Saarland
Informatics Campus

Script of Lecture 14, Approximation Algorithms
Summer term 2017

Tobias Mömke, Hang Zhou
http://www-cc.cs.uni-saarland.de/course/61/

Written by Tobias Mömke

Semidefinite Programming - Part 2

In the previous lecture, we have seen the basic ideas of semidefinite programming and we have seen the Goemans-Williamson SDP algorithm for MaxCut.

The amazing thing about the MaxCut algorithm is that it is unique games hard to improve the approximation ratio: unless the unique games conjecture is wrong, there is no polynomial time algorithm with approximation ratio better than $\alpha_{\mathrm{GW}} \approx 0.878$.

We aim to view this result in a broader scope. Before we come to this, however, let us consolidate our ability to use SDPs.

1 Vertex Coloring

Coloring graphs is a classical problem that has been around for centuries: how many colors do we need for a map if we want to color each country, but neighboring countries should be colored with different colors. For maps (which correspond to planar graphs, and we want to color the faces), the four color theorem states that 4 colors are enough. (Deciding if 3 colors are enough, though, is NP-hard.)

Here, we are interested in a more general problem. Given an arbitrary graph, how many colors do we need to color all vertices such that adjacent vertices do not have the same color? (This is a generalization of the map coloring since we can identify the faces of a planar graph with vertices of another graph, called the dual graph.)

General graph coloring turns out to be one of the hardest problems in the area of polynomial time approximation algorithms. Let us start with a few preliminary results that will be useful later on.

Let Δ be the maximum degree of a given graph G.

Theorem 1. *We can find a vertex coloring for G with at most $\Delta + 1$ colors.*

Proof. We simply color the graph greedily: we iteratively choose uncolored vertices and color them with the first color c from $\{1, 2, \ldots, \Delta + 1\}$ such that no neighbor is colored c. The color c exists because of the degree bound: there are at most Δ colors with which neighbors can be colored. □

Let us now consider the threshold problem of graph coloring: given a number k, is there a coloring with k colors? For $k = 1$ the problem is trivial and for $k = 2$, we observe that the problem is equivalent to deciding whether G is bipartite.

If G is bipartite, we can color G greedily by coloring an arbitrary vertex with 1 and then iteratively color all neighbors with the respective other color. If we are not able to perform the coloring, we have detected an odd cycle with colors $1, 2, 1, 2, \ldots, 1$ or the other way around. This contradicts that G is bipartite.

It turns out that $k = 2$ is the best we can do in polynomial time.

2 Coloring 3-colorable graphs.

How do we color a graph if we have the promise that there is a solution that only uses three colors? Coloring 3-colorable graphs turns out to be a very hard problem. To gain some more intuition, let us see how to apply our insights above.

Theorem 2. *There is a $4\sqrt{n}$-approximation algorithm to color 3-colorable graphs.*

Proof. As long as we can find a vertex v of degree at least $\sqrt{n}$, we perform the following steps.

- Color v with an unused color c_1;
- 2-color the neighborhood of v with two new colors c_2, c_3;
- Remove v and its neighborhood from G;

Since G is 3-colorable and none of the neighbors of v can have the same color as v, the neighborhood is 2-colorable. Since we take three new colors, we do not interfere with previous colors. We can iterate this at most $\sqrt{n}$ times (since $n = \sqrt{n} \cdot \sqrt{n}$), using in total $3\sqrt{n}$ colors. Afterwards, all degrees are smaller than $\sqrt{n}$ and thus we can color the remaining vertices with $\Delta + 1 \leq \sqrt{n}$ colors. □

3 Using an SDP

We obtain a better algorithm by using the following vector program. We have a vector v_i for each $i \in V(G)$.

$$\begin{aligned} &\text{minimize} \quad \lambda && && (1)\\ &\text{s.t.} \quad v_i \cdot v_j \leq \lambda && \text{for all } \{i,j\} \in E(G)\\ &\qquad v_i \cdot v_i = 1 && \text{for all } i \in V(G)\\ &\qquad v_i \in \mathbb{R}^n && \text{for all } i \in V(G) \end{aligned}$$

Suppose we are given a three-coloring of G. Then we introduce three vectors v_a, v_b, v_c, one for each color. These vectors are arranged on a disk with an angle of $2\pi/3$ between each pair of vectors. We align the vertex vectors with the color vectors, e.g., if vertex i has color a, we set $v_i = v_a$. Let us now consider two vertices i and j. If both are colored with the same color, we have $v_i \cdot v_j = 1$, since they are the same vectors and the length is one. Otherwise, if i and j are colored differently, we use the insight from the previous lecture that the angle determines the inner product:

$$v_i \cdot v_j = ||v_i||\,||v_j|| \cos(2\pi/3) = -1/2\,.$$

We next consider the relaxation, i.e., the solution obtained when solving the vector program. Since we have the promise that G is 3-colorable, we conclude that there is a solution such that $v_i \cdot v_j \leq -1/2$ for each edge $e = \{i,j\} \in E(G)$, i.e.,

$$\lambda \leq -1/2\,. \tag{2}$$

Instead of obtaining the result directly as in the case of MaxCut, we iteratively find solutions that bring us closer to the integral solution. (In spirit this has similarities to iterated linear programming, but on the technical side we do very different things.)

