E0 249: Approximation Algorithms February 23, 2022

Week 8

Lecture 2: Beyond Worst Case Analysis

Instructors: Arindam Khan, Anand Louis *Scribe: Shirish Gosavi*

1 Introduction

A problem, in its general form, may be NP-hard but may turn out to be "easy" in some restricted form. One such problem is 3-coloring a 3-colorable graph.

A vertex coloring of a graph G is *proper* if no adjacent vertices receive the same color. The **chromatic number**, $\chi(G)$, of G is the minimum number of colors in a proper vertex coloring of G. A graph is k-colorable if each of the nodes can be assigned exactly one of k colors so that it is a proper vertex coloring of the graph. It is known that to decide whether a graph is 3-colorable is NP-complete. It is also known that the problem of properly coloring a graph with k colors, given that the chromatic number of graph is k, is NP-hard for any $k \geq 3$. In this lecture we will look at an algorithm that properly colors any δ-dense 3-colorable graph with high probability in polynomial time.

Before describing the coloring algorithm for δ-dense 3-colorable graphs, we will briefly describe a couple of algorithms related to special cases of 3-colorable graphs in the next section **2**. In section **3** we will describe the main algorithm covered in this lecture that properly colors a δ-dense 3-colorable graph in polynomial time. In the last section **4** we will show that otherwise hard problems such as *max cut* in a general case becomes "easy" for special instances of graphs known as low threshold rank graphs.

Specific references are mentioned at appropriate places. A couple of general references used are [1] and [2].

2 3-colorable graphs

If G is a complete 3-partite graph, then coloring is easy.

Algorithm 1: Coloring a 3-partite graph

1. Pick an arbitrary vertex v and color it "1".
2. Color *N(v)* using two colors. *N(v)* consists of all the vertices belonging to the two partitions other than the partition of v. Therefore, they can be easily colored using two colors, "2" and "3".
3. The remaining uncolored vertices in G will be from the partition corresponding to v. Therefore, they can be colored with "1".

Let G be a random 3-colorable graph sampled as follows: partition the vertex set into 3 equal parts and add edges between different vertices belonging to different parts with probability p. For a large range of p, Alon and Kahale gave an algorithm to color it with 3 colors with high probability [3]. The authors claim that most k-colorable graphs are quite dense and, hence, easy to color. Their justification is that in a typical k-colorable graph, the number of common neighbors of any pair of vertices with the same color exceeds considerably that of any pair of vertices of distinct colors, and hence a simple coloring algorithm based on this fact already works with high probability. In their opinion it is more difficult to color sparser random k-colorable graphs. The main result presented in [3] is a polynomial-time algorithm that works for sparser random 3-colorable graphs. If the edge probability p satisfies $p \geq c/n$, where c is a sufficiently large absolute constant, the proposed algorithm optimally colors the corresponding random 3-colorable graph with high probability. It

uses spectral properties of the graph and is based on the fact that, *almost surely*, an accurate approximation of the color classes can be read from the eigenvectors corresponding to the smallest two eigenvalues of the adjacency matrix, A. In short, $A \approx E[A]$ w.h.p. The eigenvectors of A can be used to color the graph. We denote by *E[A]* the expected adjacency matrix of the random 3-colorable graph, where the *(i,j)*-th entry corresponds to the probability that there is an edge between vertex i and vertex j. If vertex i and vertex j belongs to the same partition, then *E[A(i,j)] = 0*, otherwise *E[A(i,j)] = p*. So the expected adjacency matrix *E[A]* is equal to $p \times A$ where A is the adjacency matrix of the complete 3-partite graph. Therefore, the eigenvectors of A and *E[A]* will be the same. Alon and Kahale use this fact to obtain a valid 3-coloring with high probability.

3 Coloring algorithm for δ-dense 3-colorable graphs

In the following discussion we assume that the graph is δ-dense 3-colorable. A graph is δ-dense if minimum vertex degree is $\geq \delta n$ where n is the number of vertices.
Let us recall the definition of k-list coloring. Let each vertex i in the graph has a list L_i of at most k colors. Each vertex is to be assigned a color from its list such that `color[i]` $\neq$ `color[j]` $\forall (i,j) \in E$. If such an assignment is possible, then the graph is k-list colorable and the assignment is called as k-list coloring of the graph.
It is known that 2-list coloring problem has a polynomial time algorithm. There is a polynomial time reduction from 2-list coloring problem to 2-SAT problem (see subsection 3.2) and there are many algorithms to solve a 2-SAT problem in polynomial time (e.g. [4]). The polynomial time algorithm to properly color a δ-dense 3-colorable graph is based on this fact. The following subsection describes the algorithm at a high level and the later subsections give details of each step mentioned in the algorithm.

3.1 Outline of the algorithm

The main steps in the algorithm are given below.

Algorithm 2: Coloring a δ-dense 3-colorable graph

1. Sample a sufficiently large random subset of S of vertices.
2. "Guess" the coloring of S
3. Color $V \backslash S$ using the 2-list coloring algorithm.

