CS369N: Beyond Worst-Case Analysis
Lecture #3: Deterministic Planted Models for Clustering and Graph Partitioning[*]

Tim Roughgarden[†]

September 14, 2010

1 Preamble

Last lecture motivated the need for a model of data to gain insight into the relative merits of different online paging algorithms. In this lecture and the next, we explore models of data for clustering and graph partitioning problems. We cover deterministic data models in this lecture, and probabilistic ones in the next.

In some optimization problems, the objective function can be taken quite literally. If one wants to maximize profit or accomplish some goal at minimum cost, then the goal translates directly into a numerical objective function.

In other applications, an objective function is only means to an end. Consider, for example, the problem of *clustering*. Given a set of data points, the goal is to cluster them into "coherent groups", with points in the same group being "similar" and those in different groups being "dissimilar". There is not an obvious unique way to translate this goal into a numerical objective function, and as a result many different objective functions have been studied (k-means, k-median, k-center, etc.) with the intent of making the fuzzy notion of a "good/meaningful clustering" into a concrete optimization problem. In this case, we do not care about an "optimal solution" per se; rather, we want to uncover interesting structure in the data. So we're perfectly happy to compute a "meaningful clustering" with suboptimal objective function value; and highly dissatisfied with an "optimal solution" that fails to indicate any patterns in the data (which suggests that we were asking the wrong question or expecting structure where none exists).

The point is that *if we are trying to cluster a data set, then we are implicitly assuming*

[*]©2009–2010, Tim Roughgarden. Thanks to Sanjeev Khanna for helpful comments on an earlier draft.

[†]Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

that interesting structure exists in the data.[1] This perspective suggest that an explicit model of data could sharpen the insights provided by a traditional worst-case analysis framework (cf., modeling locality of reference in online paging). We explore several methods of making this idea rigorous in this and the next lecture.

2 Stable Instances of Max Cut

The first half of this lecture is about a subset of the results in Bilu and Linial [3], on the Max Cut problem. Here, the input is an undirected graph $G = (V, E)$ with nonnegative weights w_e on the edges, and the goal is to compute a cut (i.e., a partition $(S, \bar{S})$ of V into two non-empty parts) that minimizes the total weight of the cut edges:

$$\sum_{e\,:\,v \in S, w \in \bar{S}} w_e.$$

For example, there is a cut that cut *all* of the edges if and only if the graph is bipartite (by definition).

This is a fundamental graph optimization problem, and it is NP-hard, even to approximate to within a sufficiently small approximation factor [6, 7].[2] But this hardness result is for worst-case instances — what about for "interesting instances", where there is a "clearly optimal" (read: meaningful) solution?

2.1 The Model and the Main Result

Bilu and Linial propose the following definition of Max Cut instances with "clearly optimal" solutions.

Definition 2.1 (γ-Stable Instances [3]) A weighted Max Cut instance (V, E, w) is γ-*stable* if the optimal solution $(S, \bar{S})$ is invariant under all scalings $\sigma_e \in [1, \gamma]$ of edge weights — so $(S, \bar{S})$ remains optimal for $G = (V, E, \sigma w)$ for every such $\{\sigma_e\}_{e \in E}$.

Again, the point of this definition is to formulate the idea that the optimal solution is "isolated" — no other cuts come close to encroaching on its reign of optimality.

While γ-stability asserts that an optimal solution $(S, \bar{S})$ survives all edge weight perturbations within a certain range, it is clear which perturbation is the most threatening: if $(S, \bar{S})$ is still optimal after scaling the weights of all uncut edges by γ (and leaving the weights of cut edges the same), then it is still optimal after all allowable perturbations.

Our main result in this section is the following.

[1]There are exceptions of course. For example, if one is partitioning a graph as part of a divide and conquer algorithm, then the partition is only means to an end and is not interesting in its own right.

[2]On the positive side, if you don't know the beautiful algorithm and analysis of Goemans and Williamson [4], based on semidefinite programming, that achieves a .878-approximation for Max Cut in the worst case, then go learn it ASAP.

