Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

SGT Toolbox – A Playground for Spectral Graph Theory

Master's Thesis

Michael König

mikoenig@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Tobias Langner, Barbara Keller
Prof. Dr. Roger Wattenhofer

April 4, 2012

Acknowledgements

I would like to thank my advisors, Tobias Langner and Barbara Keller, for their steady support during the development of this thesis in spite of the at times stagnant progress and changing topics of investigation. I am also thankful to Yuval Emek for sharing his wisdom on Luby's algorithm and providing the proof idea for the z_{max} proof for general graphs. Further, I would like to thank Prof. Dr. Roger Wattenhofer for overseeing this thesis and offering his approaches to problems when needed.

Abstract

We present an interactive toolbox application we have developed to enable users to gain an intuitive understanding of the "meaning" of eigenvalues and eigenvectors of graph-related matrices. In addition, we explore spectral embeddings and the effects of graph altering algorithms on the eigenvalues of a graph.

Furthermore, we examine Luby's fast distributed randomized algorithm for finding a maximal independent set (MIS) of a network of nodes. We compare different graph classes with respect to how many rounds the algorithm takes to complete on them on average. We suggest that certain graph classes exhibit a logarithmic lower bound for this number of rounds.

Lastly, we prove a logarithmic upper bound for z_{max}, the largest distance from any vertex to its nearest MIS vertex after one round of Luby's algorithm, once for path graphs and then for all graphs in general.

Contents

Chapter 1

Spectral Graph Theory

1.1 Motivation

Spectral graph theory (SGT) deals with the eigenvalues and eigenvectors of various matrices related to graphs and what they express about the graphs. There are many applications: for example, bounds on graph-theoretical properties such as a graph's diameter have been proven in terms of some of the eigenvalues. And eigenvectors can be used to generate embeddings of graphs which completely disregard the vertex labeling — meaning two isomorphic graphs will always have the same embedding.

SGT is a challenging field of research because it is hard to gain an intuitive understanding of the meaning of the eigenvalues/-vectors. The *SGT Toolbox* was developed to enable interactively exploring how graphs relate to their eigenvalues/-vectors and how small changes to a graph, such as removing a vertex or changing an edge weight, perturb those values. Furthermore, we implemented graph altering algorithms and studied their impact on a graph's eigenvalues/-vectors.

1.2 Definitions

Definition 1.1 (Graphs) *A* graph $G = (V, E, w)$ *is defined by its vertex set* V*, its edge set* $E \subseteq \{(u, v) \mid u, v \in V\}$ *and its edge weight function* $w : E \to \mathbb{R}^+$ *(where* $\mathbb{R}^+$ *denotes the set of positive real numbers).* ◇

For reasons of brevity, we will write $w(i, j)$ instead of $w((i, j))$ for the edge (i, j).

Definition 1.2 (Unweighted Graphs) *An* unweighted graph *is a graph where every edge has the same weight. We will simply write* $G = (V, E)$ *to imply the edge function* $\forall e \in E : w(e) = 1$*.* ◇

Definition 1.3 (Undirected Graphs) *A graph $G = (V, E, w)$ is called* undirected *if and only if for every edge $(u, v) \in E$*

$$(u, v) \in E \Leftrightarrow (v, u) \in E \ \wedge \ w(u, v) = w(v, u)$$

holds. ◇

All the graphs we will be examining will be undirected and most of them will be unweighted. Without loss of generality we will always assume $V = \{1, \ldots, n\}$.

The two graph related matrices considered most commonly in spectral graph theory are the adjacency matrix and the Laplacian matrix:

Definition 1.4 (Adjacency Matrix) *The* adjacency matrix A_G *of a graph $G = (V, E, w)$ is defined as the $n \times n$-matrix whose entries $a_{i,j}$ are given by:*

$$a_{i,j} = \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \ . \qquad \diamond$$

Definition 1.5 (Vertex Degree) *The* degree of a vertex $v \in V$ *in a graph $G = (V, E, w)$ is given by*

$$degree(v) = \sum_{(v,u) \in E} w(v, u) \ . \qquad \diamond$$

Definition 1.6 (Laplacian Matrix) *For a graph G, let A_G be its adjacency matrix and D_G be the diagonal $n \times n$-matrix where the i-th diagonal entry is the degree of vertex i, then the* Laplacian matrix L_G *of G is defined as the $n \times n$-matrix given by*

$$L_G = D_G - A_G \ . \qquad \diamond$$

Definition 1.7 (Path Graphs) *The* path graph $Path_n = (V, E)$ *is defined as follows:*

$$V = \{1, 2, \ldots, n\}$$
$$E = \{(i, i+1) \mid 1 \leq i < n\} \qquad \diamond$$