In step 1, if S is sufficiently large, w.h.p. each vertex $v \in V \backslash S$, will have a neighbor in S. This is because min vertex degree is $\geq \delta n$. The coloring of S can be done in brute-force manner by trying out each possibility till we get the desired coloring. It can be done in $3^{|S|}$ time. Once S is colored, each $v \in V \backslash S$ has only two choices of colors. Therefore, $V \backslash S$ vertices can be colored using the polynomial time algorithm for 2-list coloring. In the following subsections we will elaborate on the steps given the outline above 2.

3.2 2-list coloring

We will show the reduction from 2-list coloring to 2-SAT in polynomial time. Since 2-SAT is solvable in polynomial time, it will imply that 2-list coloring is also polynomial time solvable.
Reduction from 2-list coloring to 2-SAT

1. Each vertex $i \in V$ must be assigned one out of two colors $\left\{c_1^{(i)}, c_2^{(i)}\right\}$.

2. Introduce Boolean variable x_i for each vertex i.

$$x_i = \begin{cases} \textit{true} \text{ denotes } i \text{ getting color } c_1^{(i)} \\ \textit{false} \text{ denotes } i \text{ getting color } c_2^{(i)} \end{cases} \tag{1}$$

3. For edge $(i, j) \in E$, we have $color(i) \neq color(j)$. This can be formulated as 2-SAT constraint. Depending upon the color lists available at i and j, different scenarios arise and the 2-SAT constraint changes accordingly. All the different possible scenarios are described below.

   (a) $\left\{c_1^{(i)}, c_2^{(i)}\right\} \cap \left\{c_1^{(j)}, c_2^{(j)}\right\} = \phi$: If the lists associated with vertex i and vertex j are disjoint, the constraint is trivially satisfied as for any combination $color(i) \neq color(j)$.

   (b) $c_1^{(i)} = c_1^{(j)}$ and $c_2^{(i)} \neq c_2^{(j)}$: The first color in the list for vertex i is same as the first color in the list for vertex j. However, the second colors in the lists for both vertices are different from each other. Therefore, if the vertices are to get different colors, one of i and j must be colored with second color in its own list. It means either x_i or x_j or both must be false. That is, the coloring will be valid only if $\neg(x_i \wedge x_j) = \neg x_i \vee \neg x_j$ is *true*.

   (c) $c_1^{(i)} = c_2^{(j)}$ and $c_2^{(i)} \neq c_1^{(j)}$: The first color in the list of vertex i is same as the second color in the list of vertex j, however the second color in the list of vertex i is different from the first colors in the list of vertex j. Therefore, the coloring will be valid only if vertex i is assigned the second color from its list or vertex j is assigned the first color from its list. The variables x_i and x_j must be *false* and *true* respectively. This combination is captured by the expression in is in the then valid coloring if $\neg x_i \vee x_j$ is *true*..

   (d) $c_1^{(i)} = c_1^{(j)}$ and $c_2^{(i)} = c_2^{(j)}$: Applying the reasoning like previous case, there will be valid coloring if $\neg\big((x_i \wedge x_j) \vee (\neg x_i \wedge \neg x_j)\big) = (\neg x_i \vee \neg x_j) \wedge (x_i \vee x_j)$ is *true*.

   (e) $c_1^{(i)} = c_2^{(j)}$ and $c_2^{(i)} = c_1^{(j)}$: Coloring will be valid only if $\neg\big((x_i \wedge \neg x_j) \vee (\neg x_i \wedge x_j)\big) = (\neg x_i \vee x_j) \wedge (x_i \vee \neg x_j)$ is *true*.

Using the construction of clauses from the adjacency relationship between the vertices and the lists associated with the vertices, we can construct an instance of a 2-SAT problem. We can solve the 2-SAT instance using polynomial time algorithm and from the values assigned to Boolean variables in the solution, we can deduce the color for each vertex. Thus, 2-list coloring can be solved in polynomial time.

3.3 Sampling S

The first step of the algorithm is to sample a sufficiently large random subset S of vertices such that every vertex $v \in V \backslash S$ has a neighbor in S with high probability.
Let each $v \in V$ is added to S independently with probability p. We will now find out suitable value for p, that will ensure the desired property in S, that is, each vertex in $V \backslash S$ has a neighbor in S with high probability.
Consider a vertex $u \in V$. The probability that none of u's neighbors are in S is given by the following sequence of expressions.

$$Pr[|N(u) \cap S| = 0] = (1-p)^{|N(u)|} \leq (1-p)^{\delta n} \leq e^{-p\delta n} \tag{2}$$

