Max-cut problem: a quest in graph spectra

Lapo Toloni

University of Pisa, Deparment of Computer Science
l.toloni@studenti.unipi.it

Abstract. This survey paper presents an overview of graph cut problems. The main focus is on MAX-CUT, a famous network related problem known to be NP-HARD. The three classical and trivial 2-approximation algorithms are given. Then a brief introduction of spectral graph theory is presented, showing how to exploit algebraic properties of matrices associated to graphs to achieve better approximation ratios for MAX-CUT.

1 Introduction: Graphs, cuts and hardness of approximation

1.1 MAX-CUT Problem

Graphs are a mathematical construction typically used to model connections and relations between things.
Sticking to the undirected case, a graph is a pair of sets $G = (V, E)$ where V represents the collection of things to be connected, commonly called vertices, while E is a set of unordered pairs of elements of V, modelling the relation that exists between the objects of our universe.

Given an undirected graph $G = (V, E)$, a cut in G is a partition of the vertices of G graph into two disjoint subsets $C, \bar{C} \subseteq V$ such that $\bar{C} = V \setminus C$, $C \cap \bar{C} = \emptyset$ and $C \cup \bar{C} = V$. Let $E(C, \bar{C})$ denote the set of edges with one vertex in C and one vertex in $\bar{C}$.
The MAX-CUT problem consists in finding a cut C that maximizes $|E(C, \bar{C})|$, namely the number of edges crossing the cut.
A weighted version of MAX-CUT also exists and there things get more complicated. In these instances we are also given a function $w : E \to R$ that assigns a weight to every edge, so that the problem is to find a cut with maximum weight. The MAX-CUT problem is NP-hard. It is interesting to recall the contrast with the MIN-CUT problem, which admits a linear time solutions by using a randomized algorithm.

1.2 Approximation Algorithms

MAX-CUT as many other real life problems are optimization problems. In the standard complexity theory formulation, problems are given in a decisional form, meaning that what is asked is the existence of at least one element in the space of solutions. In an optimization framework the task is to build the best solution among all the feasible ones. A large number of these optimization problems are known to be in NP-HARD, that is no polynomial time, or space, algorithm computing an exact solution is known, and probably it did not even exist.

However their existence needs to be taken into account since lines of attack serve to deal with their complexity and make them more tractable. One possible plan is to exploit the notion of approximate solution. Approximate algorithm theory is the branch of complexity theory that let formalize the notion of approximate solution for optimization problems. More precisely we have:

- Π, an optimization problem whose solving algorithm is in NP-HARD,
- S^*, an optimal solution for the instance I of Π,
- S, an approximate solution for the instance I of Π,
- a function $COST : \mathcal{S}(I_\Pi) \rightarrow \mathbb{N}$, where $\mathcal{S}(I_\Pi)$ is the space of solution of the optimization problem Π with instance I. ($\mathcal{S}(I_\Pi) \rightarrow \mathbb{R}$ in the continuous case)

The goal becomes finding a procedure that given and instance I of Π computes an approximation solution S such that:

$$\frac{COST(S)}{COST(S^*)} \leq r \qquad \text{if we have a minimization problem}$$

$$\frac{COST(S^*)}{COST(S)} \leq r \qquad \text{if we have a maximization problem}$$

In both cases a lower value for r implies a better algorithm.
In general is difficult to estimate the value of $COST(S^*)$. So as we will see in the proves in the next sections we will need to use an upperbound to $COST(S^*)$ in case of maximization problems (or a lower bound in case of minimization problems). So by letting UB be a solution yielding a cost that is an upperbound of the optimal solution we have:

$$COST(S) \leq COST(S^*) \leq COST(UB)$$

And we will prove the theorems stating the value of r by showing:

$$\frac{COST(UB)}{COST(S)} \leq r \Longleftrightarrow COST(S) \geq \frac{COST(UB)}{r}$$

2 2-Approximations algorithms for Max-cut

we will now analyze three standard approximation algorithms for the max cut problem.

2.1 Local search

Local search is a widely used heuristic method to deal with hard problems. Here we can get a taste of how this technique can be deployed to deal with approximation algorithms. For an instance I of a problem Π, let $\mathcal{S}(I_{\Pi})$ denote the set of feasible solutions for the instance I. Let S be a solution of the problem instance I, then we use the term neighborhood of S to be the set of all solution S' such that S' can be obtained from S via some local moves. This idea is subsumed by the following procedure:

1. find some form of "good" starting solution S_0;
2. devise an "improving step" to build a better solution;
3. repeat untill we are stuck with a non improving solution;

In the case of MAX-CUT the algorithm is the following:

Algorithm 1: MAX-CUT via Local Search

```
Input  : A graph $G = (V, E)$ undirected and unweighted.
Output: a solution $S$ that is $C \subset V$.
$S_0 = \{\}$;
while $\exists v \in V$ s.t. moving $v$ in the cut strictly increases $|E(C, \bar{C})|$ do
    if $V \in S$ then
        $S = S \setminus \{u\}$
    end
    else
        $S = S \cup \{u\}$
    end
end
```

The two following theorems state formally termination and approximation ratio achieved by local search.

Theorem 1. ***MAX-CUT via Local Search's Termination.***
The procedure based on local search to compute an approximate solution for a MAX-CUT instance terminates.

Proof. each step increases $|E(S, \bar{S})|$ by 1. We start from an initial solution $S_0 = \emptyset$ and for sure $|E(C, \bar{C})|$ cannot become greater than the number of edges of the graph $|E| = m$. Trivially the procedure takes m steps to emit a solution. $\square$

Theorem 2. ***MAX-CUT via Local Search's approximation ratio is 2.***
The procedure based on local search to compute an approximate solution for a MAX-CUT instance gives a solution that is in the worst case half the optimal solution.