Definition 1.8 ($G_{n,p}$ Graphs) *$G_{n,p}$ is the class of random unweighted graphs with n vertices, where every possible edge between two different vertices exists independently of the other edges with probability p.* ◇

1.3 The SGT Toolbox

The *SGT Toolbox* is a lightweight application which displays an undirected graph alongside the eigenvalues and eigenvectors of its Laplacian and adjacency matrices. It allows the user to edit the graph by adding or removing vertices and edges one at a time or by running a graph altering algorithm on it. After editing, the altered graph's eigenvalues are placed into a table for easy comparison with the eigenvalues of the previous graphs. (see fig. 1.1)

A variety of ways to generate new graphs is offered. While an adjacency matrix can be entered directly, it is also possible to generate a graph choosing one from several basic graph types such as hypercubes, star graphs, grids and different kinds of random graphs. These basic graphs may be added to the current graph or even substitute every vertex of the current graph. This allows creating clustered graphs in a handy way.

Furthermore, the application supports displaying 2-dimensional spectral embeddings of graphs (cf. section 1.4). Adequate eigenvectors are chosen automatically, unless the user manually selects which ones to use.

While the SGT Toolbox was originally designed to be used in the area of spectral graph theory it has since been extended to offer features useful when studying Luby's fast MIS algorithm (see chapter 2).

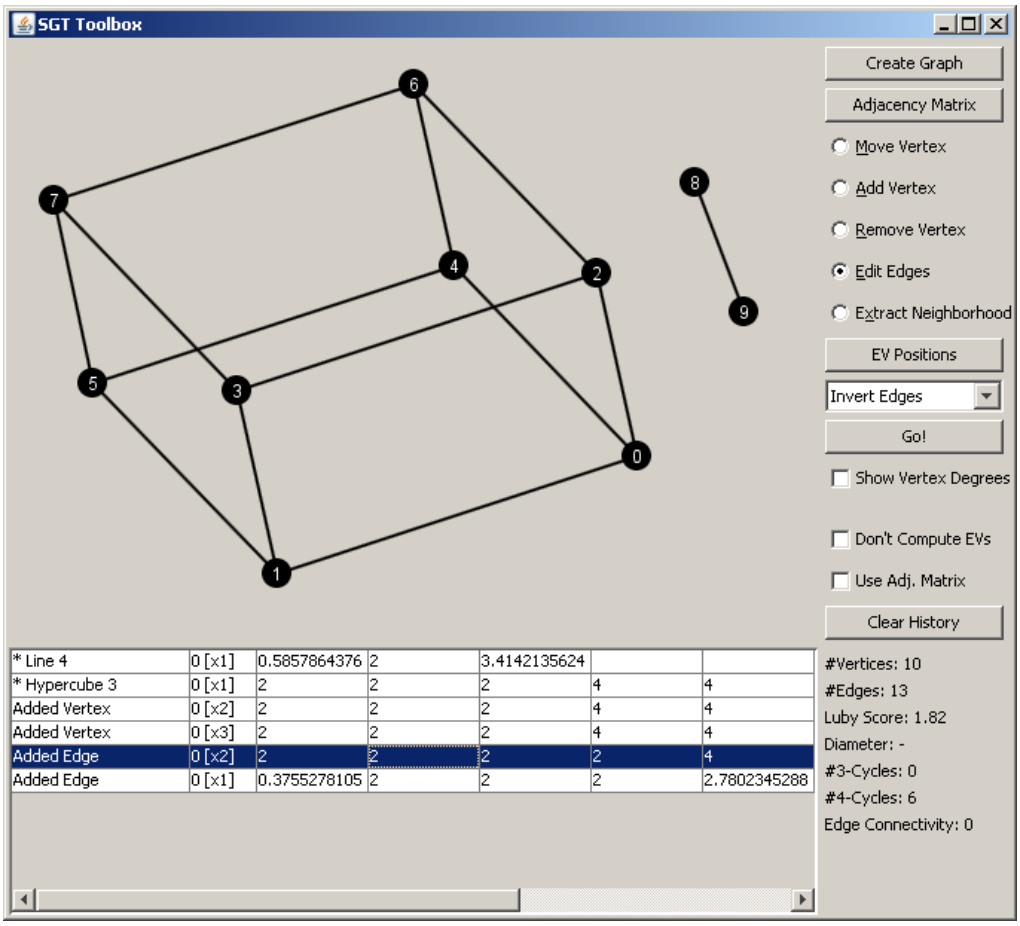

Figure 1.1: Main window of the SGT Toolbox. In the top left the current graph is displayed. At the bottom the eigenvalues of its Laplacian are shown on the left and various properties such as its diameter and its Luby score (cf. section 2.3) are displayed on the right.

