CS264: Beyond Worst-Case Analysis
Lecture #7: Perturbation Stability and Single-Link++[*]

Tim Roughgarden[†]

October 13, 2014

1 Preamble

This lecture is our second in a series that develops theory to support the idea that "clustering is hard only when it doesn't matter." That is, we generally care about clustering instances in which there is a "meaningful" or "robust" solution, and the existence of such a solution implies structure in the input that can be implicitly or explicitly exploited by an algorithm.

For consistency, we continue to study the k-median problem; much of this lecture applies equally well to other popular clustering objectives, such as k-means (see Homework #4). Recall that an instance of the k-median problem is specified by a n-point metric space (X, d). Recall that d satisfies the triangle inequality, meaning $d(x, y) \leq d(x, z) + d(z, y)$ for every $x, y, z \in X$. The goal is to choose a set S of k "centers" to minimize the sum of the nearest-center distances:

$$\min \sum_{x \in X} \left(\min_{c \in S} d(c, x) \right).$$

We will be somewhat sloppy about distinguishing between "clusters" and "centers." A choice of centers $\{c_1, \ldots, c_k\}$ naturally induces a clustering $C_1, \ldots, C_k$, with C_i denoting the points closer to c_i than any other center. Conversely, given a clustering $C_1, \ldots, C_k$, one can define the center c_i of C_i as the point of C_i that minimizes the total distance to the other points of C_i.[1]

[*]©2014, Tim Roughgarden.

[†]Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

[1]The clustering induced by these centers $c_1, \ldots, c_k$ need not be the original one $C_1, \ldots, C_k$, but the new clustering will have only better objective function value.

2 A Second Stability Notion

Today's stability notion, which is sometimes called *perturbation stability*, shares some of the spirit with that of last lecture. The motivation and technical details are somewhat different, however. First, we assume that the target clustering $C_1, \ldots, C_k$ is the one with optimal k-median objective function value. We call the instance *γ-stable* for $\gamma \geq 1$ if $C_1, \ldots, C_k$ remains optimal under every γ-perturbation, meaning a scaling $\sigma_{xy} d(x, y)$ of each distance $d(x, y)$ by a number $\sigma_{xy} \in [1, \gamma]$.[2] Note that γ-stability becomes more and more stringent as γ increases. Our goal is to prove that, for all γ at least a sufficiently large constant, there is a polynomial-time algorithm that recovers the optimal clustering in every γ-stable instance. (Recall the problem is NP-hard in worst-case instances.)

The motivation behind the γ-stability definition is that, in many clustering applications, the distance function d is heuristic. In some cases, data points really do belong to some normed space and distances can be taken literally. But when the points are objects, proteins, documents, etc., there is no "true" distance function, and various standard distance/dissimilarity measures are used. Solving a clustering problem with such a distance notion implicitly assumes that the optimal solution of the problem is not overly sensitive to small perturbations of the distance function. Put differently, sensitivity of the output to the details of the distance function suggests that the wrong question is being asked. The γ-stability condition is a natural way to make this implicit assumption precise.

3 Why Do We Need Another Stability Definition?

At a high level, this lecture executes exactly the same program as last lecture: we identify an implicit assumption made in common approaches to clustering; formulate this assumption as a condition on k-median instances; identify structural implications of this condition that are absent from worst-case instances; and design new or analyze existing algorithms that perform well in the presence of such structure. Why bother doing all this a second time?

First, as indicated last section, the motivation behind this lecture's stability notion, *perturbation stability*, is different from that of last lecture. Here, the stability definition formalizes the assumption that the optimal solution should be robust with respect to the specification of the distance functions. Last lecture's stability condition, which is sometimes called *approximation stability*, formalized the idea that clusterings with near optimal objective function value should also be structurally similar to the target clustering — structural robustness with respect to perturbations of the objective function value. Thus, the two lectures single out different implicit assumptions underlying common approaches to clustering. Second, there has been quite a bit of work on "recovering stable clusterings" over the past five years, with at least five different stability notions studied in depth (see [2, 6] for surveys). The two conditions we're covering give you a glimpse of the diversity of stability

[2]Such a perturbation might violate the triangle inequality. For example, $d(x, y) = d(y, z) = d(z, x) = 1$ might become $d'(x, y) = d'(y, z) = 1$ and $d'(x, z) = 3$. We require $C_1, \ldots, C_k$ to remain optimal even when the perturbations violate the triangle inequality.

