Stat260/CS294: Spectral Graph Methods **Lecture 5 - 02/05/2015**

Lecture: Overview of Graph Partitioning

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

5 Overview of graph partitioning

The problem of *graph partitioning* or *graph clustering* refers to a general class of problems that deals with the following task: given a graph $G = (V, E)$, group the vertices of a graph into groups or clusters or communities. (One might be interested in cases where this graph is weighted, directed, etc., but for now let's consider non-directed, possibly weighted, graphs. Dealing with weighted graphs is straightforward, but extensions to directed graphs are more problematic.) The graphs might be given or constructed, and there may or may not be extra information on the nodes/edges that are available, but insofar as the black box algorithm that actually does the graph partitioning is concerned, all there is is the information in the graph, i.e., the nodes and edges or weighted edges. Thus, the graph partitioning algorithm takes into account the node and edge properties, and thus it typically relies on some sort of "edge counting" metric to optimize. Typically, the goal is to group nodes in such a manner that nodes within a cluster are more similar to each other than to nodes in different clusters, *e.g.*, more and/or better edges within clusters and relatively few edges between clusters.

5.1 Some general comments

Two immediate questions arise.

- A first question is to settle on an objective that captures this bicriteria. There are several ways to quantify this bicriteria which we will describe, but each tries to cut a data graph into 2 or more "good" or "nice" pieces.
- A second question to address is how to compute the optimal solution to that objective. In some cases, it is "easy," e.g., it is computable in low-degree polynomial time, while in other cases it is "hard," e.g., it is intractable in the sense that the corresponding decision problem is NP-hard or NP-complete.

In the case of an intractable objective, people are often interested in computing some sort of approximate solution to optimize the objective that has been decided upon. Alternatively, people may run a procedure without a well-defined objective stated and decided upon beforehand, and in some cases this procedure returns answers that are useful. Moreover, the procedures often bear some sort of resemblance to the steps of algorithms that solve well-defined objectives exactly. Clearly, there is potential interest in understanding the relationship between these two complementary

approaches: this will help people who run procedures know what they are optimizing; this can feed back and help to develop statistically-principled and more-scalable procedures; and so on.

Here, we will focus on several different methods (i.e., classes of algorithms, e.g., "spectral graph algorithms" as well as other classes of methods) that are very widespread in practice and that can be analyzed to prove strong bounds on the quality of the partitions found. The methods are the following.

1. Spectral-based methods. This could include either global or local methods, both of which come with some sort of Cheeger Inequality.
2. Flow-based methods. These have connections with the min-cut/max-flow theorem, and they can be viewed in terms of embeddings via their LP formulation, and here too there is a local improvement version.

In addition, we will also probably consider methods that combine spectral and flow in various ways. Note that most or all of the theoretically-principles methods people use have steps that boil down to one of these. Of course, we will also make connections with methods such as local improvement heuristics that are less theoretically-principled but that are often important in practice.

Before doing that, we should point out something that has been implicit in the discussion so far. That is, while computer scientists (and in particular TCS) often draw a strong distinction between problems and algorithms, researchers in other areas (in particular machine learning and data analysis as well as quantitatively-inclined people in nearly every other applied area) often do not. For the latter people, one might run some sort of procedure that solves something insofar as, e.g., it finds clusters that are useful by a downstream metric. As you can imagine, there is a proliferation of such methods. One of the questions we will address is when we can understand those procedures in terms of the above theoretically-principled methods. In many cases, we can; and that can help to understand when/why these algorithms work and when/why they don't.

Also, while we will mostly focus on a particular objective (called expansion or conductance) that probably is the combinatorial objective that most closely captures the bicriteria of being well-connected intra-cluster and not well-connected inter-cluster, we will probably talk about some other related methods. For example, finding dense subgraphs, and finding so-called good-modularity partitions. Those are also of widespread interest; they will illustrate other ways that spectral methods can be used; and understanding the relationship between those objectives and expansion/conductance is important.

Before proceeding, a word of caution: For a given objective quantifying how "good" is a partition, it is *not* the case that all graphs have good partitions—but all graph partitioning algorithms (as will other algorithms) will return some answer, *i.e.*, they will give you some output clustering. In particular, there is a class of graphs called *expanders* that do not have good clusters with respect to the so-called expansion/conductance objective function. (Many real data graphs have strong expander-like properties.)

In this case, i.e., when there are no good clusters, the simple answer is just to say don't do clustering. Of course, it can sometimes in practice be difficult to tell if you are in that case. (For example, with a thousand graphs and a thousand methods—that may or may not be related but that have different knobs and so are at least minorly-different—you are bound to find things look like clusters, and controlling false discovery, etc., is tricky in general but in particular for graph-based data.)