Proof. Let S be the approximate solution returned by the procedure. We want to assess a ratio between the cost of the optimum and the cost of the solution found by local search. As it usually happens in these cases, the cost of the optimal solution is not known and so an upperbound is used in its place:

$$\frac{COST(S^*)}{COST(S)} \leq \frac{COST(UB)}{COST(S)}$$

Here and in the rest of this section we take $|E| = m$ as $COST(UB)$ so that in all the three theorems stating that the achieved approximation ratio is 2, the proof reduces to show that:

$$\frac{m}{COST(S)} \leq 2$$

To conclude the proof we need to observe that for each node $v \in V$ is guaranteed that at least $\frac{d_v}{2}$ incident edges to v belongs to $|E(C, \bar{C})|$, where d_v is the degree of the node v that is the number of incident edges to v.
Now let the cut at the local optimum computed by the local search procedure be S and take a node u with degree d_u. If the vertex u is in the cut for the solution S then some of its neighbors are in C and some other in $\bar{C}$ but for sure there cannot be more then half the degree of u edges crossing the cut otherwise S would not be a local optimum. So we can write:

$$\begin{aligned}
cut\,size &= COST(S) \\
&= \frac{1}{2} \sum_{v \in V} \#\, edges\, incident\, to\, v\, crossing\, the\, cut \\
&\geq \frac{1}{2} \frac{d_v}{2} \\
&= \frac{1}{2} \frac{2m}{2} \quad \text{since} \sum_{v \in V} d_v = 2m \\
&= \frac{m}{2}
\end{aligned}$$

□

This strategy takes $\mathcal{O}(poly\, n)$ times in the unweighted case. Unfortunately it is exponential in case of large weights associated to the edges. This explains why we may need other strategies.

2.2 Greedy

A greedy strategy is an approach that wants to make the best move at each step. The greedy strategy and the local search one are somehow similar in the sense that both aims to reach the global optimum passing through many local optima. The two differ in how the next move to take is chosen and how the space of neighbor solutions is explored. In greedy algorithms an ordering is often imposed by some kind of sorting before procedure starts computing over the collection of objects it is dealing with.
For greedy MAX-CUT the two subsets of V induced by the notion of cut of a

graph are imagined as two bins. After having fixed the order to scan the vertices of the input graph we start picking them one by one and put them in the bin such that $|E(C, \bar{C})|$ is maximized at each step.

Algorithm 2: Greedy MAX-CUT

```
Input  : A graph $G = (V, E)$ undirected and unweighted.
Output: a solution $S$ that is $C \subset V$.
sort $V$;
$C = \{\}$;
$\bar{C} = \{\}$;
Sort the vertices $\in V$;
foreach vertex $v \in V$ do
    put the v in the bin that maximizes $|E(C, \bar{C})|$;
end
```

Theorem 3. ***Greedy MAX-CUT time complexity is*** $\mathcal{O}(|V| \log |V|)$

Proof. trivially, the time complexity is dominated by the time it takes to sort the vertices of G □

Theorem 4. ***Greedy MAX-CUT is a 2-approximation***

Proof. Define the notion of responsible vertex that is the endpoint of an edge coming later in the fixed ordering and the relative quantity $r_i =$ "#edges v_i is responsible for".
Since every edge has exactly one responsible vertex it has to be $\sum_{v \in V} r_i = m$

Claim. when v_i is added to a bin, at least $\frac{r_i}{2}$ edges are added to the cut.
This is due to the fact that the other vertices adjacent to the r_i edges that v_i is responsible for have already been assigned to either C or $\bar{C}$. The set which contains the most endpoint vertices of these r i edges must contain at least $\frac{r_i}{2}$ endpoint vertices.

Since we are greedy v_i must go to the other set. Thus it is proved that every time we process a responsible vertex we are adding at least $\frac{r_i}{2}$ edges to the cut. Concluding:

$$\begin{aligned} COST(S) &= \frac{1}{2} \sum_{v \in V} \#\, edges\, added\, by\, each\, responsible\, vertex \\ &\geq \sum_{v \in V} \frac{r_i}{2} \\ &\geq \frac{m}{2} \end{aligned}$$

□

2.3 Randomized

Last but not least it is time for the simple randomized algorithm that provide the same approximation ration with respect to the other two already presented.

Algorithm 3: Randomized MAX-CUT

```
Input  : A graph $G = (V, E)$ undirected and unweighted.
Output: a solution $S$ that is $C \subset V$.
$C = \{\}$;
foreach vertex $v \in V$ do
    put $v$ in $C$ with probability $\frac{1}{2}$
end
```

Theorem 5. ***Randomized MAX-CUT is a 2-approximation***

Proof. define a random indicator variable:

$$X_e = \begin{cases} 1 & \text{if} e \in E(C, \bar{C}), \forall e \in E, \\ 0 & \text{otherwise.} \end{cases}$$

then by random indicator variables properties it follows:

$$\mathbf{E}[X_e] = Pr(X_e = 1) = \frac{1}{2}$$

so that we can estimate the expected number of edges crossing the cut as:

$$\begin{aligned} \mathbf{E}[E(C, \bar{C})] &= \mathbf{E}[\sum_{e \in E} X_e] \\ &= \sum_{e \in E} \mathbf{E}[X_e] \\ &= \frac{|E|}{2} = \frac{m}{2} \end{aligned}$$

□

2.4 On the upperbound used so far

In this part we would like to remark the fact that any approximation algorithm can achieve a better approximation ration than 2 if $|E| = m$ is used as upper bound.
The take home message here is that if we cannot obtain better approximation result sometimes it may be the case that other upper bounds are investigated.