1.4 Spectral Embeddings

The elements of k eigenvectors of the Laplacian of a graph can be used to create a k-dimensional embedding for the graph, which is called a "spectral embedding". The coordinates of vertex i are given by the i-th elements of each of the k vectors. Figure 1.2 shows a few examples. Spectral embeddings appear to take the inherent structure of the graph into account while not reflecting the labeling of the vertices.

Because the eigenvalues and eigenvectors do not depend on the labeling of the vertices of a graph, this gives us a way to test two graphs for isomorphism: if their spectral embeddings (picking the same eigenvectors) match, save for a

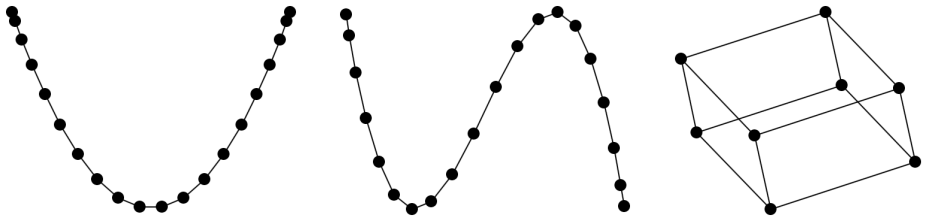

Figure 1.2: 2-dimensional spectral embeddings of $Path_{20}$ using the first and second eigenvectors (left), $Path_{20}$ using the first and third eigenvectors (middle) and a 3-dimensional hypercube using the first and third eigenvectors (right)

possible mirroring, the graphs might match, too. However, if the embeddings do not match, there cannot be an isomorphism between the graphs. On a more basic level just comparing the eigenvalues will allow for a similar test.

Another application of these embeddings is detecting vertex clusters in graphs. Figure 1.3 shows two examples illustrating the idea. McSherry [1] proved that this can be employed reasonably reliably for two clusters if we can make a few assumptions about the graphs.

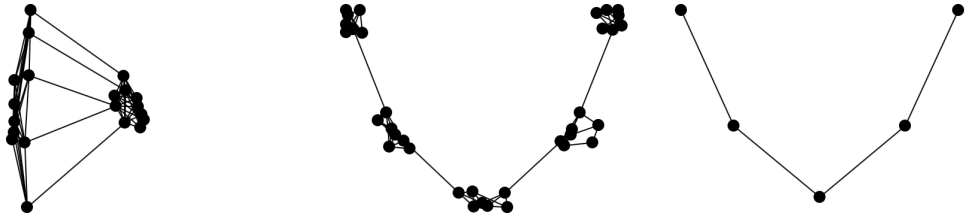

Figure 1.3: 2-dimensional spectral embeddings of a graph with two distinguishable clusters (left), $Path_5$ with each vertex replaced by a $G_{n=7,p=0.5}$ (middle) and $Path_5$ for comparison (right) (first and second eigenvectors were used in all three cases)

Note: the "first", "second" etc. eigenvectors referenced in the captions of figures 1.2 and 1.3 refer to the eigenvector corresponding to the "smallest non-zero eigenvalue", the eigenvector corresponding to the "second smallest non-zero eigenvalue", and so on.

1.5 Impact of Graph Alteration on Eigenvalues

We hoped to discover that certain graph altering algorithms affected the spectral properties of graphs in a predictable manner. In particular the smallest non-zero eigenvalue λ_2 of the Laplacian of a graph appeared promising as Cheeger's

inequalities relate it closely to the Cheeger constant of the graph. The Cheeger constant of a graph is a measure for how well connected the graph is. For a graph $G = (V, E)$ it is given by

$$h_G = \min\left\{ \frac{|\partial A|}{|A|} \;\middle|\; A \subseteq V, 0 < |A| < \frac{|V|}{2} \right\}$$

where ∂A is the set of edges crossing the boundary of A:

$$\partial A = \{(u, v) \in E \mid u \in A, v \in V \setminus A\} \ .$$

Cheeger [2] originally proved the relationship for manifolds. It was later extended to graphs, but it seems unclear whom this extension goes back on.

At first we used the SGT Toolbox to test our hypotheses, but we felt that doing this manually did not allow us to capture enough of the possible graphs. We used the toolbox's codebase to simulate different graph altering algorithms on 30000 $G_{n,p}$ graphs with 50 to 100 vertices and values for p from 0.2 to 0.8. Specifically we tracked the largest and the three smallest non-zero eigenvalues of the Laplacian before and after executing the graph altering algorithm. We then examined the difference $(old - new)$, the quotient $(\frac{old}{new})$ and the quotient of the logarithms $(\frac{\log old}{\log new})$ for each of the tracked values.