Alternatively, especially in practice, you might have a graph that has both expander-like properties and non-expander-like properties, e.g., in different parts of the graph. A toy example of this could be given by the lollipop graph. In that case, it might be good to know how algorithms behave on different classes of graphs and/or different parts of the graph.

Question (raised by this): Can we certify that there are no good clusters in a graph? Or certify the nonexistence of hypothesized things more generally? We will get back to this later.

Let's go back to finding an objective we want to consider.

As a general rule of thumb, when most people talk about clusters or communities (for some reason, in network and especially in social graph applications clusters are often called communities—they may have a different downstream, e.g., sociological motivation, but operationally they are typically found with some sort of graph clustering algorithm) "desirable" or "good" clusters tend to have the following properties:

1. Internally (intra) - well connected with other members of the cluster. Minimally, this means that it should be connected—but it is a challenge to guarantee this in a statistically and algorithmically meaningful manner. More generally, this might mean that it is "morally connected"—*e.g.*, that there are several paths between vertices in intra-clusters and that these paths should be internal to the cluster. (Note: this takes advantage of the fact that we can classify edges incident to $v \in C$ as internal (connected to other members of C) and external (connected to $\bar{C}$).
2. Externally (inter) - relatively poor connections between members of a cluster and members of a different cluster. For example, this might mean that there are very few edges with one endpoint in one cluster and the other endpoint in the other cluster.

Note that this implies that we can classify edges, i.e., pairwise connections, incident to a vertex $v \in C$ into edges that are internal (connected to other members of C) and edges that are external (connected to members of $\bar{C}$). This technically is well-defined; and, informally, it makes sense, since if we are modeling the data as a graph, then we are saying that things and pairwise relationships between things are of primary importance.

So, we want a relatively dense or well-connected (very informally, those two notions are similar, but they are often different when one focuses on a particular quantification of the informal notion) induced subgraph with relatively few inter-connections between pieces. Here are extreme cases to consider:

- Connected component, *i.e.*, the "entire graph," if the graph is connected, or one connected component if the graph is not connected.
- Clique or maximal clique, *i.e.*, complete subgraph or a maximal complete subgraph, *i.e.*, subgraph in which no other vertices can be added without loss of the clique property.

But how do we quantify this more generally?

5.2 A first try with min-cuts

Here we will describe an objective that has been used to partition graphs. Although it is widely-used for certain applications, it will have certain aspects that are undesirable for many other applications.

In particular, we cover it for a few reasons: first, as a starter objective before we get to a better objective; second, since the dual is related to a non-spectral way to partition graphs; and third, although it doesn't take into account the bi-criteria we have outlined, understanding it will be a basis for a lot of the stuff later.

5.2.1 Min-cuts and the Min-cut problem

We start with the following definition.

Definition 1. *Let $G = (V, E)$ be a graph. A* cut *$C = (S, T)$ is a partition of the vertex set V of G. An s-t-*cut *$C = (S, T)$ of $G = (V, E)$ is a cut C s.t. $s \in S$ and $t \in T$, where $s, t \in V$ are pre-specified source and sink vertices/nodes. A* cut set *is $\{(u, v) \in E : u \in S, v \in T\}$, i.e., the edges with one endpoint on each side of the cut.*

The above definition applies to both directed and undirected graphs. Notice in the directed case, the cut set contains the edges from node in S to nodes in T, but not those from T to S.

Given this set-up the *min-cut problem* is: find the "smallest" cut, *i.e.*, find the cut with the "smallest" cut set, *i.e.* the smallest boundary (or sum of edge weights, more generally). That is:

Definition 2. *The* capacity *of an s-t-cut is $c(S, T) = \sum_{(u,v)\in(S,\bar{S})} c_{uv}$. In this case, the* Min-Cut Problem *is to solve*

$$\min_{s\in S, t\in\bar{S}} c(S, \bar{S}).$$

That is, the problem is to find the "smallest" cut, where by smallest we mean the cut with the smallest total edge capacity across it, i.e., with the smallest "boundary."