Theorem 2.2 *The* Max Cut *problem is solvable in polynomial time in* $(\sqrt{n\Delta}+1)$*-stable instances, where* Δ *denotes the maximum degree.*

Note that the value of γ in Theorem 2.2 is quite large. Improving it (possibly via an algorithm different from that below) is a nice open question. For all we know, the smallest γ necessary for polynomial-time recoverability of the optimal solution could even be a constant.

2.2 The BL Algorithm

We analyze the following greedy algorithm for Max Cut, which bears a vague resemblance to Kruskal's minimum spanning tree algorithm.

Input: a γ-stable Max Cut instance (V, E, w).
Invariant #1: every edge of T is cut in the optimal solution to (V, E, w).
Invariant #2: the subgraph (V, T) is bipartite.

1. Initialize $T = \emptyset$.
2. While (V, T) is not connected:
   (a) Let $C_1 = (A_1, B_1), \ldots, C_k = (A_k, B_k)$ be the (bipartite) connected components of (V, T).
   (b) Let C_ℓ have the fewest number of vertices.
   (c) For each $i \neq \ell$, consider the subgraph of G induced by $C_i \cup C_\ell$ (see Figure 2). Let Y_i denote the "blue edges" (edges of E with endpoints in both A_i and A_ℓ or in both B_i and B_ℓ) and Z_i the "red edges" (edges of E with endpoints in both A_i and B_ℓ or in both A_ℓ and B_i). Let W_i denote the larger of $\{\text{total weight of } Y_i, \text{total weight of } Z_i\}$.
   (d) Let j maximize W_i over connected components C_i with $i \neq \ell$. Merge C_j, C_ℓ by adding either the blue edges Y_i or the red edges Z_i to T, whichever set has larger total weight.
3. Return the bipartition of (V, T).

Figure 1: The Bilu-Linial (BL) algorithm.

To understand the idea behind the algorithm, suppose that it does indeed maintain the two claimed invariants. Then, it terminates with a subset T of edges such that (V, T) is connected and bipartite. There is then a unique cut $(S, \bar{S})$ of (V, T) (up to the naming of the two sides) that cuts all of the edges of T. By the first invariant, $(S, \bar{S})$ must be the maximum cut of the original graph (V, E, w).

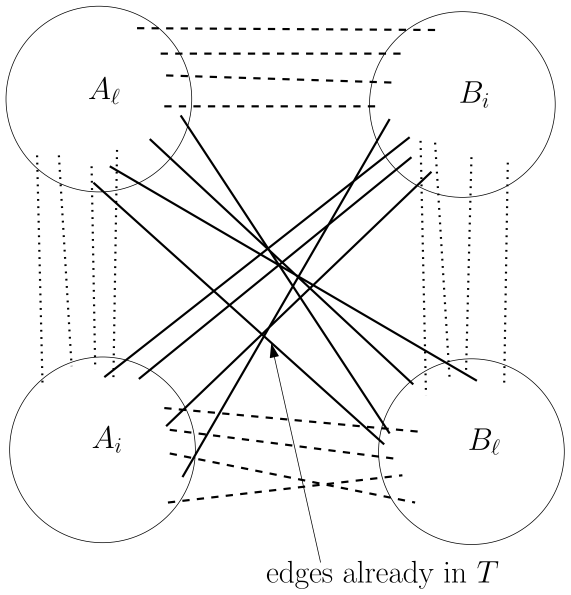


Figure 2: Step 2(c) of the BL algorithm. Solid edges denote those in T; the dotted and dashed edges indicate the blue and red edges, respectively.

The picture to have in your head in this: by assumption, there is a "planted" maximum cut $(S, \bar{S})$. The algorithm does not, of course, know what it is. The plan is to identify edges (namely T) that we are sure cross this (unknown) cut, and to avoid including in T any edges that lie within S or within $\bar{S}$. If we can keep doing this until (V, T) is connected, then we are done. The hope is that the edges in the (stable) maximum cut are conspicuous enough (for sufficiently large γ) that we can greedily identify some of them in every iteration.