The algorithms we inspected were:

- Three separately tracked subsequent rounds of Luby's fast MIS algorithm (see algorithm 2)
- 1, 30 and 300 rounds of a spring embedder algorithm (given by algorithm 1) initialized with random vertex positions; here we used the edge weight function used internally by the algorithm to compute the eigenvalues for the different embeddings before and after. Our implementation is using Coulomb repulsion and Hooke attraction as was first suggested by Fruchterman and Reingold [3].
- "Performing a min cut" on the graph, i.e. finding any minimal cut and removing the edges it consists of from the graph

The results showed the following patterns:

Luby's algorithm generally decreased the eigenvalues. The differences as well as the quotients increased monotonously as n and p grew. With each subsequent round on the same graph the change diminished.

The spring embedding also decreased all the tracked eigenvalues, and the differences and quotients increased as n and p grew as well.

For Luby's algorithm we repeated this experiment with the eigenvalues of the adjacency matrix. The results were similar with a few exceptions. This time

Algorithm 1 One Round of a 2-dimensional "spring embedder" using Coulomb repulsion and Hooke attraction

Input: An undirected, unweighted graph $G = (V, E)$ and a 2-dimensional embedding $e : V \to \mathbb{R}^2$ for G.

Compute the edge weight function $w(u, v) = \begin{cases} |e(u) - e(v)| & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$

Compute the force $f(u, v) = (\frac{c_{repulsion}}{|u-v|})^2 - c_{attraction} w(u, v)|u - v|$

Compute the new embedding $e_{out}(v) = e(v) + \sum_{\substack{u \in V \\ u \neq v}} \frac{u-v}{|u-v|} f(u, v)$

return e_{out}

larger values of n decreased the quotient (while still increasing the difference), and the differences did not monotonously grow for larger values of p, reaching a maximum somewhere between $p = 0.5$ and $p = 0.7$.

Chapter 2

Luby's Fast MIS Algorithm

2.1 Motivation

Luby's algorithm is a distributed algorithm for finding a maximal independent set (MIS) of a network's graph. It was initially introduced by Luby [4] in 1985. We will discuss an improved and simplified version based on the work of Métivier et al. [5].

The algorithm terminates in at most $O(\log n)$ rounds with high probability (n being the number of vertices), but on most non-random graphs it terminates even faster. It is unclear which properties of a graph dictate how well Luby's algorithm performs on it. We want to find a graph class on which the algorithm requires $\Theta(\log n)$ rounds, thus making this bound tight.

In order to better understand the algorithm and make arguing about it easier we also examine "z_{max}", the maximum distance from any node to an already determined MIS node after only one round of the algorithm.

2.2 Definitions

Let graphs and the graph classes $G_{n,p}$ and $Path_n$ be defined as in section 1.2. We will, however, only be looking at undirected and unweighted graphs in this chapter. Let $dist(u, v)$ be defined as the length of the shortest path between the vertices u and v.

Definition 2.1 (Independent Sets) *Given an undirected Graph $G = (V, E)$ an* independent set *is a subset of nodes $U \subseteq V$, such that no two nodes in U are adjacent (i.e. share an edge). An independent set is* maximal *if no node can be added without violating this constraint.* ◇

Definition 2.2 (W.h.p.) *We say a statement holds* w.h.p. *(with high probability) if it holds with probability at least $1 - \frac{1}{n^c}$ for every $c \geq 1$.* ◇

This notation is usually used with statements incorporating some kind of asymptotical bound such as $O(n)$, which allows masking the effect of any chosen c. It is assumed that c is chosen reasonably low in practice. Luby's algorithm was shown to terminate in at most $O(\log n)$ rounds w.h.p. by Métivier et al. [5].

The pseudocode for the version of Luby's algorithm we will discuss is given by algorithm 2.

Algorithm 2 Luby's Algorithm

The algorithm operates in rounds. It terminates once all nodes have terminated.

A single round is as follows:

1) Each node v chooses a random value $r(v) \in [0, 1]$ and sends it to its neighbors.

2) If $r(v) < r(w)$ for all neighbors $w \in N(v)$, node v enters the MIS and informs its neighbors.

3) If v or a neighbor of v entered the MIS, v terminates (v and all edges adjacent to v are removed from the graph), otherwise v enters the next round.

2.3 Luby Score

Definition 2.3 (Luby Score) *The* Luby score *of a graph or class of graphs is the average number of rounds Luby's algorithm takes to complete.* ◇

For $n \leq 1000$ this score takes on values in the range from 1 to about 5 — that is, Luby's algorithm completely devours any graph with 1000 vertices in only 5 rounds on average!

Relating the Luby score to classic graph-theoretical properties is more difficult than one might at first suspect.