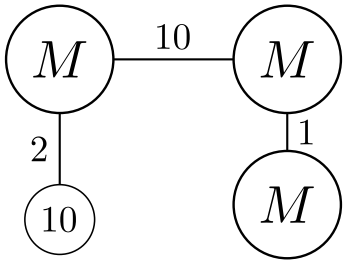

Figure 1: A bad example for single-link when $k = 3$. The circles represent M (or 10) co-located nodes, and the edges are distances. Single-link will cut the "2" and the "10" edge, allocating a single center for the right-hand side.

conditions; see also Homework #4 and Section 8 for further comparisons and discussion. Third, it is interesting to observe that different stability conditions, motivated by different implicit modeling assumptions, wind up implying similar restrictions on clustering instances — that the optimal clustering is, perhaps modulo a few outliers, "a bunch of well-separated tightly-knit clusters." Fourth, in contrast to last lecture, where we used the stability condition to motivate new algorithmic ideas, this lecture's stability definition is used to motivate and justify a nice twist on a widely used clustering algorithm.

4 Single-Link Clustering

Single-link clustering is a simple and widely known clustering algorithm. The idea is to think of the input metric space (X, d) as a complete graph, with vertices X and edge weights given by d. The algorithm runs Kruskal's minimum spanning tree (MST) algorithm,[3] except it stops when there are k connected components, where k is the desired number of clusters. That is, we skip the final $k - 1$ edge additions of Kruskal's algorithm.[4]

The simplest statement that might be true is: for γ sufficiently large, single-link clustering recovers the optimal k-median solution in every γ-stable instance. The example shown in Figure 1 shows that this statement is false. In the example, there are three cities, each with M co-located citizens, and a tiny village, with 10 co-located citizens. If $k = 4$, then it's clear where to locate the centers — one per city, and one in the village. With $k = 3$, the optimal solution is also clear: locate one center per city, with the 10 villagers each traveling 2 units to the nearest center, for an objective function value of 20. Single-link clustering, however, skips the final 2 iterations of Kruskal's algorithm (i.e., the "2" and "10" edges), thus retaining the "1" edge and forcing the location of a single center between the two cities on the right-

[3]Recall this is the algorithm where you sort the input graph's edges from cheapest to most expensive, and then do a single pass through them, including an edge in the tree-so-far if and only if it does not create a cycle.

[4]Each edge addition fuses two connected components into one and hence decreases the number of components by 1; at some point, there are exactly k connected components remaining.

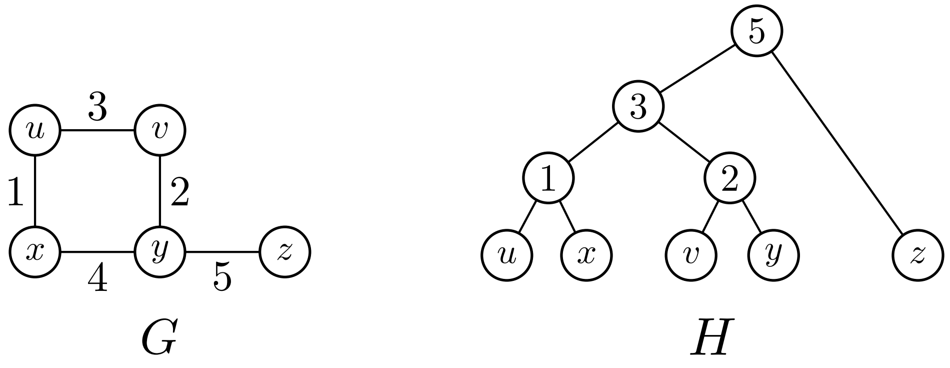

Figure 2: The hierarchical cluster tree. The subtrees of H correspond to the connected components that appear during the execution of Kruskal's algorithm on G.

hand side. The objective function value of this solution is M, arbitrarily larger than the optimal value. Moreover, such instances are γ-stable for arbitrarily large γ (as $M \to \infty$); see Homework #4. This example shows that if we want to recover optimal solutions of γ-stable instances, we need a better algorithm than single-link clustering.