2.3 Analysis of the BL Algorithm

We first note that, by induction, the second invariant is maintained (i.e., (V, T) is always bipartite). This holds because when the bipartite connected components (A_j, B_j) and (A_ℓ, B_ℓ) are merged in Step 2d the result is either the bipartite component $(A_j \cup A_\ell, B_j \cup B_\ell)$ (if the red edges are added) or the bipartite component $(A_j \cup B_\ell, A_\ell \cup B_j)$ (if the blue edges are added).

The harder part is showing that the first invariant is maintained provided γ is sufficiently large. We again proceed by induction. We show the following statement, which implies Invariant #1: for each connected component $C_i = (A_i, B_i)$, each of A_i, B_i lies entirely within opposite sides of the planted maximum cut $(S, \bar{S})$. Suppose this invariant holds up to the current iteration, and that components C_j and C_ℓ are chosen in Step 2d of this iteration. Then either the blue edges or the red edges can be added to T without destroying the invariant (corresponding to the cases where A_j, B_ℓ are on the same side of $(S, \bar{S})$ and where A_j, A_ℓ are on the same side of $(S, \bar{S})$, respectively). We need to show that the BL algorithm

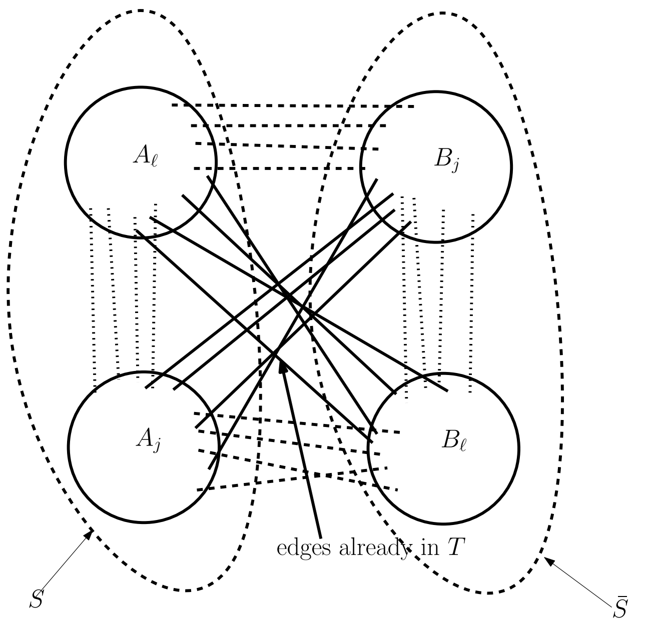


Figure 3: Analysis of the BL algorithm. $(S, \bar{S})$ denotes the optimal cut. Solid edges denote those in T; the dotted and dashed edges indicate the blue and red edges, respectively.

makes the correct choice, for sufficiently large γ.

As befits a greedy heuristic, we proceed via an exchange argument. Let $C_j = (A_j, B_j)$ and $C_\ell = (A_\ell, B_\ell)$ be the components considered in Step 2d. By symmetry, we can assume that A_j, A_ℓ are contained entirely in the set S of the maximum cut $(S, \bar{S})$, while B_j, B_ℓ are contained entirely in $\bar{S}$ (Figure 3). We need to show that the BL algorithm adds the red edges to T — i.e., that the weight of these edges is larger than that of the blue edges.

We now invoke the γ-stability hypothesis: the maximum cut $(S, \bar{S})$ remains optimal even after scaling the blue edges between C_j and C_ℓ by a γ factor. Thus swapping A_ℓ for B_ℓ — the "exchange" in this exchange argument — yields a worse cut $((S \setminus A_\ell) \cup B_\ell, (\bar{S} \setminus B_\ell) \cup A_\ell)$, even after we have scaled all of the blue edges. We next use this fact to lower bound the weight of the red edges in terms of that of the blue edges (and γ).