Most of the time, a higher edge connectivity and a higher number of edges in general lead to a higher Luby score, but optimizing these values to the extreme makes us arrive at the complete graph — which Luby's algorithm never fails to completely process in a single round.

Similarly, a graph's diameter seems to correlate with its Luby score in that a lower diameter generally increases the Luby score; however, the complete graph has the lowest possible diameter, too.

There appears to be a certain "optimal" number of edges with regard to Luby's algorithm. On the one hand, the number needs to be high enough to obtain a certain connectivity, but on the other hand, the number needs to be low enough, to avoid losing too many vertices per vertex which enters the MIS.

Finally, a higher number of vertices allows for a higher Luby score than a lower number. This might appear somewhat obvious, but is certainly worth being mentioned.

2.3.1 Pessimal Graphs

In this section we will describe the "worst" graphs we found — that is, the graphs with the highest Luby score for a given number of vertices n. Algorithm 3 gives instructions on how to create "Bad Luby" graphs and algorithm 4 shows how to use two "Bad Luby" graphs to create a "Bad Luby×2" graph which has an even higher Luby score.

Algorithm 3 Generation of "Bad Luby" graphs

$V \Leftarrow \{1, \ldots, n\}, E \Leftarrow \emptyset$
Enumerate all possible edges $\{(u, v) \mid u, v \in V, u \neq v\}$ and process them in a random order.
for each edge (u, v) **do**
  if $dist(u, v) > 2$ according to (V, E) **then**
    $E \Leftarrow E \cup \{(u, v)\}$
  end if
end for
return (V, E)

Algorithm 4 Generation of "Bad Luby×2" graphs

$V \Leftarrow \{1, \ldots, n\}$
Generate two "Bad Luby" graphs $A = (V, E_A)$ and $B = (V, E_B)$ each with n vertices independently.
return $(V, E_A \cup E_B)$

Originally, the idea behind the single Bad Luby graph was to create a graph with the smallest reasonable diameter (namely 2) while using as few edges as possible and avoiding "central" vertices. This already beat the "toughest" $G_{n,p}$ graphs we had found. But it turned out that drastically increasing the number of edges increased the Luby Score even further.

Adding the edges of a third Bad Luby graph on top does not improve the Luby score any further but instead worsens it. It is our understanding that Bad Luby×2 happens to work so well because it gets close to the "optimal" number of edges.

However, $G_{n,p}$ graphs generated with p chosen such that the resulting number of edges matches the number of edges of a Bad Luby×2 graph with the same n consistently fail to reach the Luby scores which Bad Luby×2 graphs consistently reach. Hence, there must be some property other than the number of edges

about Bad Luby×2 which increases its score. This may be a topic for further research.

The SGT Toolbox has options to generate Bad Luby graphs as well as merge two such graphs as is necessary to obtain a Bad Luby×2 graph.

Figure 2.1 shows a comparison of the Luby scores of different graph classes for different numbers of vertices. The top diagram uses a linear scale, while the bottom one uses a logarithmic scale.