Things to note about this formalization:

1. Good: Solvable in low-degree polynomial time by a polynomial time algorithm. (As we will see, min-cut = max-flow is related.)
2. Bad: Often get very unbalanced cut. (This is not *necessarily* a problem, as maybe there are no good cuts, but for this formalization, this happens even when it is known that there are good large cuts. This objective tends to nibble off small things, even when there are bigger partitions of interest.) This is problematic for several reasons:
   - **In theory.** Cut algorithms are used as a sub-routine in divide and conquer algorithm, and if we keep nibbling off small pieces then the recursion depth is very deep; alternatively, control over inference is often obtained by drawing strength over a bunch of data that are well-separated from other data, and so if that bunch is very small then the inference control is weak.
   - **In practice.** Often, we want to "interpret" the clusters or partitions, and it is not nice if the sets returned are uninteresting or trivial. Alternatively, one might want to do bucket testing or something related, and when the clusters are very small, it might not be worth the time.

   (As a forward-looking pointer, we will see that an 'improvement" of the idea of cut and min-cut may also get very imbalanced partitions, but it does so for a more subtle/non-trivial

reason. So, this is a bug or a feature, but since the reason is somewhat trivial people typically view this as a bug associated with the choice of this particular objective in many applications.)

5.2.2 A slight detour: the Max-Flow Problem

Here is a slight detour (w.r.t. spectral methods per se), but it is one that we will get back to, and it is related to our first try objective.

Here is a seemingly-different problem called the Max-Flow problem.

Definition 3. *The* capacity *of an edge $e \in E$ is a mapping $c : E \to \mathbb{R}^+$, denoted c_e or c_{uv} (which will be a constraint on the maximum amount of flow we allow on that edge).*

Definition 4. *A* flow *in a directed graph is a mapping $f : E \to \mathbb{R}$, denoted f_e or f_{uv} s.t.:*

- $f_{uv} \leq c_{uv}, \quad \forall (u,v) \in E$ *(Capacity constraint.)*
- $\sum_{v:(u,v)\in E} f_{uv} = \sum_{v:(v,u)\in E} f_{vu}, \quad \forall u \in V \backslash \{s,t\}$ *(Conservation of flow, except at source and sink.)*
- $f_e \geq 0 \quad \forall e \in E$ *(obey directions)*

A flow *in a undirected graph is a mapping $f : E \to \mathbb{R}$, denoted f_e. We arbitrarily assign directions to each edge, say $e = (u,v)$, and when we write $f_{(v,u)}$, it is just a notation for $-f_{(u,v)}$*

- $|f_{uv}| \leq c_{uv}, \quad \forall (u,v) \in E$ *(Capacity constraint.)*
- $\sum_{v:(u,v)\in E} f_{uv} = 0, \quad \forall u \in V \backslash \{s,t\}$ *(Conservation of flow, except at source and sink.)*

Definition 5. *The* value of the flow $|f| = \sum_{v \in V} |f_{sv}|$, *where s is the source. (This is the amount of flow flowing out of s. It is easy to see that as all the nodes other than s, t obey the flow conservation constraint, the flow out of s is the same as the flow into t. This is the amount of flow flowing from s to t.) In this case, the* Max-Flow Problem *is*

$$\max |f|.$$

Note: what we have just defined is really the "single commodity flow problem" since there exists only 1 commodity that we are routing and thus only 1 source/sink pair s and t that we are routing from/to. (We will soon see an important generalization of this to something called *multi-commodity flow*, and this will be very related to a non-spectral method for graph partitioning.)

Here is an important result that we won't prove.

Theorem 1 (Max-Flow-Min-Cut Theorem)**.** *The max value of an $s-t$ flow is equal to the min capacity of an $s-t$ cut.*

Here, we state the Max-Flow problem and the Min-Cut problem, in terms of the primal and dual optimization problems.

PRIMAL: (MAX-FLOW):

$$\begin{aligned} \max \quad & |f| \\ \text{s.t.} \quad & f_{uv} \le C_{uv}, \quad (uv) \in E \\ & \sum_{v:(vu)\in E} f_{vu} - \sum_{v:(uv)\in E} f_{uv} \le 0, \quad u \in V \\ & f_{uv} \ge 0 \end{aligned}$$

DUAL: (MIN-CUT):

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} c_{ij} d_{ij} \\ \text{s.t.} \quad & d_{ij} - p_i + p_j \ge 0, (ij) \in E \\ & p_s = 1, p_t = 0, \\ & p_i \ge 0, i \in V \\ & d_{ij} \ge 0, ij \in E \end{aligned}$$

There are two ideas here that are important that we will revisit.

- Weak duality: for any instance and any feasible flows and cuts, $\max flow \le \min cut$.
- Strong duality: for any instance, $\exists$ feasible flow and feasible cut s.t. the objective functions are equal, *i.e.*, s.t. $\max flow = \min cut$.

We are not going to go into these details here—for people who have seen it, it is just to set the context, and for people who haven't seen it, it is to give an important fyi. But we will note the following.