What is the effect on the cut value of swapping A_ℓ and B_ℓ? First, none of the blue edges were cut by $(S, \bar{S})$, but all are cut after the swap. Thus the weight of the newly cut edges is at least $\gamma \cdot \text{wt(blue)}$, where wt(blue) denotes the total weight of the blue edges in the original (pre-scaled) instance (V, E, w). Next we upper bound the weight of edges that are cut in $(S, \bar{S})$ but not cut after the swap. All of the red edges certainly fall into this category. In addition, consider some other connected component $C_i = (A_i, B_i)$ with $i \neq j, \ell$. Swapping A_ℓ and B_ℓ causes all of the red edges or all of the blue edges (but not both) between C_i and C_ℓ to no longer be cut. Recall that W_i was defined in Step 2c as the larger of the two corresponding total weights. This is at most W_j by our choice of j. Summarizing, an upper bound on the total weight of the edges cut in $(S, \bar{S})$ but not after the swap is

$$\mathrm{wt}(\mathrm{red}) + W_j \cdot N_\ell,$$

where N_ℓ is the number of connected components C_i adjacent to C_ℓ. (If there is no edge with endpoints in C_i and C_ℓ, then we can safely ignore C_i in our cost-benefit analysis of the exchange.)

To upper bound N_ℓ, suppose that C_ℓ has t vertices. Since C_ℓ was chosen as the smallest connected component of (V, T), there are at most n/t such components. Also, since G has maximum degree Δ, at most $t\Delta$ other components can be adjacent to C_ℓ. Since N_ℓ is upper bounded by n/t and by $t\Delta$, it is also upper bounded by the square root of their product: $\sqrt{n\Delta}$.

Recall that swapping A_ℓ and B_ℓ can only yield a cut worse that $(S, \bar{S})$, even after scaling the blue edges by γ. Thus,

$$\gamma \cdot \mathrm{wt}(\mathrm{blue}) < \mathrm{wt}(\mathrm{red}) + W_j \cdot \sqrt{n\Delta}.$$

Since $W_j = \max\{\mathrm{wt}(\mathrm{blue}), \mathrm{wt}(\mathrm{red})\}$ and $\gamma = \sqrt{n\Delta} + 1$, we must have $\mathrm{wt}(\mathrm{red}) > \mathrm{wt}(\mathrm{blue})$. Thus the BL algorithm adds the red edges in Step 2d and maintains the first invariant; this concludes the proof of correctness.

3 Isolated Instances of k-Median

3.1 Definitions and Motivation

This section continues the theme of "planted" or "isolated" target solutions in clustering problems, though for a different problem and with a different definition of "clearly optimal." The model and results are from Balcan, Blum, and Gupta [2].

We study the *k-median* problem, defined as follows. The input is an n-point metric space (X, d), meaning that d is a nonnegative function on $X \times X$ satisfying $d(x, x) = 0$ for all $x \in X$, $d(x, y) = d(y, x)$ for all $x, y \in X$ (symmetry), and $d(x, y) \leq d(x, z) + d(z, y)$ for all $x, y, z \in X$ (triangle inequality). A *k-clustering* of a finite metric space is a partition of its points into k non-empty sets. The goal in the k-median problem is to identify a subset $S \subseteq X$ of k *centers* to minimize the sum of the points' distances to their respective nearest centers:

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

Note that a choice of centers induces a k-clustering (one cluster per center, the points for which that center is the closest). Around 10 years ago Guha and Khuller [5] proved that it is NP-hard to approximate the k-median problem to within a factor strictly smaller that $1 + \frac{2}{e} \approx 1.736$. The best upper bound known is 3 [1].

The key definition is this.

Definition 3.1 ((c, ϵ)-Isolated [2]) The optimal k-median solution is (c, ϵ)-*isolated* if every c-approximate k-clustering agrees with the optimal one on more than a $(1 - \epsilon)$ fraction of the points.

Definition 3.1 says that *the only way to achieve a near-optimal k-median solution is to agree with the optimal solution on almost all points.* Thus numerical similarity (in objective function value) implies structural similarity (in the classification of points). The motivation for Definition 3.1 is the following. Why might a 3-approximation algorithm (say) for a clustering problem be interesting? Such a worst-case guarantee is both technically impressive and robust in a rigorous sense, but 200% error is obviously not acceptable in all applications. An obvious (and good) answer is that one would hope for or even expect much better approximation factors on "most" or on "real" inputs, for many of the ways that we might define these terms. For example: *if the optimal solution is more than a factor of 3 better than every other feasible solution*, then a 3-approximation algorithm is guaranteed to recover the optimal (target) clustering. Since small perturbations of an optimal clustering will typically have near-optimal objective function value, a more general statement is more useful: *if the optimal solution is more than a factor of 3 better than every other "significantly different" feasible solution*, then a 3-approximation algorithm is guaranteed to "almost recover" the optimal (target) clustering. Definition 3.1 makes precise the types of instances for which this conclusion is valid.

