Logarithmic space, PCP and the hardness of approximation

Louis Esperet

CNRS, Université Grenoble Alpes

Graphs and complexity days, Lyon, 2025

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size.

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size.

What graph problems can you solve in logarithmic space ?

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size.

What graph problems can you solve in logarithmic space ?

- you can store a constant number of names of vertices in memory

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size.

What graph problems can you solve in logarithmic space ?

- you can store a constant number of names of vertices in memory
- you can store a logarithmic number of boolean values

Logarithmic space

Let L be the class of problems that can be solved in logarithmic space.

From the point of view of Turing machines this means that there are two tapes: the first one contains the input and is read-only, and the second one is the workable tape and has logarithmic size.

What graph problems can you solve in logarithmic space ?

- you can store a constant number of names of vertices in memory
- you can store a logarithmic number of boolean values
- you can store a constant number of polynomially bounded integers (for instance counters).

s, t-CONNECTIVITY

s, t-connectivity. Given a graph G and two vertices s and t of G, are s and t in the same connected component of G? Or equivalently is there a path from s to t in G?

s, t-CONNECTIVITY

s, t-connectivity. Given a graph G and two vertices s and t of G, are s and t in the same connected component of G? Or equivalently is there a path from s to t in G?

There is a simple randomized algorithm running in logarithmic space: perform a random walk W in G during $O(n^3)$ steps starting from s. If the walk W meets t then answer yes, otherwise answer no.

s, t-connectivity

s, t-connectivity. Given a graph G and two vertices s and t of G, are s and t in the same connected component of G? Or equivalently is there a path from s to t in G?

There is a simple randomized algorithm running in logarithmic space: perform a random walk W in G during $O(n^3)$ steps starting from s. If the walk W meets t then answer yes, otherwise answer no.

(it can be proved that in $O(n^3)$ steps, the random walk has visited all the vertices of the connected component of s with high probability).

s, t-CONNECTIVITY

There is also a simple deterministic algorithm for s, t-connectivity running in $O(\log^2 n)$ space.

s, t-CONNECTIVITY

There is also a simple deterministic algorithm for s, t-connectivity running in $O(\log^2 n)$ space.

Given a constant $D \geq 1$, let $S(D)$ be the space complexity of deciding whether two vertices u and v lie at distance at most D in G.

s, t-CONNECTIVITY

There is also a simple deterministic algorithm for s, t-connectivity running in $O(\log^2 n)$ space.

Given a constant $D \geq 1$, let $S(D)$ be the space complexity of deciding whether two vertices u and v lie at distance at most D in G.

If $D = 1$ we simply check whether u and v are adjacent.

s, t-connectivity

There is also a simple deterministic algorithm for s, t-connectivity running in $O(\log^2 n)$ space.

Given a constant $D \geq 1$, let $S(D)$ be the space complexity of deciding whether two vertices u and v lie at distance at most D in G.

If $D = 1$ we simply check whether u and v are adjacent.

If $D \geq 2$, for each vertex w, we test whether $d(u, w) \leq \lfloor D/2 \rfloor$ and $d(v, w) \leq \lceil D/2 \rceil$ (one after the other) and if both are true we answer yes.

s, t-CONNECTIVITY

There is also a simple deterministic algorithm for s, t-connectivity running in $O(\log^2 n)$ space.

Given a constant $D \geq 1$, let $S(D)$ be the space complexity of deciding whether two vertices u and v lie at distance at most D in G.

If $D = 1$ we simply check whether u and v are adjacent.

If $D \geq 2$, for each vertex w, we test whether $d(u, w) \leq \lfloor D/2 \rfloor$ and $d(v, w) \leq \lceil D/2 \rceil$ (one after the other) and if both are true we answer yes.

The space needed is $S(D) \leq O(\log n) + S(D/2)$, and thus

$$S(D) = O((\log n)(\log D)).$$

In particular, testing whether two vertices are at distance at most n takes $O(\log^2 n)$ space.

s, t-connectivity in L

Theorem (Reingold 2004). s, t-connectivity is in L

s, t-connectivity in L

Theorem (Reingold 2004). s, t-connectivity is in L

If G has degree at most $d = O(1)$ and logarithmic diameter, then s, t-connectivity can easily be solved in logarithmic space. It suffices to enumerate all the paths of length $O(\log n)$ starting from s (each such path can be described using $O(\log d \cdot \log n)$ bits).

s, t-connectivity in L

Theorem (Reingold 2004). s, t-connectivity is in L

If G has degree at most $d = O(1)$ and logarithmic diameter, then s, t-connectivity can easily be solved in logarithmic space. It suffices to enumerate all the paths of length $O(\log n)$ starting from s (each such path can be described using $O(\log d \cdot \log n)$ bits).