- Weak duality generalizes to many settings, and in particular to multi-commodity flow; but strong duality does not. The next question is: does there exist a cut s.t. equality is achieved.
- We can get an approximate version of strong duality, *i.e.*, an approximate Min-Cut-Max-Flow theorem in that multi-commodity case. That we can get such a bound will have numerous algorithmic implications, in particular for graph partitioning.
- We can translate this (in particular, the all-pairs multi-commodity flow problem) into 2-way graph partitioning problems (this should not be immediately obvious, but we will cover it later) and get nontrivial approximation guarantees.

About the last point: for flows/cuts we have introduced special source and sink nodes, s and t, but when we apply it back to graph partitioning there won't be any special source/sink nodes, basically since we will relate it to the all-pairs multi-commodity flow problem, i.e., where we consider all $\binom{n}{2}$ possible source-sink pairs.

5.3 Beyond simple min-cut to "better" quotient cut objectives

The way we described what is a "good" clustering above was in terms of an intra-connectivity versus intra-connectivity bi-criterion. So, let's revisit and push on that. A related thing or a different way (that gives the same result in many cases, but sometimes does not) is to say that a bi-criterion is:

- We want a good "cut value"—not too many crossing edges—where *cut value* is $E(S, \bar{S})$ or a weighted version of that. I.e., what we just considered with min-cut.
- We want good "balance" properties—*i.e.*, both sides of the cut should be roughly the same size—so both $S, \bar{S}$ are the same size or approximately the same size.

There are several ways to impose a balance condition. Some are richer or more fruitful (in theory and/or in practice) than others. Here are several. First, we can add "hard" or *explicit* balance conditions:

- Graph bisection—find a min cut s.t. $|S| = |\bar{S}| = n/2$, *i.e.*, ask for exactly 50-50 balance.
- β-balanced cut—find a min cut s.t $|S| = \beta n$, $|\bar{S}| = (1-\beta)n$, *i.e.*, give a bit of wiggle room and ask for exactly (or more generally no worse than), say, a 70-30 balance.

Second, there are also "soft" or *implicit* balance conditions, where there is a penalty and separated nodes "pay" for edges in the cut. (Actually, these are not "implicit" in the way we will use the word later; here it is more like "hoped for, and in certain intuitive cases it is true." And they are not quite soft, in that they can still lead to imbalance; but when they do it is for much more subtle and interesting reasons.) Most of these are usually formalized as quotient-cut-style objectives:

- Expansion: $\frac{E(S,\bar{S})}{\frac{|S|}{n}}$ or $\frac{E(S,\bar{S})}{\min\{|S|,|\bar{S}|\}}$ (def this as :h(S)) (a.k.a. q-cut)
- Sparsity: $\frac{E(S,\bar{S})}{|S||\bar{S}|}$ (def this as :sp(S)) (a.k.a. approximate-expansion)
- Conductance: $\frac{E(S,\bar{S})}{\frac{\text{Vol}(S)}{n}}$ or $\frac{E(S,\bar{S})}{\min(\text{Vol}(|S|),\text{Vol}(|\bar{S}|))}$ (a.k.a. Normalized cut)
- Normalized-cut: $\frac{E(S,\bar{S})}{\text{Vol}(|S|)\cdot\text{Vol}(|\bar{S}|)}$ (a.k.a. approximate conductance)

Here, $E(S, \bar{S})$ is the number of edges between S and $\bar{S}$, or more generally for a weighted graph the sum of the edge weights between S and $\bar{S}$, and $\text{Vol}(S) = \sum_{ij \in E} \deg(V_i)$. In addition, the denominator in all four cases correspond to different volume notions: the first two are based on the number of nodes in S, and the last two are based on the number of edges in S (i.e., the sum of the degrees of the nodes in S.)

Before proceeding, it is worth asking what if we had taken a difference rather than a ratio, e.g., $SA - VOL$, rather than SA/VOL. At the high level we are discussing now, that would do the same thing—but, quantitatively, using an additive objective will generally give very different results than using a ratio objective, in particular when one is interested in fairly small clusters.

(As an FYI, the first two, i.e., expansion and sparsity, are typically used in the theory algorithms algorithms, since they tend to highlight the essential points; while the latter two, i.e., conductance

and normalized cuts, are more often used in data analysis, machine learning, and other applications, since issues of normalization are dealt with better.)