Some further comments on Definition 3.1. First, when we speak of two (unordered) k-clusterings "agreeing" on more than a $(1 - \epsilon)$ fraction of the points, we mean that there is a way to label each of the two collections of k clusters with the names $1, 2 \ldots, k$ so that more than $(1 - \epsilon)n$ points of X get the same label in both k-clusterings. (That is, we take an optimal correspondence of the clusters in the two k-clusterings.) Second, Definition 3.1 implicitly assumes that the target clustering is the one with the optimal k-median objective function value. The more general definition in [2] allows the target clustering to have a sub-optimal objective function value (though it must be close to optimal for the isolation property to hold, as you should verify).[3] Finally, compare and contrast Definitions 2.1 and 3.1. The notion of "isolation" is somewhat different in the two cases — the former looks at "nearby instances", the latter at "nearby solutions" to a single instance. The goal in the last section was to recover the optimal solution, but we accomplished this only for relatively large values of the isolation parameter γ. Here, we identify a clustering this is only close to the target one, but we can handle isolation parameters arbitrarily close to 1 (see Theorem 3.2 below).

The inapproximability factor of $1 + \frac{2}{e}$ for general k-median instances carries over to (c, ϵ)-isolated instances. The proof idea is to reduce from the standard k-median problem by adding lots (like n/ϵ) "dummy points" that are all very far from each other and from the original points, and to increase k by the same amount. In the ensuing instance, every reasonable solution makes each dummy point a center (this contributes nothing to the objective function) and then effectively has to solve the original problem for the original points. Further details are left to the reader.

[3]The keen reader might want to rework the proofs in this lecture to apply to this more general setup.

In (c, ϵ)-isolated instances, approximating the k-median objective to within a factor of c guarantees the correct classification of all but an ϵ fraction of the points. What about the converse? If all we care about is correctly classifying most points, do we necessarily have to approximate the k-median objective? The main take-away point of Balcan et al. [2] is a negative answer: in isolated k-median instances, *accurate classification is strictly easier than approximation.*

Theorem 3.2 ([2]) *For every pair α, ϵ of constants, there is a polynomial-time algorithm that, for every $(1+\alpha, \epsilon)$-isolated k-median instance, recovers a k-clustering that agrees with the optimal one on all but an $O(\epsilon/\alpha)$ fraction of the points.*

Note that when $\alpha < 2/e$, Theorem 3.2 guarantees nearly correct classification even when numerical approximation is NP-hard.[4] At a high level, this is possible because the isolation property — which cannot be directly checked by an efficient algorithm — implies the presence of detectable structure this is sufficient for near-optimal classification. We now make this precise.

3.2 Preliminaries

We show a variant of Theorem 3.2 which is easier to prove. The result itself will be incomparable to Theorem 3.2: it requires an additional non-trivial assumption about the input (see below) but correctly classify all but an ϵ fraction of the points (rather than an $O(\epsilon/\alpha)$ fraction).

Consider a k-median instance with a $(1+\alpha, \epsilon)$-isolated optimal solution, where $\alpha, \epsilon > 0$ are constants. For simplicity, assume that we know the objective function value OPT of the optimal k-clustering $C_1^*, \ldots, C_k^*$. (We don't know the C_i^*'s themselves, of course.) As an exercise, the reader should verify that the following algorithm and analysis can be adapted to solve the real problem (where OPT is unknown), by using successively doubling guesses for the value of OPT.

Definition 3.3 For $x \in X$, let $w(x)$ denote x's contribution $\min_{c \in S^*} d(c, x)$ to the optimal solution (with centers S^*). Note that the average value of $w(x)$ is precisely OPT/n.