5 Single-Link++

We now describe *single-link++ (SL++)*, a more sophisticated extension of single-link clustering proposed in [4]. The first step of the algorithm is to run Kruskal's MST algorithm until completion. The trajectory of Kruskal's algorithm on a graph G induces a "hierarchical cluster tree" H; see Figure 2. The n leaves of H are precisely the vertices of G. Each of the $n-1$ edges added by Kruskal's algorithm corresponds to an internal node of H; in Figure 2 these nodes are labelled with the cost of the corresponding edge of G. All internal nodes of H have exactly two children. The (sets of leaves of) subtrees of H correspond exactly to the connected components that appear at some point in the execution of Kruskal's algorithm.[5] For example, in Figure 2, the subtree rooted at "1" has the leaf set $\{u, x\}$, and this set is indeed a connected component in Kruskal's algorithm after the first edge addition. Similarly, the subtree rooted at "3" has leaves $\{u, x, v, y\}$ and appears as a connected component in Kruskal's algorithm, prior to the final edge addition.

Next, define a *k-pruning* of H ($k \geq 1$) as a k-clustering induced by the removal of a set S of k internal nodes "from the top" of H, meaning that if $x \in S$ then parent$(x) \in S$ as well. Note that a k-pruning shatters H into k disjoint subtrees (as each internal node of H has two children), which induces a partition of the leaves of H (i.e., the vertices of G) into k clusters. When $k = 1, 2$ there is only one k-pruning, but for $k > 2$ there are many. For

[5]There are $2n-1$ in all. Kruskal's algorithm begins with n different connected components (the isolated vertices); each of the $n-1$ edge additions destroys two of the old components and replaces them with a new one.

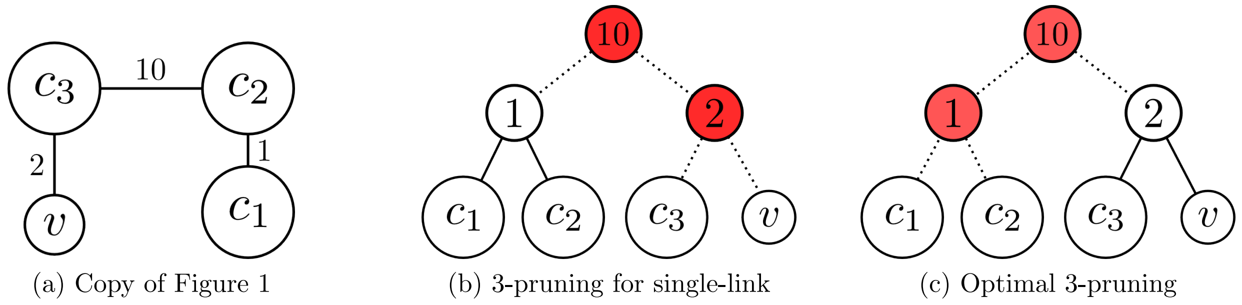

Figure 3: Two 3-prunings of the hierarchical cluster tree arising from the instance in Figure 1. The c_i's represent cities with M co-located nodes, and v is a village with $10 \ll M$ nodes. The optimal 3-pruning is also the optimal clustering for $k = 3$.

example, Figure 3 shows two 3-prunings of the hierarchical cluster tree that corresponds to the bad example of the previous section. Figure 3(b) shows the 3-pruning output by the single-link clustering algorithm, which reverses the most recent (i.e., highest-cost) steps of Kruskal's algorithm, while Figure 3(c) shows that the optimal 3-clustering also arises as a 3-pruning of the hierarchical cluster tree.