The goal is to transform G into a graph of bounded degree and logarithmic diameter, without changing the connected components.

s, t-connectivity in L

Theorem (Reingold 2004). s, t-connectivity is in L

If G has degree at most $d = O(1)$ and logarithmic diameter, then s, t-connectivity can easily be solved in logarithmic space. It suffices to enumerate all the paths of length $O(\log n)$ starting from s (each such path can be described using $O(\log d \cdot \log n)$ bits).

The goal is to transform G into a graph of bounded degree and logarithmic diameter, without changing the connected components.

We start by replacing every vertex v by a cycle a length $d(v)$, turning G into a subcubic graph.

s, t-connectivity in L

Theorem (Reingold 2004). s, t-connectivity is in L

If G has degree at most $d = O(1)$ and logarithmic diameter, then s, t-connectivity can easily be solved in logarithmic space. It suffices to enumerate all the paths of length $O(\log n)$ starting from s (each such path can be described using $O(\log d \cdot \log n)$ bits).

The goal is to transform G into a graph of bounded degree and logarithmic diameter, without changing the connected components.

We start by replacing every vertex v by a cycle a length $d(v)$, turning G into a subcubic graph.

For technical reasons we then attach self-loops to each vertex, turning G into a d-regular graph for some fixed d (constant, but not too small).

Expanders

A graph is a (c, d)-expander graph if it is d-regular and for any set S of at most $|V(G)|/2$ vertices, at least $c|S|$ edges of G connect S to $V(G) \setminus S$.

Expanders

A graph is a (c, d)-expander graph if it is d-regular and for any set S of at most $|V(G)|/2$ vertices, at least $c|S|$ edges of G connect S to $V(G) \setminus S$.

We will be interested in constructing such graphs with c and d fixed, and $|V(G)| \to \infty$.

Expanders

A graph is a (c, d)-expander graph if it is d-regular and for any set S of at most $|V(G)|/2$ vertices, at least $c|S|$ edges of G connect S to $V(G) \setminus S$.

We will be interested in constructing such graphs with c and d fixed, and $|V(G)| \to \infty$.

A crucial property of expander graphs is that they have logarithmic diameter.

Expanders

A graph is a (c, d)-expander graph if it is d-regular and for any set S of at most $|V(G)|/2$ vertices, at least $c|S|$ edges of G connect S to $V(G) \setminus S$.

We will be interested in constructing such graphs with c and d fixed, and $|V(G)| \to \infty$.

A crucial property of expander graphs is that they have logarithmic diameter.

Proving the existence of these graphs is easy (take a random d-regular graph), but constructing them (strongly) explicitly is difficult.

Expanders

A graph is a (c, d)-expander graph if it is d-regular and for any set S of at most $|V(G)|/2$ vertices, at least $c|S|$ edges of G connect S to $V(G) \setminus S$.

We will be interested in constructing such graphs with c and d fixed, and $|V(G)| \to \infty$.

A crucial property of expander graphs is that they have logarithmic diameter.

Proving the existence of these graphs is easy (take a random d-regular graph), but constructing them (strongly) explicitly is difficult.

Example. for $p \to \infty$, p prime, look at the graph with vertex set $\mathbb{Z}_p$, in which each x is adjacent to $x+1$, $x-1$, and x^{-1}.

Improving the expansion

We start with our d-regular graph with very poor expansion, and we repeat the following two steps $\log n$ times.

Improving the expansion

We start with our d-regular graph with very poor expansion, and we repeat the following two steps $\log n$ times.