Claim. There are instances of the MAX-CUT problem for which:

$$COST(S^*) \approx \frac{m}{2} = \frac{1}{2} COST(UB)$$

Proof. Consider K_{2n}, the complete graph on $2n$ nodes. By a trivial combinatory count we get that $m = |E| = n(2n-1) \approx 2n^2$
For any cut in K_{2n} such that $|C| = z$, $|\bar{C}| = 2n - z$ each of the vertices in C has $2n - z$ edges crossing the cut to reach its other endpoint that stays in $\bar{C}$.
Hence the total number of edges in the cut is $z(2n - z)$.
This quantity is maximized for $i = n$ and thus the largest cut in the graph has n^2 edges.
Now recalling that $COST(UB) = m$ and for K_{2n} we have $m = n(2n-1)$ it follows:

$$\frac{COST(UB)}{COST(S^*)} = \frac{n(2n-1)}{n^2} = 2 - \frac{1}{n}$$

Which gets arbitrary close to 2 as n increases.
We can conclude that by keeping $COST(UB) = m$ in some cases even the optimal algorithm could never be considered better than a 2-approximation. □

2.5 Some history

The three 2-approximation algorithms have been known in the literature for many years. Other invesigations led to linear programming based techniques that in the end do not take any significant improvements to the problem. In 1994 Goemans and Williamsons gave a 0.828-approximation algorithm, by exploiting semidefinite programming algorithms. Moreover assuming the "unique games conjecture" this approximation ratio is optimal unless $P = NP$. The treatment of this last SDP approximation is out of the scope of this survey. In the next sections the spotlights will be on the spectral approach devised by Trevisan in 2008. His algorithm achieved an approximation ration of 0.513. The analysis was improved by Soto up to a ratio of at least 0.614 2018.

3 Spectral Graph Theory

Spectral graph theory is a branch of graph theory that exploits linear algebra tools to investigate combinatorial properties of graphs and to find better solutions to many very well-known graph algorithms. It is astonishing how the mixing of two apparently unrelated mathematical fields may take such improvements.
Vectors and matrices are the bread and butter of linear algebra, in addition to a number of useful concepts and algorithms, such as determinants, **eigenvalues, eigenvectors**, and solutions to systems of linear equations. The natural application of linear algebra to graph theory comes up when we try to relate a matrix to a graph and then interpreting linear algebra concepts in the graph-theoretic framework.
The easiest representation of a graph as a matrix is via its **adjacency matrix**. Such a matrix is symmetric and has dimension $|V| \times |V|$. Its entries are 0 or 1 according to the presence of an edge between the two nodes relative to that element of the matrix. It is interesting to think of $|V|$-dimensional boolean vectors as a bipartition of the vertices, that is a **cut**.

3.1 Basic linear algebra notions

Before diving in some of the results coming out from spectral graph theory we quickly review some basic linear algebra definitions and theorems that will be useful in the next part. The proves of the two main theorems can be found in [**?**].

Definition 1. ***Symmetric Matrix***
A square matrix M on a field $\mathbb{K}$ is symmetric if and only if:

$$M = M^T$$

where M^T is the transpose of M, that is the matrix obtained by swapping the rows and the columns of M

Definition 2. ***Inner Product***
a function $< \cdot, \cdot >: \mathbb{K}^n \times \mathbb{K}^n \longrightarrow \mathbb{K}$ from vectors to scalar such that:

$$< \mathbf{v}, \mathbf{u} >= \mathbf{v}^T \mathbf{u} = \sum_{i=1}^{n} v_i^T u_i$$

Geometrically speaking the inner product is the length of the projection of the first vector onto the second, multiplied by the length of the second. Two vectors are said to be orthogonal, indicated as $\mathbf{v} \perp \mathbf{u}$, if their inner product is 0

Definition 3. ***Eigenvalues and eigenvectors***
Let M be a square matrix on a field $\mathbb{K}$, $\lambda \in \mathbb{K}$ a scalar and $mathbfv \in \mathbb{K}^n \setminus \mathbf{0}$ a vector, then if we have

$$M\mathbf{v} = \lambda \mathbf{v}$$

we say what λ is an eigenvalue of M and that $\mathbf{v}$ is an eigenvector of M corresponding to the *eigenvalue λ.*

Theorem 6. ***Spectral Theorem***
Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix with real-valued entries, then there are n real numbers (not necessarily distinct) $\lambda_1, \ldots, \lambda_n$ and n orthonormal real vectors $\in \mathbb{R}$ such that $\mathbf{x}_i$ is an eigenvector of λ_i.

3.2 Graph spectrum and bipartite graphs

The most common matrices we will associate to graphs will be symmetric and real, so we would apply the spectral theorem on them to build an orthonormal basis of eigenvectors. However it is not granted that these eigenvalues would provide any information about graph properties. The following example instead shows how the eigenvalues of a matrix are a mirror of the graph's topology

Proposition 1. ***Spectral characterization of bipartite graphs.***
If G is a bipartite graph and λ is an eigenvalue of A, adjacency matrix of G, with multiplicity k, then $-\lambda$ is an eigenvalue of A with multiplicity k.