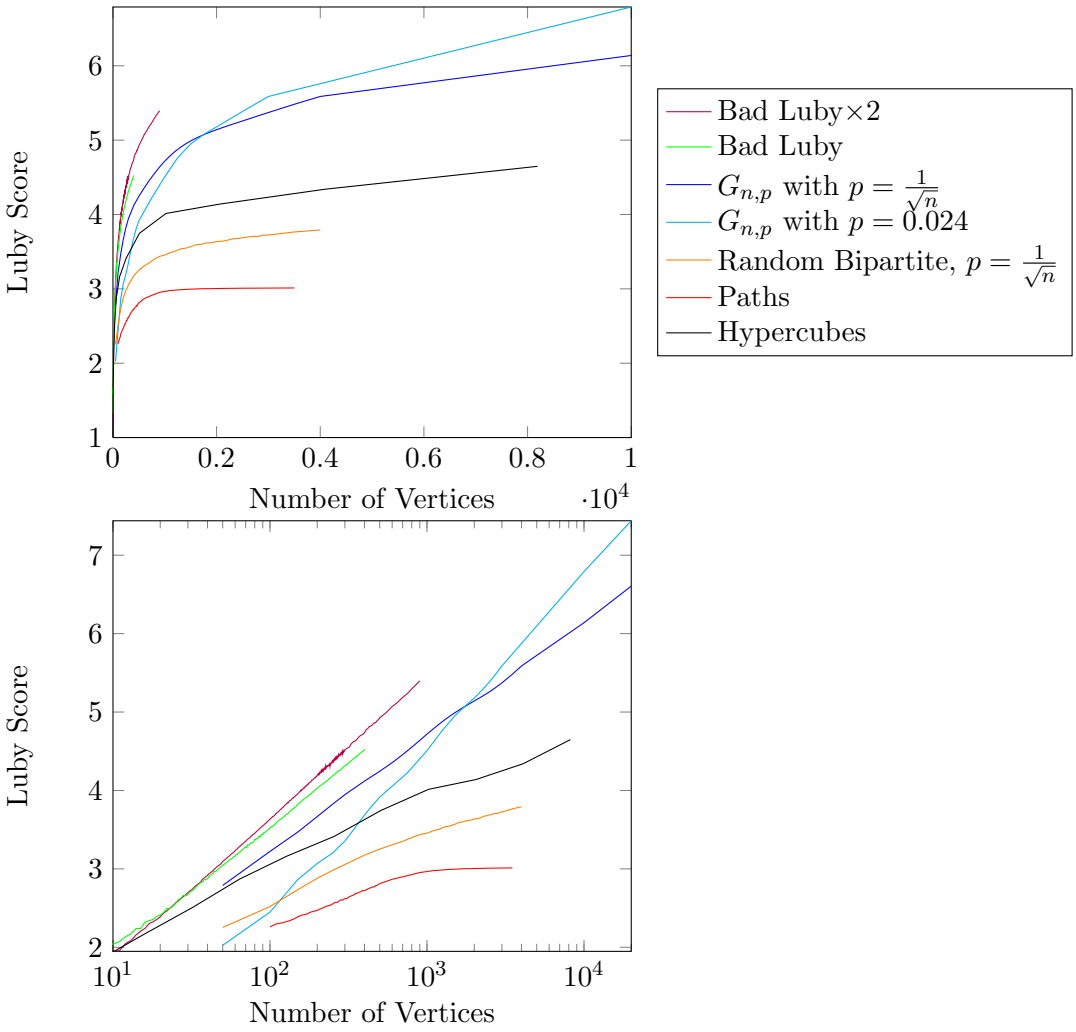

Figure 2.1: Comparison of Luby scores between graph classes

It is to be noted that the Luby scores for individual $G_{n,p}$ graphs actually fluctuate significantly. For this plot the Luby scores of about 200 graphs were averaged for each data point.

However, in turn, Bad Luby and Bad Luby×2 graphs exhibit a very stable

Luby score.

Note that the curve of $G_{n,p}$ with fixed $p = 0.024$ on the logarithmic scale mostly resembles a straight line but seems to be overlaid with a periodic wave. To verify that this was no coincidence, we increased the sampling rate of n for this curve to 50 up to $n = 500$ and to 250 up to $n = 3000$.

As the period of the wave appears to coincide with the rate at which the Luby score increases by 1, one might speculate that this is caused by some kind of peculiar "end game" forming around the completion of the final round of Luby's algorithm.

2.3.2 Lower Bound

While it has been shown that Luby's algorithm requires at most $O(\log n)$ rounds w.h.p. for any graph, fig. 2.1 suggests that certain graph classes may actually also exhibit a lower bound of $\Omega(\log n)$: $G_{n,p}$ both with fixed p and $p = \frac{1}{\sqrt{n}}$ and Bad Luby($\times 2$) are candidates with their close to logarithmic curves.

Bad Luby graphs are not easily argued about due to the convoluted nature of their generation. However, $G_{n,p}$ seem simple enough to reason about.

While we did not succeed in proving the lower bound for $G_{n,p}$, we will outline the proof attempted and show where it fails:

If we can show that (a) on average a constant percentage of all vertices of the given graph survives a single round of Luby's and (b) after each round of Luby's we again have a graph where every edge (between the remaining vertices) has the same probability to exist, this implies exactly the lower bound on the number of rounds we want to prove: a logarithmic one.

Part (a) seems easily shown: if every edge has the same fixed probability to exist, there is also a fixed probability for every vertex to have only one neighbor with at least two neighbors, which results in a chance of at least $\frac{1}{6}$ for the first vertex to survive. The probability for vertices to satisfy this criterion may actually rise as the number of vertices decreases, however, it never increases beyond a value around roughly $\frac{1}{3}$ for any n or p.

Part (b), however, is impossible to show, because it is not quite true! While the graphs which endured one round of Luby's algorithm seem to be close to proper $G_{n,p}$ graphs (i.e. have the same probability for each edge), they do not exactly agree for any probability p. A test over many thousands of instances of $G_{n,p}$ showed that after one round of Luby's the distribution of vertex degrees favored vertices with an average degree notably stronger than the vertex degree distribution in a proper $G_{n,p}$ (with adequatly set n and p to imitate the vertex and edge count).

2.4 An Upper Bound for z_{max}

In order to better understand Luby's algorithm we will now focus on a single round of it. In particular, we are interested in z_{max}, the maximum distance from any node to an MIS node.