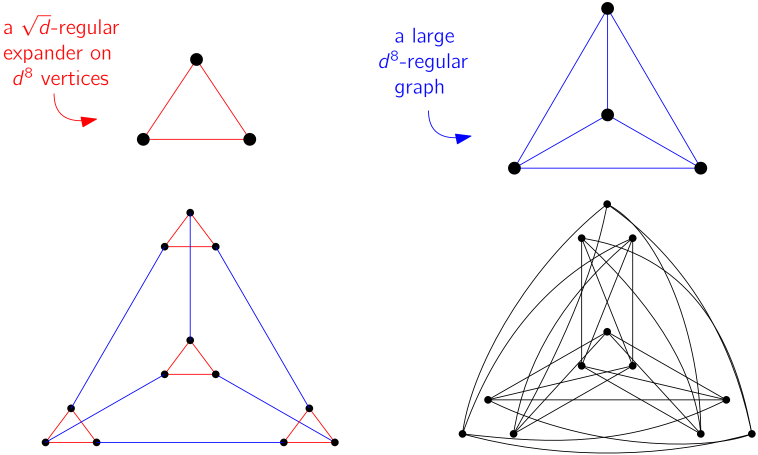
- take a small power of the current graph (say add edges between any two vertices at distance 8). The expansion improves significantly but the degree jumps to d^8.

Improving the expansion

We start with our d-regular graph with very poor expansion, and we repeat the following two steps $\log n$ times.

- take a small power of the current graph (say add edges between any two vertices at distance 8). The expansion improves significantly but the degree jumps to d^8.
- take the zig-zag product of our current graph with a small (fixed) expander. The expansion decreases slightly but the degree goes back to d.

The zig-zag product



Improving the expansion

We start with our d-regular graph with very poor expansion, and we repeat the following two steps $\log n$ times.

- take a small power of the current graph (say add edges between any two vertices at distance 8). The expansion improves significantly but the degree jumps to d^8.
- take the zig-zag product of our current graph with a small (fixed) expander. The expansion decreases slightly but the degree goes back to d.

Improving the expansion

We start with our d-regular graph with very poor expansion, and we repeat the following two steps $\log n$ times.

- take a small power of the current graph (say add edges between any two vertices at distance 8). The expansion improves significantly but the degree jumps to d^8.
- take the zig-zag product of our current graph with a small (fixed) expander. The expansion decreases slightly but the degree goes back to d.

After $\log n$ iterations the graph still has size polynomial in n, and it is a (c, d)-expander for some constant c. So the graph has logarithmic diameter and we can solve s, t-connectivity in this graph in logarithmic space.

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).
- k-vertex-disjoint paths: are there k internally vertex-disjoint paths between two vertices s, t?

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).
- k-vertex-disjoint paths: are there k internally vertex-disjoint paths between two vertices s, t?
- Given a graph, is there a cycle containing a given edge?

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).
- k-vertex-disjoint paths: are there k internally vertex-disjoint paths between two vertices s, t?
- Given a graph, is there a cycle containing a given edge?
- Is a given graph bipartite?

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).
- k-vertex-disjoint paths: are there k internally vertex-disjoint paths between two vertices s, t?
- Given a graph, is there a cycle containing a given edge?
- Is a given graph bipartite?
- Do two graphs have the same number of connected components?

Consequences

- A problem is in L if and only if it is log-space reducible to s, t-connectivity.
- L contains exactly the problems expressible in first-order logic with an additional transitive closure operator (in graphs: an operator that turns any connected component into a clique).
- k-vertex-disjoint paths: are there k internally vertex-disjoint paths between two vertices s, t?
- Given a graph, is there a cycle containing a given edge?
- Is a given graph bipartite?
- Do two graphs have the same number of connected components?
- Does a graph have an even number of connected components?

Conclusion on L

- It is known that $\mathsf{L} \subseteq \mathsf{P}$ (more generally problems that can be solved in space $s(n)$ can be solved in time $n \cdot 2^{s(n)+\log s(n)}$).

Conclusion on L

- It is known that $\mathsf{L} \subseteq \mathsf{P}$ (more generally problems that can be solved in space $s(n)$ can be solved in time $n \cdot 2^{s(n)+\log s(n)}$).
- It is not known whether $\mathsf{L} \neq \mathsf{P}$, or even whether $\mathsf{L} \neq \mathsf{NP}$.

Conclusion on L

- It is known that $L \subseteq P$ (more generally problems that can be solved in space $s(n)$ can be solved in time $n \cdot 2^{s(n)+\log s(n)}$).
- It is not known whether $L \neq P$, or even whether $L \neq NP$.
- Every problem in P is log-space reducible to the evaluation of a given circuit on a given input (which is in P). This implies that $L \neq P$ if and only if evaluating a given circuit on a given imput is not in L.