1. The point x is *close* if
$$w(x) \leq \frac{OPT}{n} \cdot \frac{\alpha}{5\epsilon}. \tag{1}$$

2. The point x is *well separated* if
$$w_2(x) - w(x) > \frac{OPT}{n} \cdot \frac{\alpha}{\epsilon}, \tag{2}$$

   where $w_2(x)$ denotes the distance between X and its second-closest center in the optimal solution S^*.

[4]It turns out that ϵ, α need to be strictly positive to enable the polynomial-time recovery of a good clustering, though this is not obvious [2].

3. The point x is *good* if it is close and well separated, and *bad* otherwise.

Note that Definition 3.3 is just for the sake of analysis; an algorithm (which is ignorant of the optimal solution) has no idea which points are good and which points are bad.

An averaging argument (i.e., Markov's inequality) implies that a constant fraction of all points are close, whether the instance is isolated or not.

Lemma 3.4 *For every k-median instance, all but $\frac{5\epsilon}{\alpha}n$ points are close.*

Using the isolation hypothesis, we can also limit the number of points that are not well separated.

Lemma 3.5 *Less than ϵn points are not well separated.*

Proof: If not, start with the optimal clustering and consider reassigning ϵn points that are not well separated to their second-closest centers. This yields a clustering that misclassifies at most an ϵ fraction of the points, and that has objective function value at most

$$OPT + \epsilon n \frac{OPT}{n} \cdot \frac{\alpha}{\epsilon} = (1+\alpha) \cdot OPT.$$

This contradicts the assumption that the instance is $(1+\alpha, \epsilon)$-isolated. ■

Combining Lemmas 3.4 and 3.5, we find that there are at most

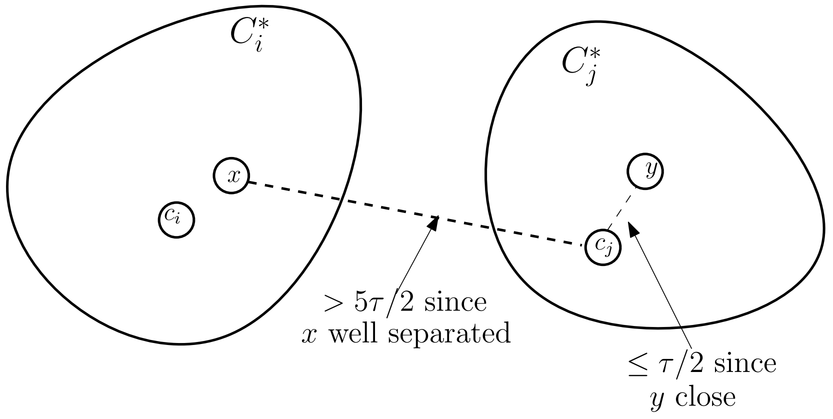
$$b := \epsilon n \left(1 + \frac{5}{\alpha}\right) \tag{3}$$

bad points.

Non-trivial assumption: For the rest of this lecture, we assume that every cluster C_i^* in the optimal clustering contains at least $2b+2$ points, and hence at least $b+2$ good points. This "large optimal clusters" assumption is strong enough that the k-median problem becomes polynomial-time solvable — it implies that $k = O(1/\epsilon)$ and hence an optimal solution can be computed in polynomial time via brute-force search. In the next section we give a much more efficient algorithm for this case. Balcan et al. [2] use a few extra ideas to also handle the case where some optimal clusters are small, which proves Theorem 3.2 without any assumptions (see the paper for details).

3.3 The BBG Algorithm and Analysis

We now give an intertwined presentation of the algorithm and analysis of [2] for the "large optimal cluster" version of Theorem 3.2. The high-level idea is to form a graph on the points X based on pairwise distances, and to run some efficient algorithms on this graph to illuminate easily detectable signatures of the hidden optimal clusters $C_1^*, \ldots, C_k^*$.



C_i^*
C_j^*
x
c_i
y
c_j
$> 5\tau/2$ since x well separated
$\leq \tau/2$ since y close

Figure 4: Observation #2 in the analysis of the BBG algorithm.