Here are several things to note:

- Expansion provides a *slightly* stronger bias toward being well-balanced than sparsity (i.e. between a $10-90$ cut and a $40-60$ cut, the advantage in the denominator for the more balanced $40-60$ cut in expansion is $4:1$, while it is $2400:900 < 4:1$ in sparsity) (and there *might* be some cases where this is important. That is, the product variants have a "factor of 2" weaker preference for balance than the min variants. Similarly for normalized cuts versus conductance.

- That being said, that difference is swamped by the following. Expansion and sparsity are the "same" (in the following sense:)

  $$\min h(S) \approx \min sp(S)$$

  Similarly for normalized cuts versus conductance.

- Somewhat more precisely, although the expansion of any particular set isn't in general close to the sparsity of that set, The expansion problem and the sparsity problem are equivalent in the following sense:

  It is clear that

  $$\operatorname*{argmin}_S \Phi'(G) = \operatorname*{argmin}_S n\Phi'(G) = \operatorname*{argmin}_S \frac{C(S,\bar{S})}{\min\{|S|,|\bar{S}|\}} \frac{n}{\max\{|S|,|\bar{S}|\}}$$

  As $1 < \frac{n}{\max\{|S|,|\bar{S}|\}} \leq 2$, the min partition we find by optimizing sparsity will also give, off by a multiplicative factor of 2, the optimal expansion. As we will see this is small compared to $O(\log n)$ approximation from flow or the quadratic factor with Cheeger, and so is not worth worrying about from an optimization perspective. Thus, we will be mostly cavalier about going back and forth.

- Of course, the sets achieving the optimal may be very different. An analogous thing was seen in vector spaces—the optimal may rotate by 90 degrees, but for many things you only need that the Rayleigh quotient is approximately optimal. Here, however, the situation is worse. Asking for the certificates achieving the optimum is a more difficult thing—in the vector space case, this means making awkward "gap" assumptions, and in the graph case it means making strong and awkward combinatorial statements.

- Expansion $\neq$ Conductance, in general, except for regular graphs. (Similarly, Sparsity $\neq$ Normalized Cuts, in general, except for regular graphs.) The latter is in general preferable for heterogeneous graphs, *i.e.*, very irregular graphs. The reason is that there are closer connections with random walks and we get tighter versions of Cheeger's inequality if we take the weights into account.

- Quotient cuts capture exactly the surface area to volume bicriteria that we wanted. (As a forward pointer, a question is the following: what does this mean if the data come from a low dimensional space versus a high dimensional space; or if the data are more or less expander-like; and what is the relationship between the original data being low or high dimensional versus the graph being expander-like or not?

- For "space-like" graphs, these two bicriteria as "synergistic," in that they work together; for expanders, they are "uncoupled," in that the best cuts don't depend on size, as they are all bad; and for many "real-world" heavy-tailed informatics graphs, they are "anti-correlated," in that better balance means worse cuts.
- An obvious question: are there other notions, e.g., of "volume" that might be useful and that will lead to similar results we can show about this? (In some cases, the answer to this is yes: we may revisit this later.) Moreover, one might want to choose a different reweighting for statistical or robustness reasons.

(We will get back to the issues raised by that second-to-last point later when we discuss "local" partitioning methods. We simply note that "space-like" graphs include, *e.g.*, $\mathcal{Z}^2$ or random geometric graphs or "nice" planar graphs or graphs that "live" on the earth. More generally, there is a trade-off and we might get very imbalanced clusters or even disconnected clusters. For example, for the $G(n,p)$ random graph model if $p \geq \log n^2/n$ then we have an expander, while for extremely sparse random graphs, *i.e.*, $p < \log n/n$, then due to lack of concentration we can have deep small cuts but be expander-like at larger size scales.)

5.4 Overview Graph Partition Algorithms

Here, we will briefly describe the "lay of the land" when it comes to graph partitioning algorithms—in the next few classes, we will go into a lot more details about these methods. There are three basic ideas you need to know for graph partitioning, in that nearly all methods can be understood in terms of some combination of these methods.

- Local Improvement (and multi-resolution).
- Spectral Methods.
- Flow-based Methods.

As we will see, in addition to being of interest in clustering data graphs, graph partitioning is a nice test-case since it has been very well-studied in theory and in practice and there exists a large number of very different algorithms, the respective strengths and weaknesses of which are well-known for dealing with it.

5.4.1 Local Improvement

Local improvement methods refer to a class of methods that take an input partition and do more-or-less naive steps to get a better partition:

- 70s Kernighan-Lin.
- 80s Fiduccia-Mattheyses—FM and KL start with a partition and improve the cuts by flipping nodes back and forth. Local minima can be a big problem for these methods. But they can be useful as a post-processing step—can give a big difference in practice. FM is better than KL since it runs in linear time, and it is still commonly used, often in packages.