Conclusion on L

- It is known that $L \subseteq P$ (more generally problems that can be solved in space $s(n)$ can be solved in time $n \cdot 2^{s(n)+\log s(n)}$).
- It is not known whether $L \neq P$, or even whether $L \neq NP$.
- Every problem in P is log-space reducible to the evaluation of a given circuit on a given input (which is in P). This implies that $L \neq P$ if and only if evaluating a given circuit on a given imput is not in L.
- It is not known whether $RL = L$ (i.e. whether randomized logarithmic space can be completely derandomized).

The PCP theorem

Recall that NP is the class of problems for which positive instances have certificates that can be checked in polynomial time.

The PCP theorem

Recall that NP is the class of problems for which positive instances have certificates that can be checked in polynomial time.

The PCP theorem (1998) states that problems in NP have another useful type of certificates, still of polynomial size:

The PCP theorem

Recall that NP is the class of problems for which positive instances have certificates that can be checked in polynomial time.

The PCP theorem (1998) states that problems in NP have another useful type of certificates, still of polynomial size:

- The verifier starts by looking at the instance and using some randomized algorithm (which only relies on $O(\log n)$ random bits) decides a constant number of random locations.

The PCP theorem

Recall that NP is the class of problems for which positive instances have certificates that can be checked in polynomial time.

The PCP theorem (1998) states that problems in NP have another useful type of certificates, still of polynomial size:

- The verifier starts by looking at the instance and using some randomized algorithm (which only relies on $O(\log n)$ random bits) decides a constant number of random locations.
- The prover presents a certificate of polynomial size, and the verifier only checks the constant number of locations it chose, and based only on this constant number of bits, decides to accept or reject the proof.

The PCP theorem

Recall that NP is the class of problems for which positive instances have certificates that can be checked in polynomial time.

The PCP theorem (1998) states that problems in NP have another useful type of certificates, still of polynomial size:

- The verifier starts by looking at the instance and using some randomized algorithm (which only relies on $O(\log n)$ random bits) decides a constant number of random locations.
- The prover presents a certificate of polynomial size, and the verifier only checks the constant number of locations it chose, and based only on this constant number of bits, decides to accept or reject the proof.
- If the instance is positive, then the verifier must accept, and if the instance is negative, then the verifier must reject with probability at least $\frac{1}{2}$, for any certificate.

Hardness of approximation

A clique in a graph is a set of pairwise adjacent vertices, and the clique number of G, denoted by $\omega(G)$, is the maximum size of a clique in G.

Hardness of approximation

A clique in a graph is a set of pairwise adjacent vertices, and the clique number of G, denoted by $\omega(G)$, is the maximum size of a clique in G.

The PCP theorem (clique approximation version)

There exist constants $0 < a < b < 1$ such that given an n-vertex graph G for which

(1) $\omega(G) \leq an$, or

(2) $\omega(G) \geq bn$,

there is no polynomial time algorithm distinguishing between the two cases (1) and (2), unless $\mathrm{P} = \mathrm{NP}$.

Hardness of approximation

A clique in a graph is a set of pairwise adjacent vertices, and the clique number of G, denoted by $\omega(G)$, is the maximum size of a clique in G.

The PCP theorem (clique approximation version)

There exist constants $0 < a < b < 1$ such that given an n-vertex graph G for which
(1) $\omega(G) \leq an$, or
(2) $\omega(G) \geq bn$,
there is no polynomial time algorithm distinguishing between the two cases (1) and (2), unless $\mathrm{P} = \mathrm{NP}$.

In particular, we cannot approximate the clique number in polynomial time within a factor $c < b/a$ unless $\mathrm{P} = \mathrm{NP}$.

Hardness of approximation

A clique in a graph is a set of pairwise adjacent vertices, and the clique number of G, denoted by $\omega(G)$, is the maximum size of a clique in G.

The PCP theorem (clique approximation version)

There exist constants $0 < a < b < 1$ such that given an n-vertex graph G for which
(1) $\omega(G) \leq an$, or
(2) $\omega(G) \geq bn$,
there is no polynomial time algorithm distinguishing between the two cases (1) and (2), unless $\mathrm{P} = \mathrm{NP}$.

In particular, we cannot approximate the clique number in polynomial time within a factor $c < b/a$ unless $\mathrm{P} = \mathrm{NP}$.

(by c-approximation algorithm, we mean an algorithm that returns a value between $\omega(G)/c$ and $\omega(G)$.)