Step 1 of the BBG Algorithm: Set

$$\tau := \frac{OPT}{n} \cdot \frac{2\alpha}{5\epsilon},$$

which is twice the right-hand side of (1) and 2/5 times the right-hand side of (2). Form the graph $G = (X, E)$, where the edge (x, y) is in E if and only if $d(x, y) \leq \tau$. (The algorithm is in a position to construct G, provided it knows OPT as discussed earlier.)

Observation #1: *all good points within a single C_i^* form a clique in G.* The reason is that, by (1), all such points are within $\tau/2$ of the center of C_i^*, and so by the triangle inequality they are all within distance τ of each other.

Observation #2: *if x, y are both good and are in different optimal clusters C_i^* and C_j^*, then there are no 1-hop or 2-hop x-y paths in G.* To see this, refer to Figure 4. Since x is well separated, its distance from the center c_j of C_j^* is more than $\frac{5}{2}\tau$. Since y is close, its distance to c_j is at most $\tau/2$. The triangle inequality implies that $d(x, y) > 2\tau$, and hence at least 3 hops in G are needed to get between x and y.

Figure 5 summarizes what the graph G looks like in light of these two observations — a clique for each cluster C_i^*, with different cliques connected only by relatively long paths. Unfortunately, this is not immediately useful because finding a large clique in a graph is an NP-hard problem. The next step of the algorithm filters the edges of G to highlight better the cliques.

Step 2 of the BBG Algorithm: Form the graph $H = (X, F)$, where $(x, y) \in F$ if and only if $(x, y) \in E$ and x, y have at least b common neighbors in G (recall (3)).

Our third observation is immediate from the definitions.

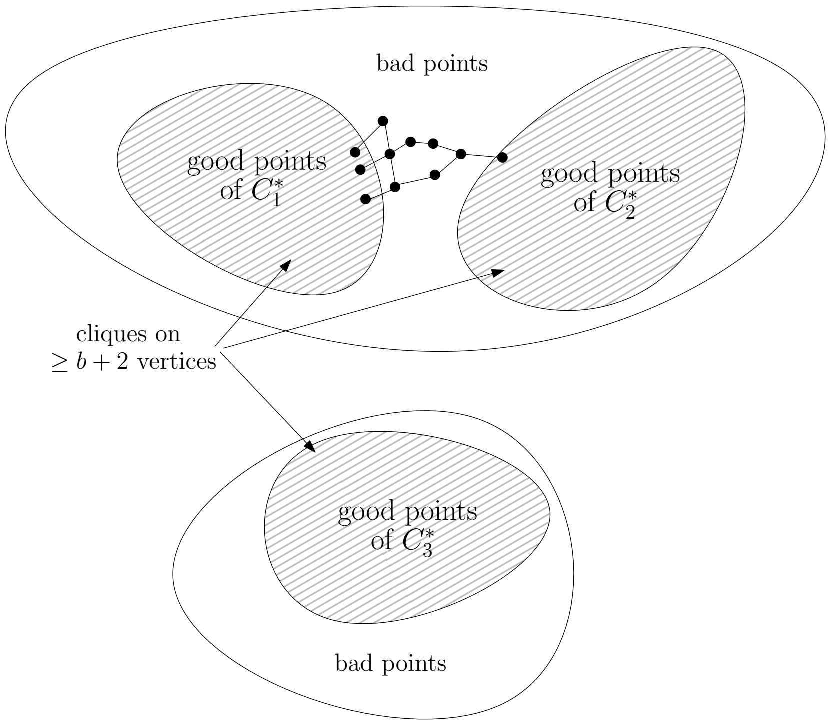


Figure 5: The graph G after the first step of the BBG algorithm.

Observation #3: *the cliques of good points in G (each on at least $b+2$ nodes) survive in H.*

The fourth observation follows easily from the second one.

Observation #4: *the neighbors of a bad point in G consist of other bad points (at most $b-1$ of them) and possibly some good points from a single cluster C_i^*.*

Thus, two bad points are adjacent in H only if each is adjacent in G to good points from the same cluster C_i^*, and necessarily to no good points from the other clusters. The second observation then implies that every clique of good points lies in a different connected component of H.