Proof. If G is a bipartite graph, then there exists π a permutation of the rows and the columns of A, such that

$$\pi A = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$$

Let $\mathbf{u} = \begin{pmatrix} x \\ y \end{pmatrix}$ *with eigenvalue λ. Then* $\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}$
which implies $B^T x = \lambda y$ and $By = \lambda x$
This implies that $\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ -y \end{pmatrix} = \begin{pmatrix} -By \\ B^T x \end{pmatrix} = \begin{pmatrix} -\lambda y \\ \lambda x \end{pmatrix} = -\lambda \begin{pmatrix} \lambda x \\ -\lambda y \end{pmatrix}$
and thus $\begin{pmatrix} x \\ -y \end{pmatrix}$ *is an eigenvector of A with eigenvalue $-\lambda$.*
To conclude the proof, k linearly independent eigenvector with eigenvalue λ would give k linearly independent eigenvector with eigenvalue $-\lambda$, hence the claim □

This was the proof that the spectrum of a bipartite graph is symmetric around the origin. Now we show that the converse is also true

Proposition 2. ***The spectrum inducing a bipartite graph.***
If the nonzero eigenvalues of the adjacency matrix A of $G = (V, E)$ come in pairs λ_i, λ_j, with $\lambda_i = -\lambda_j$, then G is bipartite

Proof. Let k be an odd number, it implies that $\sum_{i=1}^{n} \lambda_i^k = 0$.
Note that $\lambda_1^k, \ldots, \lambda_n^k$ are the eigenvalues of A^k.
Observe that the entry of$A_i j^k$ is the number of length k walks from i to j in G the length. (it is a simple induction argument).
If G has an odd cycle of length k, the $\exists i \in [1, n] : A_i i^k > 0$ but it would mean that $\sum_{i=1}^{n} \lambda_i^k > 0$.
So we get to an absurd and it is implied that G cannot contain an odd cycle and so it is bipartite. □

Apart from the boring and specific case of bipartite graphs this approach can be further generalized to show that: λ_n is as close to λ_1 as G is similar to a bipartite graph.
And since bipartite graphs yield large max cuts, this observation will allow to obtain a new approximation algorithm for MAX-CUT.

3.3 Graphs, matrices and eigenvalues

For later algorithmic usage we need a tool that sticks together eigenvalues and optimization problems. The following characterization of eigenvalues is what serves the purpose

Theorem 7. ***(Variational characterization of eigenvalues)***
Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix, and $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of M in non-increasing order then:

$$\lambda_k = \min_{k-dim V} \max_{\boldsymbol{x} \in V - \{\boldsymbol{0}\}} \frac{\boldsymbol{x}^T M \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

The quantity $\frac{\boldsymbol{x}^T M \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$ *is called* Rayleigh quotient of $\mathbf{x}$ with respect to M, and we denote it as $R_M(\mathbf{x})$.

The three following corollaries make explicit the algorithmic useful part of this characterization.

Corollary 1. *if λ_1 is the smallest eigenvalue of a real symmetric matrix M, then*

$$\lambda_1 = \min_{\mathbf{x} \neq \mathbf{0}} R_M(\mathbf{x})$$

Furthermore, every minimizer is an eigenvector of λ_1

Corollary 2. *if λ_n is the largest eigenvalue of a real symmetric matrix M, then*

$$\lambda_n = \max_{\mathbf{x} \neq \mathbf{0}} R_M(\mathbf{x})$$

Furthermore, every maximizer is an eigenvector of λ_n

Corollary 3. *if λ_1 is the smallest eigenvalue of a real symmetric matrix M, and $\mathbf{x}_1$ is an eigenvector of λ_1 then*

$$\lambda_2 = \min_{\mathbf{x} \neq \mathbf{0},\, \mathbf{x} \perp \mathbf{x_1}} R_M(\mathbf{x})$$

The last one enligths a procedure to iteratively compute all the eigenvalues starting from the smallest.

Given an undirected d-regular graph $G = (V, E)$, the approach of spectral graph theory is to associate a symmetric real matrix to G and to relate the eigenvalues of the matrix to combinatorial properties of G. While an adjacenjy matrix A such that $A_{i,j} = 1$ if $(i, j) \in E$ and $A_{i,j} = 0$ otherwise, would be the most natural representation of G, what we really need to study cut algorithms is a slight modification of A, that is the Laplacian matrix, defined as

Definition 4. ***Normalized Laplacian***
The normalized Laplacian matrix of an undirected d-regular graph $G = (V, E)$ is

$$L := I - \frac{1}{d} A$$

Such a choice is motivated by the fact that if we want to study cuts in graphs, it makes sense to choose a matrix M such that

$$\mathbf{x}^T M \mathbf{x} = \sum_{(u,v) \in E} (x_u - x_v)^2$$

because by using boolean vectors $\mathbf{x}$ of dimension $|V|$ we are inducing a partition over the vertices of G and the right-hand-side of the previous expression is equivalent to count the number of edges that cross the cut. Using a sort of reverse engineering process we can see that the matrix with the properties we would like to have is $dI - A$, that is we subtract to the diagonal matrix whose diagonal values are the degree of the vertices of G, the adjacency matrix A of G. Lastly the matrix is normalized by scaling of a factor $\frac{1}{d}$ so that all its eigenvalues are in the range $[0, 2]$ and the average values of the eigenvalues of L are independent of the degrees of the nodes.

In addition thanks to the variational characterization of eigenvalues of real symmetric matrices, we can think of the eigenvalues of a matrix as optima of min-max optimization problems in which the cost function is the Rayleigh quotient.