- 90s Chaco, Metis, etc. In particular, METIS algorithm from Karypis and Kumar, works very well in practice, especially on low dimensional graphs.

The methods of the 90s used the idea of local improvement, coupled with the basically linear algebraic idea of *Multiresolution* get algorithms that are designed to work well on space-like graphs and that can perform very well in practice. The basic idea is:

- Contract edges to get a smaller graph.
- Cut the resulting graph.
- Unfold back up to the original graph.

Informally, the basic idea is that if there is some sort of geometry, say the graph being partitioned is the road network of the US, i.e., that lives on a two-dimensional surface, then we can "coarse grain" over the geometry, to get effective nodes and edges, and then partition over the coarsely-defined graph. The algorithm will, of course, work for any graph, and one of the difficulties people have when applying algorithms as a black box is that the coarse graining follows rules that can behave in funny ways when applied to a graph that doesn't have the underlying geometry.

Here are several things to note:

- These methods grew out of scientific computing and parallel processing, so they tend to work on "space-like" graphs, where there are nice homogeneity properties—even if the matrices aren't low-rank, they might be diagonal plus low-rank off-diagonal blocks for physical reasons, or whatever.
- The idea used previously to speed up convergence of iterative methods.
- Multiresolution allows globally coherent solutions, so it avoids some of the local minima problems.
- 90s: Karger showed that one can compute min-cut by randomly contracting edges, and so multiresolution may not be just changing the resolution at which one views the graph, but it may be taking advantage of this property also.

An important point is that local improvement (and even multiresolution methods) can easily get stuck in local optima. Thus, they are of limited interest by themselves. But they can be very useful to "clean up" or "improve" the output of other methods, e.g., spectral methods, that in a principled way lead to a good solution, but where the solution can be improved a bit by doing some sort or moderately-greedy local improvement.

5.4.2 Spectral methods

Spectral methods refer to a class of methods that, at root, are a relaxation or rounding method derived from an NP-hard QIP and that involves eigenvector computations. In this case that we are discussing, it is the QIP formulation of the graph bisection problem that is relaxed. Here is a bit of incomplete history.

- Donath and Hoffman (ca. 72,73) introduced the idea of using the leading eigenvector of the Adjacency Matrix A_G as a heuristic to find good partitions.
- Fiedler (ca. 73) associated the second smallest eigenvalue of the Laplacian L_G with graph connectivity and suggested splitting the graph by the value along the associated eigenvector.
- Barnes and Hoffman (82,83) and Bopanna (87) used LP, SDP, and convex programming methods to look at the leading nontrivial eigenvector.
- Cheeger (ca. 70) established connections with isoperimetric relationships on continuous manifolds, establishing what is now known ad Cheeger's Inequality.
- 80s: saw performance guarantees from Alon-Milman, Jerrum-Sinclair, etc., connecting λ_2 to expanders and rapidly mixing Markov chains.
- 80s: saw improvements to approximate eigenvector computation, e.g., Lanczos methods, which made computing eigenvectors more practical and easier.
- 80s/90s: saw algorithms to find separators in certain classes of graphs, e.g., planar graphs, bounds on degree, genus, etc.
- Early 90s: saw lots of empirical work showing that spectral partitioning works for "real" graphs such as those arising in scientific computing applications
- Spielman and Teng (96) showed that "spectral partitioning works" on bounded degree planar graphs and well-shaped meshed, i.e., in the application where it is usually applied.
- Guattery and Miller (95, 97) showed "spectral partitioning doesn't work" on certain classes of graphs, e.g., the cockroach graph, in the sense that there are graphs for which the quadratic factor is achieved. That particular result holds for vanilla spectral, but similar constructions hold for non-vanilla spectral partitioning methods.
- Leighton and Rao (87, 98) established a bound on the duality gap for multi-commodity flow problems, and used multi-commodity flow methods to get an $O(\log n)$ approximation to the graph partitioning problem.
- LLR (95) considered the geometry of graphs and algorithmic applications, and interpreted LR as embedding G in a metric space, making the connection with the $O(\log n)$ approximation guarantee via Bourgain's embedding lemma.
- 90s: saw lots of work in TCS on LP/SDP relaxations of IPs and randomized rounding to get $\{\pm 1\}$ solutions from fractional solutions.
- Chung (97) focused on the normalized Laplacian for degree irregular graphs and the associated metric of conductance.
- Shi and Malik (99) used normalized cuts for computer vision applications, which is essentially a version of conductance.
- Early 00s: saw lots of work in ML inventing and reinventing and reinterpreting spectral partitioning methods, including relating it to other problems like semi-supervised learning and prediction (with, e.g., boundaries between classes being given by low-density regions).