Thus, the graph H has precisely k connected components that have size at least $b+2$, and each of these contains all of the good points from some C_i^*. These k components can be identified in linear time, and small components (of bad points) can be merged in arbitrarily to get a k-clustering $(\hat{C}_1, \ldots, \hat{C}_k)$. This k-clustering correctly classifies all good points, which constitute at least a $1 - \epsilon(1 + \frac{5}{\alpha})$ fraction of all of the points.

Step 3 of the BBG Algorithm. After Step 2, we have correctly classified all but an $O(\epsilon/\alpha)$ fraction of the points (all but the bad points). This is the guarantee asserted in the original statement of Theorem 3.2, without the "big optimal clusters" assumption. We now show that under the latter assumption, we can correctly classify all but the ϵ fraction of the points that are not well separated.[5] Thus the goal is correctly classify points that are well separated but that might not be close. The idea is to use a clever form of majority vote.

Precisely, we execute the following procedure in parallel (i.e., independently) for each $x \in X$:

1. For each cluster $\widehat{C}_i$ computed in Step 2, look at the *median* distance $d(x, y)$ among points $y \in \widehat{C}_i$ (not including x itself, in case it is already in $\widehat{C}_i$).

2. Re-assign x to the cluster to which its median distance is the smallest.

Every Well Separated Point Is Correctly Re-assigned: Suppose x rightfully belongs to the optimal cluster C_i^*. (It may or may not be there already.) Then for all of the $\geq (b+1)$ good points of C_i^* that are different from x (which are all also members of $\widehat{C}_i$), the triangle inequality implies that

$$d(x, y) \leq \underbrace{d(x, c_i)}_{\text{(where } c_i \text{ is the center of } C_i^*)} + \underbrace{d(c_i, y)}_{\leq \tau/2 \text{ since } y \text{ is close}} \leq d(x, c_i) + \frac{\tau}{2}. \tag{4}$$

Since at most b points of $\widehat{C}_i$ are not good, the median distance $d(x, y)$ with $y \in \widehat{C}_i \setminus \{x\}$ also satisfies (4).

For some other cluster $\widehat{C}_j$, and for all of the good points y in C_j^* (and $\widehat{C}_j$) different than x, the triangle inequality implies that

$$d(x, y) \geq \underbrace{d(x, c_j)}_{\geq d(x,c_i)+5\tau/2 \text{ since } x \text{ well separated}} - \underbrace{d(c_i, y)}_{\leq \tau/2 \text{ since } y \text{ is close}} \geq d(x, c_i) + 2\tau. \tag{5}$$

Again, the median distance $d(x, y)$ with $y \in \widehat{C}_i \setminus \{x\}$ must also satisfy this inequality. Comparing (4) and (5), we see that every well separated point x is inevitably re-assigned to the correct cluster.

4 Take-Away Points

We conclude with two comments that apply to both of the papers studied in this lecture.

First, in both works, the theme is to begin with an uncheckable assumption about an "isolated" optimal solution, and to then deduce from this assumption detectable structure in the problem instance — such as a heavy-weight collection of edges between two subsets of nodes, or a large and dense connected component in a graph defined from pairwise distances. Relatively straightforward algorithmic techniques can then be used to recover the planted

[5]Previous uses of the "big optimal clusters" assumption can be essentially removed with some further work. However, it is not clear how to implement this third step without this assumption.

solution, or something close to it. We will see further examples of this algorithmic and analysis approach in the next lecture.

Second, both of the algorithms that we studied are reasonably natural, and one can imagine extracting from them useful heuristics for finding good cuts and clusterings. On the other hand, the algorithms do feel a bit tailored to the underlying data model. Pushing toward less specific data models is an important goal, and should encourage the design of more robust heuristics. The next lecture offers further progress along these lines.

References

[1] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004. Preliminary version in *STOC '01*.

[2] M.-F. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1068–1077, 2009.

[3] Y. Bilu and N. Linial. Are stable instances easy? In *Proceedings of the First Symposium on Innovations in Computer Science*, pages ??–??, 2010.

[4] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995. Preliminary version in *STOC '94*.

[5] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999. Preliminary version in *SODA '98*.

[6] J. Høastad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001. Preliminary version in *STOC '97*.

[7] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29(6):2074–2097, 2000. Preliminary version in *FOCS '96*.