The second step of the SL++ algorithm is to compute the optimal k-pruning of H — among all k-clusterings induced by k-prunings, the one with the smallest k-median objective function value (for the best choice of cluster centers). This step can be implemented in polynomial time via dynamic programming; see Homework #4. In the bad example of the previous section, since the optimal solution arises as a 3-pruning, the SL++ algorithm will recover the optimal solution.

The main result of this lecture is the following.

Theorem 5.1 ([3]) *For every γ-stable k-median instance with $\gamma > 3$, the SL++ algorithm recovers the optimal solution.*

The analysis in Theorem 5.1 is tight: for every $\epsilon > 0$, there is a $(3 - \epsilon)$-stable k-median instance such that the SL++ algorithm does not recover the optimal solution (Homework #4). A variation of the algorithm, which uses a different criterion to choose the order in which to merge connected components, recovers the optimal solution in $(1 + \sqrt{2})$-stable instances (Homework #4). On the other hand, for $\epsilon > 0$, no polynomial-time algorithm always recovers the optimal solution in $(2 - \epsilon)$-stable k-median instances, unless $P = NP$ (Homework #4).

6 When Does Single-Link++ Succeed?

This section identifies sufficient conditions on a k-median instance such that the SL++ algorithm recovers the optimal solution. The next section proves that every γ-stable instance with $\gamma > 3$ meets these conditions.

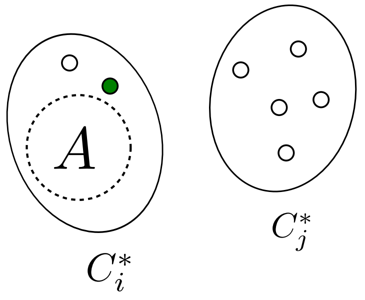

Figure 4: Tightly knit clusters — the closest point to A is always contained in the optimal cluster C_i^* that contains A.

We begin with a condition that characterizes the instances in which SL++ computes the optimal clustering $C_1^*, \ldots, C_k^*$:

(*) for every i, the cluster C_i^* appears as a connected component at some stage of Kruskal's algorithm.

This condition is clearly necessary, since the SL++ algorithm always outputs a k-pruning, every cluster of a k-pruning corresponds to the leaves of a subtree of the hierarchical cluster tree H, and such subtrees correspond to the clusters that appear during the execution of Kruskal's algorithm. Conversely, if all of the optimal clusters are represented as (disjoint) subtrees, then there is a k-pruning of H that induces the optimal clustering, and the SL++ algorithm will find it (since it optimizes over all clusterings induced by k-prunings).

Next, we claim the following property is a sufficient condition for the condition in (*):

(**) For every cluster i, every strict subset $A \subsetneq C_i^*$ of the ith optimal cluster, the point $\operatorname{argmax}_{x \in X \setminus A} \min_{a \in A} d(a, x)$ of $X \setminus A$ closest to A lies in C_i^*.

Intuitively, condition (**) asserts that the optimal solution comprises only "tightly knit clusters;" see Figure 4.

Suppose that an instance satisfies the condition (**). Fix a cluster C_i^*, and consider the first iteration in which Kruskal's algorithm adds an edge between a point a in C_i^* and a point x outside of C_i^*. (Since Kruskal eventually merges everything, such an iteration exists.) Since this is the first such iteration, a's current component A is a subset of C_i^*. By the definition of Kruskal's algorithm, x is the point of $X \setminus A$ closest to A. Since $x \notin C_i^*$, condition (**) implies that A cannot be a strict subset of C_i^* — that is, $A = C_i^*$. This exhibits an iteration of Kruskal's algorithm in which C_i^* appears as a connected component, so we have verified the condition (*).

7 Proof of Theorem 5.1

This section shows that every γ-stable k-median instance with $\gamma > 3$ satisfies the condition (**) of the previous section, and hence the SL++ algorithm recovers the optimal

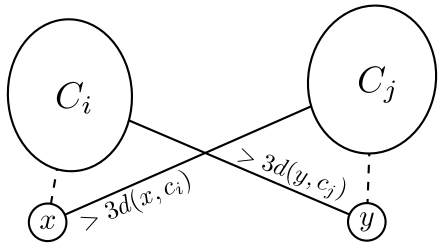

Figure 5: The second step of the proof of Theorem 5.1.

solution in such instances. The proof has three steps, each a little trickier than the previous one.