- Early 00s: saw lots of work in ML on manifold learning, etc., where one constructs a graph and recovers an hypothesized manifold; constructs graphs for semi-supervised learning applications; and where the diffusion/resistance coordinates are better or more useful/robust than geodesic distances.
- ARV (05) got an SDP-based embedding to get an $O(\sqrt{\log n})$ approximation, which combined ideas from spectral and flow; and there was related follow-up work.
- 00s: saw local/locally-biased spectral methods and improvements to flow improve methods.
- 00s: saw lots of spectral-like methods like viral diffusions with social/complex networks.

For the moment and for simplicity, say that we are working with unweighted graphs. The graph partitioning QIP is:

$$\begin{aligned} \min \quad & x^T L x \\ \text{s.t.} \quad & x^T 1 = 0 \\ & x_i \in \{-1, +1\} \end{aligned}$$

and the spectral relaxation is:

$$\begin{aligned} \min \quad & x^T L x \\ \text{s.t.} \quad & x^T 1 = 0 \\ & x_i \in \mathbb{R}, \quad x^T x = n \end{aligned}$$

That is, we relax x from being in $\{-1, 1\}$, which is a discrete/combinatorial constraint, to being a real continuous number that is 1 "on average." (One could relax in other ways—*e.g.*, we could relax to say that it's magnitude is equal to 1, but that it sits on a higher-dimensional sphere. We will see an example of this later. Or other things, like relaxing to a metric.) This spectral relaxation is not obviously a nice problem, e.g., it is not even convex; but it can be shown that the solution to this relaxation can be computed as the second smallest eigenvector of L, the Fiedler vector, so we can use an eigensolver to get the eigenvector.

Given that vector, we then have to perform a *rounding* to get an actual cut. That is, we need to take the real-valued/fractional solution obtained from the continuous relaxation and round it back to $\{-1, +1\}$. There are different ways to do that.

So, here is the basic spectral partitioning method.

- Compute an eigenvector of the above program.
- Cut according to some rules, *e.g.*, do a hyperplane rounding, or perform some other more complex rounding rule.
- Post process with a local improvements method.

The hyperplane rounding is the easiest to analyze, and we will do it here, but not surprisingly factors of 2 can matter in practice; and so—*when spectral is an appropriate thing to do*—other rounding rules often do better in practice. (But that is a "tweak" on the larger question of spectral versus flow approximations.) In particular, we can do local improvements here to make the output

slightly better in practice. Also, there is the issue of what exactly is a rounding, e.g., if one performs a sophisticated flow-based rounding then one may obtain a better objective function but a worse cut value. Hyperplane rounding involves:

- Choose a split point $\hat{x}$ along the vector x
- Partition nodes into 2 sets: $\{x_i < \hat{x}\}$ and $\{x_i > \hat{x}\}$

By *Vanilla spectral*, we refer to spectral with hyperplane rounding of the Fiedler vector embedding.

Given this setup of spectral-based partitioning, what can go "wrong" with this approach.

- We can choose the wrong direction for the cut:
  - Example—Guattery and Miller construct an example that is "quadratically bad" by taking advantage of the confusion that spectral has between "long paths" and "deep cuts."
  - Random walk interpretation—long paths can also cause slow mixing since the expected progress of a t-step random walk is $O(\sqrt{t})$.
- The hyperplane rounding can hide good cuts:
  - In practice, it is often better to post-process with FM to improve the solution, especially if want good cuts, i.e., cuts with good objective function value.

An important point to emphasize is that, although both of these examples of "wrong" mean that the task one is trying to accomplish might not work, i.e., one might not find the best partition, sometimes that is not all bad. For example, the fact that spectral methods "strip off" long stringy pieces might be ok if, e.g., one obtains partitions that are "nice" in other ways. That is, the direction chosen by spectral partitioning might be nice or regularized relative to the optimal direction. We will see examples of this, and in fact it is often for this reason that spectral performs well in practice. Similarly, the rounding step can also potentially give an implicit regularization, compared to more sophisticated rounding methods, and we will return to discuss this.

5.4.3 Flow-based methods

There is another class of methods that uses very different ideas to partition graphs. Although this will not be our main focus, since they are not spectral methods, we will spend a few classes on it, and it will be good to know about them since in many ways they provide a strong contrast with spectral methods.