Definition 1. A *semicoloring* of a graph G with $n = |V(G)|$ vertices is a vertex coloring such that at most $n/4$ edges have end points with the same color.

In particular, if we find a semi-coloring of G, there is an induced subgraph H of G with at least $n/2$ vertices such that H has a coloring (where for all edges the end points have distinct colors). The reason is that $n/4$ edges are incident to at most $n/2$ distinct vertices.

If we have an algorithm that finds a semi-coloring with k colors, we can run the following algorithm.

While $V(G) \neq \emptyset$

(i) Pick k new colors;

(ii) Find a semi-coloring of G with these k colors;

(iii) Identify a set S of at least $n/2$ vertices such the subgraph H induced by S is 3-colored;

(iv) Remove $V(H)$ from G and remove the colors from $V(G) \setminus V(H)$;

Observe that identifying the subgraph can be done efficiently: consider all edges with equally colored end points and collect these points in a set S'. Then the complement $V(G) \setminus S'$ is the set that we want to have.

The algorithm produces a coloring with $k \cdot \log n$ colors. What remains is an algorithm that finds a semi-coloring. In the spirit of the MaxCut algorithm, we compute an optimal solution to the vector program and then do the following.

We set $t := 2 + \log_3(\Delta)$ and choose t random vectors $r_1, r_2, \ldots, r_t$. Each of these vectors is chosen exactly in the same way as r was chosen for MaxCut.

Recall that all vectors in the solution to the vector program point to the n-dimensional unit sphere $\mathbb{S}^n$, and each vector r_i determines a hyperplane that separates $\mathbb{S}^n$ into two halves.

Then the t random vectors partition $\mathcal{S}^n$ into 2^t regions. We now obtain the following coloring. Number the regions with colors $\{1, 2, \ldots, 2^t\}$. For each vertex i, if v_i points to region number a, color i with a.

Theorem 3. *With probability at least* $1/2$*, the algorithm produces a semicoloring of* $4\Delta^{\log_3 2}$ *colors.*

Proof. The number of colors follows directly from the construction:

$$2^t = 2^{2+\log_3 \Delta} = 4 \cdot 2^{\log_2 \Delta / \log_2 3} = 4\Delta^{1/\log_2 3} = 4\Delta^{\log_3 2}\,.$$

We are left with showing that we obtain a semicoloring with probability $1/2$.

Suppose two vertices $i \neq j$ obtain the same color c. This means that both vertices fall into region c. In our analysis of MaxCut we have seen that for each vector r_i, the probability that v_i and v_j are separated by r is $\arccos(v_i \cdot v_j)/\pi$. Since we chose the t random vectors independently, the probabilities multiply. Since we are interested in the complementary event (the vectors are not separated), we obtain the following.

$$\Pr[i \text{ and } j \text{ are colored with the same color}] = (1 - \arccos(v_i \cdot v_j)/\pi)^t\,.$$

If there is an edge $e = \{i, j\}$, (2) implies

$$(1 - \arccos(v_i \cdot v_j)/\pi)^t \leq (1 - \arccos(\lambda)/\pi)^t \leq (1 - \arccos(-1/2)/\pi)^t\,.$$

Recall that $\cos(2\pi/3) = -1/2$. We therefore obtain

$$(1 - \arccos(-1/2)/\pi)^t = (1 - (2\pi/3)/\pi)^t = (1/3)^t \leq 1/(9\Delta)\,.$$

To summarize, for each edge e, the end-points are colored with the same color with probability at most $1/(9\Delta)$. To compute the expected number of edges with this property, we observe that G has $m \leq n \cdot \Delta/2$ edges. Thus, in expectation there are at most $n/18$ edges with both ends colored with the same color.

To finish the proof, we apply Markov's inequality which states that for a non-negative random variable X and $a > 0$, $\Pr[X \geq a] \leq \mathrm{E}[X]/a$. Let X be the random variable that has as value the number of edges with both ends colored with the same color. We obtain

$$\Pr[X \geq 1/4] \leq \mathrm{E}[X]/(n/4) \leq (n/18)/(n/4) \leq 1/2\,.$$

□

Since $\Delta < n$, we obtain a semicoloring with $O(n^{\log_3 2}) \approx O(n^{0.631})$ colors. Observe that this is worse than $O(n^{1/2})$, which we obtained with the simple algorithm in the beginning.

