . The domain  $D$  can be  $\mathbb{R}^d$  or, as for KLD, the probability simplex. Now we approximate  $\Phi(x)$  from  $c$  by a linear approximation:  $\bar{\Phi}(x) = \Phi(c) + (x - c)^T \nabla \Phi(c)$ . The distance of  $x$  to  $c$  is the error of this approximation:  $\delta_{\Phi}(x, c) = \Phi(x) - \bar{\Phi}(x)$ . Since  $\Phi$  is strictly convex,  $\delta_{\Phi}(x, c) \geq 0$  and  $\delta_{\Phi}(x, c) = 0$  if and only if  $x = c$ . Bregman divergences are exactly the distance measures that can be used for  $k$ -means, as they can be characterized by the property that the center should be chosen as the center of mass [4].

Unfortunately, Gaussian perturbations are not appropriate for every Bregman divergence. For instance, negative values are not allowed for KLD. Since the proof of the polynomial bound heavily relies on properties of Gaussian perturbations, only the following weaker bounds are known for  $k$ -means with nice Bregman divergences. (Many Bregman divergences are nice: squared Euclidean distance, KLD, Mahalanobis distances, Itakura-Saito divergence. . .)

**Theorem 5 (Manthey, Röglin [21]).** *Consider  $k$ -means with a nice Bregman divergence. Then the smoothed number of iteration is bounded by  $\text{poly}(n^{\sqrt{k}}, 1/\sigma)$  and  $k^{kd} \cdot \text{poly}(n, 1/\sigma)$ .*

The second bound is particularly interesting as it yields polynomial smoothed running-time if  $k$  and  $d$  are small compared to  $n$ , which is often the case in practice.

## 4 Concluding Remarks

Let us conclude with two remarks. First, the obvious open question is to prove a polynomial bound for the smoothed running-time of  $k$ -means with Bregman divergences.

Second, the polynomial bound of Theorem 1 is  $O(n^{34} k^{34} d^8 \log^4(n)/\sigma^6)$ , which is quite huge. Often in smoothed analysis, results are more qualitative than quantitative. This is because the combination of a worst-case part (the adversary) and the average-case part (the perturbation) makes the analysis technically very challenging. However, Spielman and Teng's bound for the

simplex algorithm is  $O(n^{86}d^{55}/\sigma^{30})$ . This bound has been improved dramatically to  $O((d^9 + d^3/\sigma^4) \log^7 n)$  by Vershynin [30]. Thus, there is hope that, with new techniques, the smoothed running-time of  $k$ -means can also be improved drastically.

## References

- [1] David Arthur, Bodo Manthey, and Heiko Röglin.  $k$ -means has smoothed polynomial complexity. In *Proc. of the 50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 405–414. IEEE Computer Society, 2009.
- [2] David Arthur and Sergei Vassilvitskii. Worst-case and smoothed analysis of the ICP algorithm, with an application to the  $k$ -means method. *SIAM Journal on Computing*, 39(2):766–782, 2009.
- [3] Cyril Banderier, René Beier, and Kurt Mehlhorn. Smoothed analysis of three combinatorial problems. In Branislav Rován and Peter Vojtás, editors, *Proc. of the 28th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 2747 of *Lecture Notes in Computer Science*, pages 198–207. Springer, 2003.
- [4] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.
- [5] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis of the multilevel feedback algorithm. *Mathematics of Operations Research*, 31(1):85–108, 2006.
- [6] René Beier, Heiko Röglin, and Berthold Vöcking. The smoothed number of Pareto optimal solutions in bicriteria integer optimization. In Matteo Fischetti and David P. Williamson, editors, *Proc. of the 12th Int. Conf. on Integer Programming and Combinatorial Optimization (IPCO)*, volume 4513 of *Lecture Notes in Computer Science*, pages 53–67. Springer, 2007.
- [7] René Beier and Berthold Vöcking. Random knapsack in expected polynomial time. *Journal of Computer and System Sciences*, 69(3):306–329, 2004.
- [8] René Beier and Berthold Vöcking. Typical properties of winners and losers in discrete optimization. *SIAM Journal on Computing*, 35(4):855–881, 2006.
- [9] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002.
- [10] Avrim L. Blum and John D. Dunagan. Smoothed analysis of the perceptron algorithm for linear programming. In *Proc. of the 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 905–914. SIAM, 2002.
- [11] Endre Boros, Khaled Elbassioni, Mahmoud Fouz, Vladimir Gurvich, Kazuhisa Makino, and Bodo Manthey. Stochastic mean payoff games: Smoothed analysis and approximation schemes. Manuscript, 2011.
- [12] Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.
- [13] Valentina Damerow, Bodo Manthey, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler, and Till Tantau. Smoothed analysis of left-to-right maxima with applications. *ACM Transactions on Algorithms*, to appear.

- [14] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, 2000.
- [15] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. In *Proc. of the 18th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1295–1304. SIAM, 2007.
- [16] Mahmoud Fouz, Manfred Kufleitner, Bodo Manthey, and Nima Zeini Jahromi. On smoothed analysis of quicksort and Hoare’s find. *Algorithmica*, to appear.
- [17] Mary Inaba, Naoki Katoh, and Hiroshi Imai. Variance-based  $k$ -clustering algorithms by Voronoi diagrams and randomization. *IEICE Transactions on Information and Systems*, E83-D(6):1199–1206, 2000.
- [18] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [19] Bodo Manthey and Rüdiger Reischuk. Smoothed analysis of binary search trees. *Theoretical Computer Science*, 378(3):292–315, 2007.
- [20] Bodo Manthey and Heiko Röglin. Improved smoothed analysis of  $k$ -means clustering. In *Proc. of the 20th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 461–470. SIAM, 2009.
- [21] Bodo Manthey and Heiko Röglin. Worst-case and smoothed analysis of  $k$ -means clustering with Bregman divergences. In Yingfei Dong, Ding-Zhu Du, and Oscar Ibarra, editors, *Proc. of the 20th Ann. Int. Symp. on Algorithms and Computation (ISAAC)*, volume 5878 of *Lecture Notes in Computer Science*, pages 1024–1033. Springer, 2009.
- [22] Ankur Moitra and Ryan O’Donnell. Pareto optimal solutions for smoothed analysts. arXiv:1011.2249v1 [cs.DS], 2010.
- [23] Heiko Röglin and Shang-Hua Teng. Smoothed analysis of multiobjective optimization. In *Proc. of the 50th Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 681–690. IEEE Computer Society, 2009.
- [24] Heiko Röglin and Berthold Vöcking. Smoothed analysis of integer programming. *Mathematical Programming*, 110(1):21–56, 2007.
- [25] Guido Schäfer and Naveen Sivadasan. Topology matters: Smoothed competitiveness of metrical task systems. *Theoretical Computer Science*, 241(1–3):216–246, 2005.
- [26] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of termination of linear programming algorithms. *Mathematical Programming, Series B*, 97(1–2):375–404, 2003.
- [27] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.
- [28] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis: An attempt to explain the behavior of algorithms in practice. *Communications of the ACM*, 52(10):76–84, 2009.
- [29] Andrea Vattani.  $k$ -means requires exponentially many iterations even in the plane. *Discrete and Computational Geometry*, to appear.
- [30] Roman Vershynin. Beyond Hirsch conjecture: Walks on random polytopes and smoothed complexity of the simplex method. *SIAM Journal on Computing*, 39(2):646–678, 2009.