If we choose $p = \frac{10}{\delta}\frac{\log n}{n}$, then

$$Pr[|N(u) \cap S| = 0] \leq e^{-p\delta n} = e^{-10 \log n} = n^{-10} \tag{3}$$

Since vertices are added independently, the upper bound on the probability that at least one vertex $\in V \backslash S$ does not have a neighbor in S is $\sum_{i=1}^{n} n^{-10} = n(n^{-10}) = n^{-9}$. Therefore, with probability at least 1 - n^{-9}, each vertex $v \in V$ has a neighbor in S.
With the chosen value of p ($p = \frac{10}{\delta}\frac{\log n}{n}$), we will get the following:

- Expected size of sample = $\mathrm{E}[|S|] = pn = \frac{10 \log n}{\delta}$
- Using Chernoff bound, $\Pr\left[|S| \leq \frac{20 \log n}{\delta}\right] \geq 1 - n^{O(1)}$. Thus, $|S|$ is less than or equal to $\frac{20 \log n}{\delta}$ with high probability.
- Even if brute force method is used to color vertices in S in all possible ways, the cost will not be too much as can be seen from the inequalities given below.
  Number of possible colorings of S is at most $3^{|S|} \leq 3^{\frac{20 \log n}{\delta}} \leq n^{O(\frac{1}{\delta})}$, which is a polynomial since δ is a constant.

3.4 Coloring $V \backslash S$

We have already seen that S can be colored using 3 colors and the operation is not expensive. We have also seen that every vertex in $V \backslash S$ has a neighbor in S with high probability. Therefore, with high probability, each vertex in $V \backslash S$ has at most 2 possible colors. Thus, coloring of vertices in $V \backslash S$ is nothing but the 2-list coloring problem. As discussed earlier, it can be solved in polynomial time by reducing it to 2SAT problem. Summing up all the results shown above, we can conclude that a δ-dense 3 colorable graph can be colored in time $n^{o(\frac{1}{\delta})}$ with high probability.
This concludes our discussion about the polynomial time coloring algorithm to color a δ-dense 3-colorable graph with high probability.

4 Low threshold rank graphs

Researchers have found that dense instances are "easy" for any 2-CSP, for example max cut, max 2-SAT, unique games etc. It is also observed that instances which are close to "expanders" are "easy" for any 2-CSP.
Let $\mathcal{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ be the normalized adjacency matrix of the graph.
Let $rank_{\geq \varepsilon}(G)$ be the number of eigenvalues of $\mathcal{A}$ which are $\geq \varepsilon$.

The following theorem formalizes the notion that instances which are "close" to expanders are "easy" for any 2-CSP.

Theorem 1. *(informal statement) [5] and [6] : For problems such as 2-CSP, etc. there exists an algorithm running in time $n^{O(rank_{\geq \varepsilon}(G)/\varepsilon^2)}$ and produces a $1 + O(\varepsilon)$ approximation to the problem.*

4.1 Low threshold rank graphs

For a β-spectral expander G, $rank_{\geq \varepsilon}(G) = 1$ for $\varepsilon \geq \beta$.
Let G be a (δn)-regular graph.

$$\mathrm{trace}(\mathcal{A}^2) = \|\mathcal{A}\|_F^2 = n \cdot (\delta n) \cdot \frac{1}{(\delta n)^2} = \frac{1}{\delta}$$
$$\mathrm{trace}(\mathcal{A}^2) = \sum_{i \in [n]} \lambda_i((\mathcal{A})^2) \geq \varepsilon^2 |\{i : |\lambda_i| \geq \varepsilon\}|$$

Therefore, $rank_{\geq \varepsilon}(G) \leq \frac{1}{\varepsilon^2 \delta}$. Applying the theorem 1 we can immediately see that for problems such as 2-CSP etc there exists a polynomial time algorithm with $1 + O(\varepsilon)$ approximation to the problem.

Therefore, low threshold rank graphs serve as examples of "easy" instances for any 2-CSP.

References

[1] V. Vazirani, *Approximation Algorithms*, Springer Berlin Heidelberg, 2013. URL: https://books.google.co.in/books?id=bJmqCAAAQBAJ.

[2] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms.*, Cambridge University Press, 2011.

[3] N. Alon and N. Kahale, *A spectral technique for coloring random 3-colorable graphs*, SIAM J Comput **26** (1970). doi:10.1137/S0097539794270248.

[4] B. Aspvall, M. Plass and R. Tarjan, *A linear-time algorithm for testing the truth of certain quantified boolean formulas*, Information Processing Letters **14** (1982). doi:10.1016/0020-0190(82)90036-9.

[5] B. Barak, P. Raghavendra and D. Steurer, *Rounding semidefinite programming hierarchies via global correlation*, CoRR **abs/1104.4680** (2011). URL: http://arxiv.org/abs/1104.4680, arXiv:1104.4680.

[6] V. Guruswami and A. K. Sinop, *Lasserre hierarchy, higher eigenvalues, and approximation schemes for quadratic integer programming with PSD objectives*, CoRR **abs/1104.4746** (2011). URL: http://arxiv.org/abs/1104.4746, arXiv:1104.4746.