Connection between the two versions

Recall that in 3-SAT, we have a formula of the type $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_4 \vee \neg x_5) \wedge \ldots$, where there are 3 literals per clause and the x_i's are boolean variables, and the goal is to decide whether the formula can be satisfied (so all clauses have to be satisfied) by some assignment of values to the x_i's.

Connection between the two versions

The clique approximation version of the PCP theorem is equivalent to the statement that there is some $\varepsilon > 0$ such that if we are given a 3-SAT formula in which

(1) all clauses can be satisfied, or

(2) only a fraction of at most $1 - \varepsilon$ of the clauses can be satisfied,

then no polynomial time algorithm can make the distinction between (1) and (2), unless P = NP.

Connection between the two versions

The clique approximation version of the PCP theorem is equivalent to the statement that there is some $\varepsilon > 0$ such that if we are given a 3-SAT formula in which

(1) all clauses can be satisfied, or

(2) only a fraction of at most $1 - \varepsilon$ of the clauses can be satisfied,

then no polynomial time algorithm can make the distinction between (1) and (2), unless $\mathsf{P} = \mathsf{NP}$.

Now any problem in NP can be reduced to this gap version of 3-SAT, and this version has a simple probabilistically checkable certificate, consisting of the values of the variables x_i.

Connection between the two versions

The clique approximation version of the PCP theorem is equivalent to the statement that there is some $\varepsilon > 0$ such that if we are given a 3-SAT formula in which

(1) all clauses can be satisfied, or

(2) only a fraction of at most $1 - \varepsilon$ of the clauses can be satisfied,

then no polynomial time algorithm can make the distinction between (1) and (2), unless $\mathsf{P} = \mathsf{NP}$.

Now any problem in NP can be reduced to this gap version of 3-SAT, and this version has a simple probabilistically checkable certificate, consisting of the values of the variables x_i.

The verifier just checks a constant number of clauses selected uniformy at random and says that the formula is satisfiable if all these clauses are satisfied.

Gap amplification

The PCP theorem

There exist constants $0 < a < b < 1$ such that given an n-vertex graph G for which

(1) $\omega(G) \leq an$, or

(2) $\omega(G) \geq bn$,

there is no polynomial time algorithm distinguishing between the two cases (1) and (2), unless $\mathrm{P} = \mathrm{NP}$.

In particular, we cannot approximate the clique number in polynomial time within a factor $c < b/a$ unless $\mathrm{P} = \mathrm{NP}$.

(recall that by c-approximation algorithm, we mean an algorithm that returns a value between $\omega(G)/c$ and $\omega(G)$).

Gap amplification

The PCP theorem easily implies that for any $c > 1$, we cannot approximate the clique number in polynomial time within a factor c unless $\mathsf{P} = \mathsf{NP}$.

Gap amplification

The PCP theorem easily implies that for any $c > 1$, we cannot approximate the clique number in polynomial time within a factor c unless $\mathrm{P} = \mathrm{NP}$.

Proof. Consider $G * k$, the graph whose vertices are the k-tuples of vertices of G, with adjacency between two k-tuples if their union is a clique of G.

Gap amplification

The PCP theorem easily implies that for any $c > 1$, we cannot approximate the clique number in polynomial time within a factor c unless $\mathrm{P} = \mathrm{NP}$.

Proof. Consider $G * k$, the graph whose vertices are the k-tuples of vertices of G, with adjacency between two k-tuples if their union is a clique of G.

It is not hard to check that $\omega(G * k) = \omega(G)^k$.

Gap amplification

The PCP theorem easily implies that for any $c > 1$, we cannot approximate the clique number in polynomial time within a factor c unless $\mathrm{P} = \mathrm{NP}$.

Proof. Consider $G * k$, the graph whose vertices are the k-tuples of vertices of G, with adjacency between two k-tuples if their union is a clique of G.

It is not hard to check that $\omega(G * k) = \omega(G)^k$.

Now any c_1-approximation algorithm $\mathcal{A}_1$ for the clique number can be turned into a c_2-approximation algorithm $\mathcal{A}_2$ with $1 < c_2 < c_1$ by defining $\mathcal{A}_2(G) = (\mathcal{A}_1(G * k))^{1/k}$ for sufficiently large (but constant) k, because:

$$\omega(G)/c_2 \leq \omega(G)/c_1^{1/k} = (\omega(G * k)/c_1)^{1/k} \leq \mathcal{A}_2(G) \leq \omega(G),$$

Gap amplification II

Theorem

There is $\varepsilon > 0$ such that there is no polynomial time n^ε-approximation algorithm for the clique number in n-vertex graphs, unless $\mathrm{P} = \mathrm{NP}$.