Definition 2.4 ($z(v)$ and z_{max}) *We call the distance of a vertex v to the closest MIS vertex $z(v)$. We call the largest such value z_{max}.*

$$z(v) = \min\{dist(v,u) \mid u \in MIS\}$$
$$z_{max} = \max_{v \in V} z(v)$$ ◇

We want to show the following:

Theorem 2.5 (Upper Bound for z_{max} for Any Graph) *For any graph $G = (V, E)$ the following holds with high probability:*

$$z_{max} \leq O(\log n)$$ ◇

We first require the following lemma:

Lemma 2.6 *For any graph $G = (V, E)$:*

$$z(v) \leq O(\log n) \text{ w.h.p. for every vertex } v \in V \Rightarrow z_{max} \leq O(\log n) \text{ w.h.p.}$$ ◇

Proof (Lemma 2.6) Consider the following:

$$\begin{aligned} P[z_{max} > O(\log n)] &= P[\exists v \in V : z(v) > O(\log n)] \\ &\leq \sum_{v \in V} P[z(v) > O(\log n)] \qquad \text{(Union Bound)} \\ &\leq n \cdot \frac{1}{n^{c_{left}}} \\ &= \frac{1}{n^{c_{left}-1}} \\ \Rightarrow P[z_{max} \leq O(\log n)] &\geq 1 - \frac{1}{n^{c_{left}-1}} \end{aligned}$$

where c_{left} is the "w.h.p." constant of the left side of our implication. Let c_{right} be the "w.h.p." constant of the right side. We can then choose $c_{left} = c_{right} + 1$ to obtain

$$P[z_{max} \leq O(\log n)] \geq 1 - \frac{1}{n^{c_{right}}} \ .$$ ■

2.4.1 Path Graphs

Let us first consider the bound only for the class of path graphs.

Theorem 2.7 (Upper Bound for z_{max} for Path Graphs) *For any path graph $Path_n$ the following holds with high probability:*

$$z_{max} \leq O(\log n) \qquad \diamond$$

For the proof we will need the following technical lemma:

Lemma 2.8 *For any $n \geq 2$ and $c \geq 1$ the following holds:*

$$(5c \log n)!^2 \geq n^c \qquad \diamond$$

Proof (Lemma 2.8) Using Stirling's approximation we obtain:

$$\begin{aligned}
(5c \log n)!^2 &\geq \left(\sqrt{2\pi 5c \log n} \left(\frac{5c \log n}{e} \right)^{5c \log n} \right)^2 \\
&\geq \left(\left(\frac{5c \log n}{e} \right)^{5c \log n} \right)^2 \\
&= \left(\frac{5c \log n}{e} \right)^{10c \log n} \\
&= (5c \log n)^{10c \log n} n^{-10c}
\end{aligned}$$

Thus, all that remains to be shown is

$$\begin{aligned}
(5c \log n)^{10c \log n} n^{-10c} &\geq n^c \\
(5c \log n)^{10c \log n} &\geq n^{11c} \\
\log((5c \log n)^{10c \log n}) &\geq \log(n^{11c}) \\
10c \log n \log(5c \log n) &\geq 11c \log n \\
10 \log(5c \log n) &\geq 11 \\
\log(5c \log n) &\geq 1.1 \\
\log 5 + \log c + \log \log n &\geq 1.1
\end{aligned}$$

We know $\log 5 \geq 1.6$, $\log c \geq 0$ (for $c \geq 1$) and $\log \log n \geq -0.3666$ (for $n \geq 2$), which leaves us with

$$1.6 + 0 - 0.3666 \geq 1.1 \ . \qquad \blacksquare$$

Proof (Theorem 2.7) Consider the set of the random values in the x-neighborhood $N_x(v)$ of a vertex v to be an ordering on those $2x+1$ vertices. Let u be the vertex with the highest random value in $N_x(v)$. Note that $z(v) \geq x$ holds if and only if starting from u the random values of vertices in $N_x(v)$ decrease monotonically in both directions as you follow the path.

Assume u is given by being i hops from the vertex in $N_x(v)$ with the lowest index. Then, the number of different possible orderings for the remaining $2x$ vertices is $\binom{2x}{i}$ (the orderings on both sides of u are fixed, but there are "$2x$ choose i" ways to distribute the vertices among the sides). Thus, overall there are $\sum_{i=0}^{2x} \binom{2x}{i}$ orderings which satisfy $z(v) \geq x$.