This class of flow-based methods uses the "all pairs" multicommodity flow procedure to reveal bottlenecks in the graph. Intuitively, flow should be "perpendicular" to the cut (i.e. in the sense of complementary slackness for LPs, and similar relationship between primal/dual variables to dual/primal constraints in general). The idea is to route a large number of commodities *simultaneously* between random pairs of nodes and then choose the cut with the most edges congested—the idea being that a bottleneck in the flow computation corresponds to a good cut.

Recall that the single commodity max-flow-min-cut procedure has zero duality gap, but that is not the case for multi-commodity problem. On the other hand, the k-multicommodity has $O(\log k)$

duality gap—this result is due to LR and LLR, and it says that there is an *approximate* min-flow-max-cut. Also, it implies an $O(\log n)$ gap for the all pairs problem.

The following is an important point to note.

Claim 1. *The $O(\log n)$ is tight on expanders.*

For flow, there are connections to embedding and linear programming, so as we will see, we can think of the algorithm as being:

- Relax flow to LP, and solve the LP.
- Embed solution in the ℓ_1 metric space.
- Round solution to $\{0, 1\}$.

5.5 Advanced material and general comments

We will conclude with a brief discussion of these results in a broader context. Some of these issues we may return to later.

5.5.1 Extensions of the basic spectral/flow ideas

Given the basic setup of spectral and flow methods, both of which come with strong theory, here are some extensions of the basic ideas.

- Huge graphs. Here want to do computations depending on the size of the sets and not the size of the graph, *i.e.*, we don't even want to touch all the nodes in the graph, and we want to return a cut that is nearby an input seed set of nodes. This includes "local" spectral methods—that take advantage of diffusion to approximate eigenvectors and get Cheeger-like guarantees.
- Improvement Methods. Here we want to "improve" an input partition—there are both spectral and flow versions.
- Combining Spectral and Flow.
  - ARV solves an SDP, which takes time like $O(n^{4.5})$ or so; but we can do it faster (*e.g.*, on graphs with $\approx 10^5$ nodes) using ideas related to approximate multiplicative weights.
  - There are strong connections here to online learning—roughly since we can view "worst case" analysis as a "game" between a cut player and a matching player.
  - Similarly, there are strong connections to boosting, which suggest that these combinations might have interesting statistical properties.

A final word to reemphasize: at least as important for what we will be doing as understanding when these methods work is understanding when these methods "fail"—that is, when they achieve their worst case quality-of-approximation guarantees:

- Spectral methods "fail" on graphs with "long stringy" pieces, like that constructed by Guattery and Miller.
- Flow-based methods "fail" on expander graphs (and, more generally, on graphs where most of the $\binom{n}{2}$ pairs but most pairs are far $\log n$ apart).

Importantly, a lot of real data have "stringy" pieces, as well as expander-like parts; and so it is not hard to see artifacts of spectral and flow based approximation algorithms when they are run on real data.

5.5.2 Additional comments on these methods

Here are some other comments on spectral versus flow.

- The SVD gives good "global" but not good "local" guarantees. For example, it provides global reconstruction error, and going to the low-dimensional space might help to speed up all sorts of algorithms; but any pair of distances might be changed a lot in the low-dimensional space, since the distance constraints are only satisfied on average. This should be contrasted with flow-based embedding methods and all sorts of other embedding methods that are used in TCS and related areas, where one obtains very strong local or pairwise guarantees. There are two important (but not immediately obvious) consequences of this.
  - The lack of local guarantees makes it hard to exploit these embeddings algorithmically (in worst-case), whereas the pair-wise guarantees provided by other types of embeddings means that you can get worst-case bounds and show that the solution to the subproblem approximates in worst case the solution to the original problem.
  - That being said, the global guarantee means that one obtains results that are more robust to noise and not very sensitive to a few "bad" distances, which explains why spectral methods are more popular in many machine learning and data analysis applications.
  - That local guarantees hold for all pair-wise interactions to get worst-case bounds in non-spectral embeddings essentially means that we are "overfitting" or "most sensitive to" data points that are most far apart. This is counter to a common design principle, e.g., exploited by Gaussian rbf kernels and other NN methods, that the most reliable information in the data is given by nearby points rather than far away points.

5.6 References

1. Schaeffer, "Graph Clustering", Computer Science Review 1(1): 27-64, 2007
2. Kernighan, B. W.; Lin, Shen (1970). "An efficient heuristic procedure for partitioning graphs". Bell Systems Technical Journal 49: 291-307.
3. CM Fiduccia, RM Mattheyses. "A Linear-Time Heuristic for Improving Network Partitions". Design Automation Conference.
4. G Karypis, V Kumar (1999). "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". Siam Journal on Scientific Computing.