What we get is the reformulation of two famous hard problems as relaxations of convex discrete optimization problems. The minimization problem whose cost function is the Rayleigh quotient of a normalizad Laplacian matrix of a graphs corresponds to the uniform sparsest cut problem, while the maximization problem with the same cost function corresponds to the max cut problem.

3.4 First results of spectral graph theory

In this section we will see some easy consequences of the definitions given so far and we will build some intuition that will be used in the next section to present an algorithm to compute a better approximation for MAX-CUT.

Theorem 8. *Let G be a d-regular undirected graph, let A be the adjacency matrix of G and $L := I - \frac{1}{d}A$ its normalized Laplacian. Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the real eigenvalues of L with multiplicities, in nondecreasing order. Then:*

1. *$\lambda_1 = 0$ and $\lambda_n = 2$.*
2. *$\lambda_k = 0$ if and only if G has at least k connected components.*
3. *$\lambda_n = 2$ if and only if one of the connected component of G is bipartite.*

Proof. Glueing together the variational characterization of eigenvalues and the fact the concept of the Rayleigh quotient of L we get

$$\lambda_1 = \min_{\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}} \frac{\mathbf{x}^T L \mathbf{x}}{\mathbf{x}^T \mathbf{x}} = \min_{\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}} \frac{\sum_{(u,v) \in E} (x_u - x_v)^2}{d \sum_v x_v^2}$$

Since this is a ratio between two sum of squares, for sure $\lambda_1 \geq 0$
To find the minimum possible value for the smallest eigenvalues it is enough to note that the constant vector $\mathbf{1}$ makes the numerator of the Rayleigh quotient 0. So 0 is the smallest eigenvalue of L and the n dimensional vector $\mathbf{1}$

Now for simplicity instead of proving the second point that is:
"$\lambda_k = 0$ if and only if G has at least k connected components."

we will just show what are the consequence of $\lambda_2 = 0$, To get the full statement of the second point we repeat the same argument more generally by using the

generic min-max formula for λ_k*.*
Now recalling that λ_2 *by the variational characterization is equal to*

$$\lambda_2 = \min_{\substack{\mathbf{x}\neq\mathbf{0} \\ \sum_{v\in V} x_v = 0}} \frac{\sum_{(u,v)\in E}(x_u - x_v)^2}{d\sum_{v\in V} x_v^2}$$

where $\sum_{v\in V} x_v = 0$ *is equivalent to say that we want* $\mathbf{x}$ *to be orthogonal to the minimizer we used for* λ_1 *that is* $\mathbf{1}$*. This is needed in order to find the eigenvalues iteratively by exploiting the third corollary of the min-max theorem.*

So we have

$$\lambda_2 = 0 \iff \exists\, \mathbf{x} \neq \mathbf{0} \ \textit{s.t.} \sum_{v\in V} x_v = 0 \ \textit{and} \ \forall (u,v) \in E\, x_u = x_v$$

That explicitly means:

- *For* λ_2 *to be zero there must be some vector* $\mathbf{x}$ *such that for every edge with endpoints* u *and* v*,* $x_u = x_v$*.*
- *This means, by induction, that such a vector has the same value on all the entries that are on the same path.*
- *This implies that in every connected component this vector* $\mathbf{x}$ *must have the same value in all the entries corresponding to an edge in the same connected component.*
- $\mathbf{x}$ *is not zero but it has to sum to zero which means it must have strictly positive values in some entries and strictly negative values in some other entries.*
- *It implies that the graph cannot be connected because there cannot be any path from a vertex that corresponds to a negative entry of* $\mathbf{x}$ *to a vertex that corresponds to a positive entry of* $\mathbf{x}$*.*
- *We can conclude that the graph is disconnected.*

It is left to show the converse: "G is disconnected $\implies \lambda_2 = 0$*"*

- *if there are two connected components we can give value* -1 *to vertices in one connected component and* $+1$ *to vertices in the other.*
- *in this way the vector is non zero and the property* $\forall(u,v) \in E\, x_u = x_v$ *holds.*
- *to get that* $\mathbf{x}$ *sums to zero it is enough to scale* $\mathbf{x}'s$ *entries assigning to vertices in one component the value of the number of vertices in the other component and viceversa.*

($\lambda_{\mathbf{n}}$)

It is left to study the greatest eigenvalue of the Laplacian λ_n, computable as:

$$\lambda_n = \max_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u - x_v)^2}{d\sum_{v\in V} x_v^2}$$

we want to rewrite that numerator using the following identities:

$$(x_u - x_v)^2 = x_u^2 - 2x_ux_v + x_v^2 = 2(x_u^2 + x_v^2) - (x_u + x_v)^2$$

$$2\mathbf{x}^T\mathbf{x} - \mathbf{x}^T L\mathbf{x} = \frac{1}{d}\sum_{(u,v)\in E}(x_u + x_v)^2$$

to obtain

$$\begin{aligned}
\lambda_n &= \max_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u - x_v)^2}{d\sum_{v\in V} x_v^2} && (1)\\
&= \max_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq\mathbf{0}}} \frac{2d\sum_{v\in V} x_v^2 - \sum_{(u,v)\in E}(x_u + x_v)^2}{d\sum_{v\in V} x_v^2} && (2)\\
&= 2 - \min_{\substack{\mathbf{x}\in\mathbb{R}^n\\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u + x_v)^2}{d\sum_{v\in V} x_v^2} && (3)\\
& && (4)
\end{aligned}$$

from this follows that

$$\lambda_n \leq 2$$

and to have $\lambda_n = 2$ there must exist a non-zero vector $\mathbf{x}$ such that

$$\sum_{(u,v)\in E}(x_u + x_v)^2 = 0$$

so that we can recap what we have just stated as

$$\lambda_n = 2 \iff \exists\,\mathbf{x} \neq \mathbf{0} \text{ s.t. } \forall (u,v)\in E\; x_u = -x_v$$