4 Combining both algorithms.

To finish our analysis, we combine the combinatorial insights with our SDP-based algorithm. The basic idea is that a small Δ leads to a good SDP-based approximation, whereas a large Δ (close to n) is good for the combinatorial algorithm. To balance both algorithms, we introduce a parameter σ.

If there is a vertex v of degree at least σ, we run the combinatorial algorithm: we color v with a new color and the neighborhood with two further new colors; we remove all of these vertices from G. Otherwise, if there is no degree σ vertex, we run the SDP-based algorithm. Unlike in the previous analysis, however, we can bound m by $n \cdot \sigma/2$. We can therefore color the remaining graph with $\sigma^{\log_3 2}$ colors.

Theorem 4. *With probability* $1/2$*, the algorithm computes a semicoloring with* $O(n^{0.387})$ *colors.*

Proof. The combinatorial part uses $3n/\sigma$ many colors in total, since we remove at least σ vertices in each iteration. We therefore want to find a σ that minimizes $3n/\sigma + \sigma^{\log_3 2}$. Asymptotically, it is sufficient to minimize the maximum (we only lose a factor of at most 2). We therefore want both parts of the algorithm to use the same number of colors. We also don't care about the factor 3 of the combinatorial algorithm and solve $n/\sigma = \sigma^{\log_3 2}$. This implies $\sigma = n^{\log_6 3} \approx n^{0.613}$. By using this σ in both algorithms we conclude that the total number of colors is $n^{0.387}$ as claimed. □

Observe that the probability $1/2$ can be amplified to an arbitrary number $1 - \epsilon$ by running the algorithm multiple times.

Finding good approximation algorithm for coloring 3-colorable graphs stays one of the important problems in the field. The currently best algorithm uses slightly less than $n^{1/5}$ colors and was obtained Kawarabayashi and Thorup [1].

The lower bounds are far away: we know that it is NP-hard to obtain a 5-coloring and that it is UG-hard to obtain a coloring with $O(1)$ colors.

5 General View

In the beginning of the class, we said that α_{GW} is tight for MaxCut unless the unique games conjecture (UGC) is false. We can see this result in a broader context.

MaxCut can be seen as a so-called constraint satisfaction problem (CSP).

We identify each vertex i with a boolean variable x_i, i.e., x_i can take the value 0 or 1. Then each edge $e = \{i, j\}$ in G imposes a constraint that $x_i \neq x_j$. This way, finding a maximum cut in G is the same as finding an assignment of boolean values to the variables that maximizes the number of satisfied constraints.

You have seen further CSPs in your Bachelor's classes on theoretical computer science: Satisfiability (SAT) is a famous example of a CSP. An important version of SAT is 2SAT which is defined as follows. There are m clauses, each of them composed of exactly two literals (l_a, l_b).

A literal l can be x_i of $\bar{x_i}$ for a boolean variable x_i. A clause is satisfied, if at least one of its literals is one. We formula (a set of clauses) is satisfied, by an assignment of variables to $0, 1$, if all clauses are satisfied. The optimization version of 2SAT is Max-2SAT: we want to find an assignment that maximizes the number of satisfied clauses. Thus the clauses determine the constraints and we want to maximize the number of satisfied constraints. We note that while 2SAT is in P, Max-2SAT is NP-hard.

Using SDPs, we can solve *all* 2CSPs optimally, as shown by Raghavendra [2].

Theorem 5. *Let Λ be an arbitrary max- or min CSP with a finite domain. We can construct an SDP and a rounding scheme for Λ such that, if the UGC is true, the obtained approximation ratio is best possible.*

How does this result relate to coloring 3-colorable graphs? If we want an optimal solution, we can formulate the problem as a CSP with ternary variables. then we have constraints $x_i \neq x_j$ for each edge $\{i, j\}$, and the only difference to MaxCut is that $x_i, x_j \in \{0, 1, 2\}$. Observe that Theorem 5 can handle domains of size three.

There is, however, another important difference. We do not accept violations of constraints at all, i.e., instead of maximizing the number of satisfied constraints, we want all constraints to be satisfied and we want to find a minimal size domain that can do so.

References

[1] Ken-ichi Kawarabayashi and Mikkel Thorup. Coloring 3-colorable graphs with less than $n^{1/5}$ colors. *J. ACM*, 64(1):4:1–4:23, 2017.

[2] Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *STOC*, pages 245–254. ACM, 2008.

[3] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

The first four sections are based on the book of Williamson and Shmoys [3]