First, we claim that in every γ-stable instance with $\gamma > 3$, for every point x in an optimal cluster C_i^*,

$$d(x, c_j) > 3d(x, c_i), \tag{1}$$

where c_i and c_j denote the centers of the optimal clusters C_i^* and C_j^*, respectively (with $j \neq i$). That is, in the optimal clustering, every point is much closer to the center of its cluster than to any of the other cluster centers. This is reminiscent of the "well-separated" property from last lecture.

To prove the claim, consider blowing up all distances inside C_i^* by a factor of 3 and leaving all the other distances alone. In particular, $d(x, c_i)$ gets multiplied by 3 while $d(x, c_j)$ stays the same. Since the instance is γ-stable with $\gamma > 3$, the optimal solution stays the same.[6] Since all distances within C_i^* have been scaled by the same factor, c_i remains the optimal choice of a center within C_i^*. Since the solution is optimal, reassigning x from c_i to c_j only makes the solution worse. This means that, even after blowing up the distance $d(x, c_i)$ by a factor of 3, x is still closer to c_i than to c_j, which is exactly the assertion in (1).

The second step of the proof is to show that: for every $x \in C_i^*$ and $y \in C_j^*$ with $i \neq j$,

$$d(x, y) > 2 \max\{d(x, c_i), d(y, c_j)\}. \tag{2}$$

To prove this, consider the picture in Figure 5. By the first step and the triangle inequality, we have

$$d(x, y) + d(x, c_i) > 3d(y, c_j) \tag{3}$$

and

$$d(x, y) + d(y, c_j) > 3d(x, c_i). \tag{4}$$

Suppose that $d(x, c_i) \geq d(y, c_j)$; the other case is symmetric. Inequality (4) simplifies to $d(x, y) > 2d(x, c_i)$, which then implies that $d(x, y) > 2d(y, c_j)$ as well.

For the final step, we verify the sufficient condition (**) for the SL++ algorithm to recover the optimal solution. Choose an optimal cluster C_i^* and a strict subset A of C_i^*. We need to show that the point of $X \setminus A$ closest to A lies in C_i^*.

[6]This is the only point in the entire proof of Theorem 5.1 where we use the stability assumption.

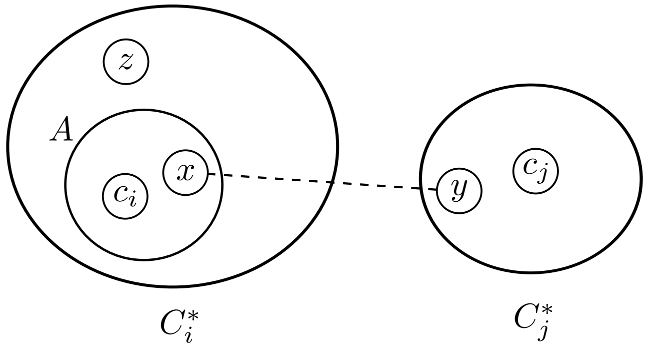

Figure 6: Step 3 of the proof of Theorem 5.1. The dashed line is assumed to be short.

The easy case is when the center c_i is not in A. Then, Step 2 immediately implies that c_i is closer to every point of A than any point outside of C_i^* is to any point of A. This implies that the closest point to A must lie in C_i^*. For the rest of the proof, we assume that $c_i \in A$ (and we will use this fact).

To obtain a contradiction, suppose that the closest point to A is $y \in C_j^*$ with $j \neq i$. Let x denote the closest point $\text{argmin}_{x \in A} d(x,y)$ of A to y. Since A is a strict subset of C_i^*, we can also choose a point $z \in C_i^* \setminus A$. See Figure 6.