(assuming that G is connected)

- *define $A := \{v : x_v > 0\}$ and $B := \{v : x_v < 0\}$.*

- *the set $A \cup B \neq \emptyset$ since $\mathbf{x} \neq \mathbf{0}$*
- *$A \cup B$ is either the entire graph or else it is disconnected from the rest of the graph, because otherwise an edge with an endpoint in $A \cup B$ and an endpoint in its complement $V - (A \cup B)$ would give a positive contribution to $\sum_{(u,v)\in E}(x_u + x_v)^2$.*

- *every edge incident on a vertex on A must have the other endpoint in B, and viceversa.*
- *Thus, $A \cup B$ is a connected component or a collection of connected components of G, which is bipartite into A and B*

$\square$

The usual next step to be taken in spectral graph theory is to give a robust version of this characterization of the extremes of the spectrum of the Laplacian. Robust versions of the structural results, called in the literature Cheeger's inequalities, make it possible interpreting fractional solutions of this relaxations (i.e. what if λ_2 is close to 0.000000001) as significant combinatorial properties of graphs.

4 Spectral Max Cut

We saw that the largest eigenvalue λ_n of the Laplacian matrix of a graph G is 2 if and only if G contains a bipartite connected component.
Since λ_n is a continuous value that has a discrete and combinatorial effect on the topology of the relative graph, we may hope to find a measure of how close a graph is to be bipartite with respect to how close is λ_n to 2.

We want a fractional parameter that is zero if and only the graph has a bipartite connected component, and that is close to zero if and only if the graph is close to contain a bipartite connected components.

We start the quest in defining this parameter by recalling the result we got for λ_n in the last section:

$$\lambda_n = 2 - \min_{\substack{\mathbf{x}\in\mathbb{R}^n \\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u + x_v)^2}{d\sum_{v\in V} x_v^2} \iff 2 - \lambda_n = \min_{\substack{\mathbf{x}\in\mathbb{R}^n \\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u + x_v)^2}{d\sum_{v\in V} x_v^2}$$

Now we want to restrict this continuous minimization problem to the discrete case, that is we would like to move from looking for minimizer in $\mathbb{R}^{|V|}$ to minimizer over a discrete field.

The combinatorial problem equivalent to finding an almost bipartite connected component in a graph can be restated as:

"given an unweighted, undirected, d-regular graph $G = (V, E)$ look for a non-empty $C \subseteq V$ and a bipartition (A, B) of C such that:
the ratio between the number of violating edges and incident edges to S is small.
We say that an edge $(u, v) \in E$ is violating if it is an edge between two vertices in C and one endpoint is in A and the other in B."

The discrete field of our vector space that serves the purpose is very close to the boolean one.
We need to distinguish between vertices in the cut C and vertices outside of it, so in $V - C$.
Moreover we need to further distinguish between vertices that are in $A \subseteq C$ and vertices in $B \subseteq C$.
Let $\mathbf{y}$ be a vector in $\{-1, 0, 1\}^{|V|}$ that maps vertices of G in the following manner:

– $v \in V$ is a node of $A \subseteq C \iff y_v = -1$;
– $v \in V$ is a node of $B \subseteq C \iff y_v = 1$;
– $v \in V$ is a node of $V - C \iff y_v = 0$;

Keep in mind that A and B are a bipartition of C. It means that $A \cup B = C$ and $A \cap B = \emptyset$.
We now define the **bipartiteness ratio of y** as

$$\beta(\mathbf{y}) = \frac{\sum_{(u,v)\in E} |y_u + y_v|}{d \sum_{v \in V} y_v}$$

looking at how this parameter has been defined we can note that:

– the numerator counts for the two possible kinds of violating edges:
  • edges contained in C that have endpoints both lying in A or both in B are counted with a weight of 2;
  • edges going from vertices in C to vertices in $V - C$ are counted with a weight of 1;
– the denominator contains the sum of the degrees of the vertices of C. For non regular graphs, this quantity is usually called the volume of C (indicated as $vol(C)$).
  Note that $vol(V) = 2 \cdot |E|$.

So in a more explicit way we can write

$$\beta(\mathbf{y}) = \frac{2E(A, A) + 2E(B, B) + E(A \cup B, V - A \cup B)}{vol(A \cup B)}$$

Thus this parameter $\beta(\mathbf{y})$ stands for the fraction of edges of G incident to the cut C, induced by the vector $\mathbf{y}$, that have to be removed to convert C into a bipartite connected component of G.

We can now recast the expression of $\beta(\mathbf{y})$ into the combinatorial optimization problem equivalent to the continuous one regarding λ_n by define the **bipartiteness ratio of the graph** G as

$$\beta(G) = \min_{\substack{\mathbf{y} \in \{-1,0,1\}^n \\ \mathbf{y} \neq \mathbf{0}}} \beta(\mathbf{y})$$

Smaller values of $\beta(G)$ corresponds to a smaller number of edges to remove from G to get a bipartite connected component in G.

It is left to find how to relate the discrete optimization problem that wants to compute $\beta(G)$ and the continuos one that aims for λ_n. One of the so called Cheeger's inequalities formalizes this relation binding λ_n to the bipartiteness ratio of the graph $\beta(G)$.