Gap amplification II

Theorem

There is $\varepsilon > 0$ such that there is no polynomial time n^ε-approximation algorithm for the clique number in n-vertex graphs, unless $\mathrm{P} = \mathrm{NP}$.

Let G be an n-vertex graph, and let F be a (c, d)-expander graph on the same vertex set as G. For some integer t, we define a new graph H whose vertices are the $t+1$-vertex walks $x_0, \ldots, x_t$ in F, and in which two walks are adjacent if and only if their union is contained in some clique of G.

Gap amplification II

Theorem

There is $\varepsilon > 0$ such that there is no polynomial time n^ε-approximation algorithm for the clique number in n-vertex graphs, unless $\mathrm{P} = \mathrm{NP}$.

Let G be an n-vertex graph, and let F be a (c, d)-expander graph on the same vertex set as G. For some integer t, we define a new graph H whose vertices are the $t + 1$-vertex walks $x_0, \ldots, x_t$ in F, and in which two walks are adjacent if and only if their union is contained in some clique of G.

Note that H has $N = nd^t$ vertices, which is $n^{O(1)}$ when $t = O(\log n)$.

Gap amplification II

Theorem

There is $\varepsilon > 0$ such that there is no polynomial time n^ε-approximation algorithm for the clique number in n-vertex graphs, unless $\mathrm{P} = \mathrm{NP}$.

Let G be an n-vertex graph, and let F be a (c, d)-expander graph on the same vertex set as G. For some integer t, we define a new graph H whose vertices are the $t+1$-vertex walks $x_0, \ldots, x_t$ in F, and in which two walks are adjacent if and only if their union is contained in some clique of G.

Note that H has $N = nd^t$ vertices, which is $n^{O(1)}$ when $t = O(\log n)$.

We claim that

- if $\omega(G) \leq an$, then $\omega(H) \leq (a + o(1))^t N$.
- if $\omega(G) \geq bn$, then $\omega(H) \geq (b - o(1))^t N$.

Gap amplification II

Theorem

There is $\varepsilon > 0$ such that there is no polynomial time n^ε-approximation algorithm for the clique number in n-vertex graphs, unless $\mathrm{P} = \mathrm{NP}$.

Let G be an n-vertex graph, and let F be a (c, d)-expander graph on the same vertex set as G. For some integer t, we define a new graph H whose vertices are the $t+1$-vertex walks $x_0, \ldots, x_t$ in F, and in which two walks are adjacent if and only if their union is contained in some clique of G.

Note that H has $N = nd^t$ vertices, which is $n^{O(1)}$ when $t = O(\log n)$.

We claim that

- if $\omega(G) \leq an$, then $\omega(H) \leq (a + o(1))^t N$.
- if $\omega(G) \geq bn$, then $\omega(H) \geq (b - o(1))^t N$.

With $t = \log n$, the ratio between the two bounds is close to $(b/a)^t$ which is polynomial in n (and N).

Random walks in expander graphs

We only need to show that

- if $\omega(G) \leq an$, then $\omega(H) \leq (a + o(1))^t N$.
- if $\omega(G) \geq bn$, then $\omega(H) \geq (b - o(1))^t N$.

Random walks in expander graphs

We only need to show that

- if $\omega(G) \leq an$, then $\omega(H) \leq (a + o(1))^t N$.
- if $\omega(G) \geq bn$, then $\omega(H) \geq (b - o(1))^t N$.

This is an immediate consequence of the following theorem, which says that random walks in expander graphs behave like random sampling.

Random walks in expander graphs

We only need to show that

- if $\omega(G) \leq an$, then $\omega(H) \leq (a + o(1))^t N$.
- if $\omega(G) \geq bn$, then $\omega(H) \geq (b - o(1))^t N$.

This is an immediate consequence of the following theorem, which says that random walks in expander graphs behave like random sampling.

Random walks in expander graphs

Let G be a (c, d)-expander graph on n vertices.
Suppose we take a vertex x_0 uniformly at random in G, and then perform a random walk $x_0, \ldots, x_t$ of length t starting at x_0.
Then for any subset $S \subseteq V(G)$, the probability that $x_0, x_1, \ldots, x_t$ are all in S is at most $(|S|/n + o(1))^t$ and at least $(|S|/n - o(1))^t$.