The idea for obtaining a contradiction is to consider the four-hop path $z \rightarrow c_i \rightarrow x \rightarrow y \rightarrow c_j$. Roughly speaking, three of these hops involve points and their centers, and so are relatively short. The assumption that y is A's nearest neighbor implies that $d(x,y)$ is also not too big. But z is supposed to be much farther from c_j than from c_i (by Step 1). The following derivation makes this contradiction precise.

Beginning with the triangle inequality, we have

$$\begin{aligned} d(z,c_j) &\leq d(z,c_i) + \underbrace{d(c_i,x)}_{<\frac{1}{2}d(x,y)} + d(x,y) + \underbrace{(y,c_j)}_{<\frac{1}{2}d(x,y)} \\ &< d(z,c_i) + 2d(x,y), \end{aligned}$$

with the second inequality following from (2). Next, since $c_i \in A$ and $z \notin A$, and since x, y are the closest pair of points with one in and one out of A, we have $d(x,y) \leq d(z,c_i)$ and hence

$$d(z,c_j) < 3d(z,c_i),$$

which contradicts (1) and completes the proof of Theorem 5.1.

8 Comparison of the Two Stability Notions

We conclude with some further comments on the differences between approximation stability (Lecture #6) and perturbation stability (this lecture), to complement the discussion of Section 3. Last lecture's definition had an "ϵ" while this lecture did not. This approximation parameter appeared in both the stability assumption and in the algorithmic goal — near-optimal solutions only had to be approximately the same as the optimal one, and the algorithm was only required to return a clustering close to the optimal one. Here, we required the optimal solution to remain unchanged under all perturbations, and insisted on exact recovery of the optimal solution. Of course, one can instantiate approximation stability with $\epsilon = 0$. One can equally well add an approximation parameter to the definition of perturbation stability — the new definition assumes that the optimal solution after a γ-perturbation is ϵ-close to the original optimal solution, and the goal is to recover such an ϵ-close clustering. For a given value of ϵ, perturbation stability is a strictly weaker restriction (Homework #4), so positive results for perturbation-stable instances are stronger than for approximation-stable instances. Today's results can be extended, albeit with extra algorithmic and analytic work and with worse constants, to such "(γ, ϵ)-stable" k-median instances under the assumption that every optimal cluster is sufficiently big (similar to last lecture) [5].

At a high level, the two stability notions share much of the same spirit, and turn out to impose similar structure on the optimal solution — roughly, that it comprises well-separated and tightly knit clusters, perhaps modulo a few outliers. This is encouraging: in the absence of any one truly convincing model of "clustering instances with a meaningful solution," a good goal is to show that different models support the same conclusion. When multiple reasonable if imperfect models lead to similar results, it increases one's confidence in these conclusions. Thus, the two models and results of this and the previous lecture begin to build evidence that, as hoped, clustering instances that are relevant in applications seem to possess structure that render them more tractable that worst-case clustering instances. See also [1, 7, 8, 9] for still more models and results that support this conclusion.

References

[1] M. Ackerman and S. Ben-David. Clusterability: A theoretical study. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1–8, 2009.

[2] P. Awasthi and M.-F. Balcan. Center based clustering: A foundational perspective. In C. M. Henni, M. Meila, F. Murtagh, and R. Rocci, editors, *Handbook of Cluster Analysis*. CRC Press, 2015.

[3] P. Awasthi, A. Blum, and O. Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1-2):49–54, 2012.

[4] M.-F. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 671–680, 2008.

[5] M.-F. Balcan and Y. Liang. Clustering under perturbation resilience. In *Proceedings of the 39th Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 63–74, 2012.

[6] S. Ben-David. Computational feasibility of clustering under clusterability assumptions. Manuscript, 2014.

[7] Y. Bilu, A. Daniely, N. Linial, and M. Saks. On the practically interesting instances of MAXCUT. In *Proceedings of the 30th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 526–537, 2013.

[8] Y. Bilu and N. Linial. Are stable instances easy? *Combinatorics, Probability & Computing*, 21(5):643–660, 2012.

[9] A. Daniely, N. Linial, and M. Saks. Clustering is difficult only when it does not matter. `http://arxiv.org/abs/1205.4891`, 2012.