Theorem 9. ***(Cheeger's inequalities for λ_n, Trevisan)***
Let G be an undirected regular graph and let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of the normalized laplacian, with repetitions, then

$$\frac{2-\lambda_n}{2} \leq \beta(G) \leq \sqrt{2 \cdot (2-\lambda_n)}$$

Proof. the left part is the easy direction:

$$\begin{aligned} 2-\lambda_n &= \min_{\substack{\mathbf{x}\in\mathbb{R}^n \\ \mathbf{x}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}(x_u+x_v)^2}{d\sum_{v\in V} x_v^2} \\ &\leq \min_{\substack{\mathbf{y}\in\{-1,0,1\}^n \\ \mathbf{y}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E}|y_u+y_v|^2}{d\sum_{v\in V}|y_v|^2} \\ &\leq \min_{\substack{\mathbf{y}\in\{-1,0,1\}^n \\ \mathbf{y}\neq\mathbf{0}}} \frac{\sum_{(u,v)\in E} 2\cdot|y_u+y_v|}{d\sum_{v\in V}|y_v|} \\ &= 2\cdot\beta(G) \end{aligned}$$

In particular, proving this direction of the inequality is equivalent to state that if there exists a local bipartite connected component in G then λ_n is close to two
The second inequality is harder to prove but it is useful to design a procedure that finds a cut $C \subseteq V$ with a nice bipartiteness ratio.
The prove the other direction we need to apply the next lemma to an eigenvector $\mathbf{x}$ of λ_n.

Lemma 1 (main). *For every $\mathbf{x}\in\mathbb{R}^n$, $\mathbf{x}\neq\mathbf{0}$ there is a threshold t*

$$0 < t < \max_v |x_v|$$

such that, if we define $\mathbf{y}^{(t)} \in \{-1,0,1\}^n$ as

$$\mathbf{y}_v^{(t)} = \begin{cases} -1 & \text{if } x_v \leq -t \\ 0 & \text{if } -t \leq x_v < t \\ 1 & \text{if } x_v \geq t \end{cases}$$

we have

$$\beta(\mathbf{y}^{(t)}) \leq \sqrt{2 \cdot \frac{\sum_{(u,v \in E)} (x_u + x_v)^2}{d \sum_{v \in V} x_v^2}} = \sqrt{2 \cdot (2 - \lambda_n)}$$

Proof. look at [1], page 8.

□

Now it is possible to constructively use the second inequality

$$\beta(\mathbf{y}^{(t)}) \leq \sqrt{2 \cdot (2 - \lambda_n)}$$

to give a procedure that returns a partition C of V yielding a nice bipartiteness ratio for G.
The procedure does the following steps:

- sort the vertices according to $|y_v|$;
- process the vertices in the sorted order $\{v_1, \dots, v_n\}$
- for every cut C_k that is a suffix of the sorted order, bipartite C_k in (A, B) according to the sign of y_v and compute $\beta(C_k)$;
- return the cut C_k that minimizes its bipartiteness ratio;

We call this last procedure **QuasiBipartition** and by a simple combinatorial argument we can see that

Proposition 3. ***QuasiBipartition terminates in polynomial time***
Quasi-Bipartition returns a solution in time $\mathcal{O}(|E| + |V| \log |V|)$

Proof. The term $|V| \log |V|$ comes out from the initial sorting of the vertices according to their coordinates.
The cut that minimizes $\beta(G)$ can be computed in $\mathcal{O}(E)$, since the numerator of every $\beta(C_k)$ is obtained iteratively reusing in part the numerator of the bipartiteness ratio of the previous step.
We start by computing the first β in constant time and in every following step we subtract edges that were violating for the previous cut and add other edges that are violating for the current cut.
Both the operations take time $\mathcal{O}(d_{v_k})$. All this operations take time be done in $\mathcal{O}(d_v)$ and thus repeating this argument for every vertex we get a total running time that is order $\mathcal{O}(\sum_{v \in V} d_v) = \mathcal{O}(|E|)$ □

4.1 Spectral MAX-CUT

Finally we can give an approximation algorithm for MAX-CUT that exploits the procedure to compute $\beta(G)$.
The idea is to recursively apply that procedure to certain subgraphs of G:

- find $\mathbf{y}$ that minimizes $\beta(G)$;
- remove the non-zero vertices from G (note that this is the safest choice since $\mathbf{y}$ yields a way to partition them removing the minimum number of violating edges)

- recur on $G' = (V', E')$ where $V' \subseteq V$ is the set of vertices such that $\forall v' \in V' : y_{v'} = 0$
- stop the recursion when the volume of the residual graph is very small.

Note that a small volume of the residual graph implies that we already have an approximated optimal global cut in the current residual. (think to the expression of $\beta(\mathbf{y})$)
Otherwise if the volume is still large, the optimal cut of G is still a good cut in the next residual graph. This last statement implies that λ_n is close to 2 and that we can find a $\mathbf{y}'$ that gives a small bipartiteness ratio.