Since there are $(2x+1)!$ possible orderings and every ordering has the same probability of occurring, we obtain:

$$\begin{aligned} P[z(v) \geq x] &= \frac{\sum_{i=0}^{2x} \binom{2x}{i}}{(2x+1)!} \\ &\leq \frac{\binom{2x}{x}(2x+1)}{(2x+1)!} \\ &= \frac{\binom{2x}{x}}{(2x)!} \\ &= \frac{(2x)!}{(2x)!x!x!} \\ &= \frac{1}{x!^2} \\ P[z(v) \leq x-1] &\geq 1 - \frac{1}{x!^2} \end{aligned}$$

Note that for vertices within $x-1$ hops of the first or last vertex of the path $P[z(v) \geq x]$ is lower than for regular vertices. This is because there is only a single ordering of the vertices in the x-neighborhood of v which allows this event to happen: the highest random value at the "end" of the graph and the others following in descending order. Hence, we can disregard these special cases for our bound.

If we now choose $x = 5c \log n$ and apply lemma 2.8 we obtain

$$P[z(v) \leq 5c \log n - 1] \geq 1 - \frac{1}{n^c}$$

(for $n \geq 2$ and $c \geq 1$). And since $5c \log n - 1 \in O(\log n)$ we arrive at

$$z(v) \leq O(\log n) \text{ holds w.h.p.}$$

Now we can apply lemma 2.6 to conclude the proof. ∎

2.4.2 General Graphs

We can actually even prove that the same bound holds for any graph:

Proof (Theorem 2.5) For every vertex v consider its i-neighborhoods N_i, their borders $B(N_i)$ and the events A_i and A, which are defined as follows:

$$N_i = \{u \in V \mid dist(v,u) \leq i\}$$
$$B(N_i) = \begin{cases} N_i \setminus N_{i-1} & \text{if } i > 0 \\ N_0 & \text{if } i = 0 \end{cases}$$
$$A_i : \text{“the maximum random value of all nodes in } N_i \text{ lies in } B(N_i)\text{”}$$
$$A : A_1 \wedge \cdots \wedge A_{3c\log n}$$

Note that the events A_i are independent. We have:

$$P[A] = P[A_1 \wedge \cdots \wedge A_{3c\log n}] = \prod_{i=1}^{3c\log n} P[A_i]$$
$$P[A_i] = \frac{|B(N_i)|}{|N_i|}$$

Now consider grouping the indices of the events into two sets I_{large} and I_{small}:

$$I_{large} = \left\{ i \;\middle|\; P[A_i] > \frac{1}{2} \right\}$$
$$I_{small} = \left\{ i \;\middle|\; P[A_i] \leq \frac{1}{2} \right\}$$

Note that $i \in I_{large}$ implies the size of neighborhood doubled in step i ($|N_i| > 2 \cdot |N_{i-1}|$). Since the graph only consists of $n = 2^{\log_2 n}$ vertices, this gives us an upper bound for $|I_{large}|$ and a lower bound for $|I_{small}|$:

$$|I_{large}| \leq \log_2 n$$
$$|I_{small}| \geq 3c\log n - \log_2 n = \left(3c - \frac{1}{\log 2}\right)\log n$$

From this we can obtain the following bound for $P[A]$:

$$P[A] \leq \left(\frac{1}{2}\right)^{(3c-\frac{1}{\log 2})\log n} = \left(n^{\log\frac{1}{2}}\right)^{3c-\frac{1}{\log 2}} = \frac{1}{n^{3c\log 2 - 1}}$$

Note that $3 \cdot \log 2 \geq 2.07$ and thus for every $c \geq 1$:

$$\frac{1}{n^{3c\log 2-1}} \leq \frac{1}{n^c}$$
$$P[A] \leq \frac{1}{n^c}$$

Now observe that "$z(v) > 3c \log n$" implies A, which gives us the following bounds:

$$P[z(v) > 3c \log n] \leq P[A]$$
$$P[z(v) \leq 3c \log n] \geq 1 - P[A]$$
$$P[z(v) \leq 3c \log n] \geq 1 - \frac{1}{n^c}$$

Applying lemma 2.6 concludes the proof. ∎

Bibliography

[1] McSherry, F.: Spectral partitioning of random graphs. In: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science. FOCS '01, Washington, DC, USA, IEEE Computer Society (2001) 529–

[2] Cheeger, J.: A lower bound for the smallest eigenvalue of the laplacian. Problems in analysis **195** (1970) 199

[3] Fruchterman, T.M.J., Reingold, E.M.: Graph drawing by force-directed placement. Softw. Pract. Exper. **21**(11) (November 1991) 1129–1164

[4] Luby, M.: A simple parallel algorithm for the maximal independent set problem. In: Proceedings of the seventeenth annual ACM symposium on Theory of computing. STOC '85, New York, NY, USA, ACM (1985) 1–10

[5] Métivier, Y., Robson, J., Saheb-Djahromi, N., Zemmari, A.: An optimal bit complexity randomized distributed mis algorithm. Distributed Computing **23** (2011) 331–340 10.1007/s00446-010-0121-5.