The **RecursiveSpectralCut** overall algorithm is the following:

- take in input $G = (V, E)$;
- use **QuasiBipartition** on G to find the cut C and the bipartition (A, B) of C such that:

$$2E(A, A) + 2E(B, B) + E(A \cup B, V - A \cup B) \leq \sqrt{2 \cdot (2 - \lambda_n)} \cdot vol(A \cup B)$$

- if $V = A \cup B$
  - return (A, B);
- else
  - Let $V' := V - (A \cup B)$ and let $G' = (V', E')$ be the subgraph induced by V';
  - $(C, V' - C) :=$ **RecursiveSpectralCut**(G')
  - return the best cut between $(C \cup A, (V' - C) \cup B)$ and $(C \cup B, (V' - C) \cup A)$

4.2 Analysis of RecursiveSpectralCut

We call $maxcut(G)$ the fraction of edges of G cut by an optimal solution S^*. The link between this notation and the one used in the first section is

$$maxcut(G) = \frac{COST(S^*)}{|E|}$$

We assume $maxcut(G) = 1 - \epsilon$, that is the optimal solution is not cutting all the edges of G and so G is not bipartite. Then we have

Lemma 2. *if $maxcut(G) = 1 - \epsilon$, then RecursiveSpectralCut(G) finds a cut crossed by at least $(1 - 4\sqrt{\epsilon}) \cdot |E|$ edges.*

Proof. the proof is by induction on the number of recursive calls to RecursiveSpectralCut.

(base case)

In the base case the algorithm never recurs.

So it must be the case that $A \cup B = V$, and so (A, B) is already a cut of G. Let count the number of edges of G not crossing the cut:

$$E(A,A) + E(B,B) \leq \frac{1}{2}\sqrt{2 \cdot (2-\lambda_n)} \cdot vol(V) \leq 2\sqrt{\epsilon} \cdot |E| \leq 4\sqrt{\epsilon}|E|$$

since $2 - \lambda_n \leq 2\epsilon$ and $vol(V) = 2|E|$

(inductive step)

We start by assessing the number of uncut edges by the algorithm

$$E(A,A) + E(B,B) + \frac{1}{2}E(A \cup B, V') + (|E'| - E'(C, V' - C))$$

because we have to count:

- *all the edges with both endpoints in A and both endpoints B;*
- *all the edges of G' not cut in the recursive step;*
- *half of the edges from $A \cup B$ to V' (because the best way of combining the cuts loses at most half of the edges);*

by using the inductive hypothesis and $2 - \lambda_n \leq 2\epsilon$ it follows

$$E(A,A) + E(B,B) + \frac{1}{2}E(A \cup B, V') \leq \sqrt{\epsilon} \cdot vol(A \cup B)$$

$$|E'| - E'(C, V' - C) \leq 4\sqrt{\epsilon'} \cdot |E'|$$

where ϵ' is such that $maxcut(G') = 1 - \epsilon'$ Call

$$\rho \ = \frac{|E| - |E'|}{|E|}$$

the fraction of edges of G not in G'.
Then we have:

$$vol(A \cup B) \leq 2|E - E'| = 2\rho|E|$$

and

$$|E'| = |E| \cdot (1 - \rho)$$

Recalling that the number of edges not cut by the optimal cut of G' is at most the number of edges not cut by the optimal cut of G, given that G' is a subgraph of G, follows that

$$\epsilon'|E'| \leq \epsilon|E| \implies \epsilon' \leq \epsilon \cdot \frac{|E|}{|E|} = \epsilon \cdot \frac{1}{1-\rho}$$

The total number of edges not cut by the algorithm is at most

$$\sqrt{\epsilon} \cdot vol(A \cup B) + 4\sqrt{\epsilon'} \cdot |E'|$$

$$\leq \sqrt{\epsilon} \cdot 2\rho|E| + 4\sqrt{\epsilon \cdot (1-\rho)}|E|$$
$$\leq 4\sqrt{\epsilon}|E|$$

where in the last line we used

$$4\sqrt{\epsilon}|E| = 2\rho + 4\sqrt{1-\rho} \leq 4$$

which follows from

$$\sqrt{1-\rho} \leq \sqrt{(1-\frac{\rho}{2})^2} = 1 - \frac{\rho}{2}$$

We can conclude that the total number of edges cut is at most

$$|E| - \#\mathit{uncut\ edges} = |E| - 4\sqrt{\epsilon}|E| = (1-4\sqrt{\epsilon})|E|$$

□

4.3 Interpreting the spectral approximation of MAX-CUT

This algorithm works well in case of graphs that are almost bipartite, that is when ϵ is small, since this would imply that $maxcut(G)$ is very close to 1, we know that $maxcut(G) = 1$ in case of a bipartite G.
For small ϵ the quality of the solution found by this spectral algorithm is close to the one achieved by the SDP algorithm. Infact SDP-MAX-CUT algorithm finds a cut crossed by about $(1 - \frac{2}{\pi}\sqrt{\epsilon})|E|$ edges, that for small ϵ it is $\approx (1-4\sqrt{\epsilon})|E|$. The SDP-algorithm bound is the best possible polynomial time approximation if we assume the unique games conjecture and $P \neq NP$.

Unfortunately in case of large ϵ the spectral algorithms does not find nice cuts. Infact if $\epsilon > \frac{1}{64}$, the algorithm guarantees to cut less then half the edges of the graph, and this is worse than the three trivial approximation algorithms' performances we saw in the first section.

However it is possible to prove that by always choosing the best cut between the one returned by this spectral algorithm and the one returned by the greedy algorithm, we can deal with large ϵ too, achieving an approximation ratio equal to 0.531. That is indeed better than the trivial one. Moreover the results of a tighter analysis on the same algorithm, led by Soto in 2018 [2], reached an approximation ration equal to 0.614247.

References

1. Luca Trevisan. Max Cut and the smallest eigenvalue. In *SIAM J. Comput. 41(6):1769–1786, 2012.*
2. José Soto. Improved Analysis of a Max Cut Algorithm Based on Spectral Partitioning. In *arXiv:0910.0504v2.*