CS 710: Complexity Theory 10/3/2017

Kernelization Lower Bounds

Instructor: Dieter van Melkebeek Scribes: Sam Lemley and Sam Vinitsky

1 Introduction

In the previous lecture, we discussed the notion of advice, and proved some results about how helpful it could be in solving problems. In particular, we proved the Karp-Lipton Theorem, which states that if NP $\subseteq$ P/poly, then the polynomial hierarchy collapses to the second level, which is unlikely. We stated the following theorem, which will be useful later in this lecture. The proof is left as an exercise.

Theorem 1. coNP $\subseteq$ NP/poly $\implies$ PH $= \Sigma_3^{\mathrm{P}}$.

We also introduced the notion of a selector for a language L, which is a function $f(x_0, x_1)$ that, if either x_0 or x_1 is in L, returns that element. This means the only way f returns an element not in L is if both inputs are not in L. We noted that if SAT has a selector, then P = NP. We also proved the following theorem:

Theorem 2. *L has a selector* $\implies$ $L \in$ P/poly.

Recall that the proof idea was to find an element y that was not in L such that $f(x, y) = y$ for many values of $x \in \overline{L}$ (we say that y dominates x). We can then give the machine y as advice. In fact, we can construct polynomially many y_i's in $\overline{L}$ such that every $x \in \overline{L}$ is dominated by some y_i, and then give the whole set $\{y_i\}$ as advice. In order to test whether a given x is in L, we merely need to compute $f(x, y_i)$ for each y_i. If any of these functions return y_i, then $x \in \overline{L}$, and otherwise $x \in L$.

In this lecture, we will use the ideas from the previous lecture in a more general manner in order to prove lower bounds for the kernelization of vertex cover.

2 Compression

We begin our discussion of kernelization by formulating a generalization of selectors. We first need two definitions:

Definition 1 (compression to length ℓ). *We say that a language L can be compressed to length ℓ if there exists some polynomial time mapping reduction f from L to some L' such that $|f(x)| \leq l(x) < |x|$.*

Definition 2 (OR(L)). *Given some language L, then* $\mathrm{OR(L)} = \{\langle x_1, ..., x_t \rangle \mid \bigvee_{i=1}^{t} (x_i \in L)\}$.

In other words, OR(L) consists of all tuples (of any size) such that *some* member of the tuple is in L. We will use s to denote the length of each tuple element x_i, and we use n to denote the length of the entire tuple. That is, $s = |x_i|$, and $n = |\langle x_1, ..., x_t \rangle|$. Note that n is about st.

By applying compression to this language OR(L) obtained from L, we get a generalization of a selector for L. This is formalized in the following claim.

Proposition 1. If L has a selector, then OR(L) can be compressed to length s.

Proof. Suppose L has a selector f. We can use f to compress some instance $\langle x_1, ..., x_n \rangle$ of OR(L) to length s. We first compute $f(x_1, x_2) = y_2$, and then iteratively compute $y_{i+1} = f(y_i, x_i)$ for $i \in \{2, ..., t\}$. Finally, let $x^* = y_t$. If at least one x_i is in L, then every y_j with $j \geq i$ will be in L by the definition of selectors. So if there is some $x_i \in L$, then $x^* = y_t \in L$. Conversely, if none of the x_i are in L, then none of the y_i will be in L, and thus $x^* = y_t \notin L$. This means that $x^* \in L$ if and only if some x_i is in L, which is the case if and only if $\langle x_1, ..., x_n \rangle \in \text{OR(L)}$.

Note that x^* only takes polynomial time to compute, since we are doing a linear number of comparisons $f(x_i, x_j)$, and f is polynomial time computable. Thus we have a polynomial time mapping reduction from OR(L) to L such that the reduced instance x^* has size s. This means OR(L) can be compressed to length s. ∎

We will now prove a theorem about compression that will be instrumental in our later proof of the kernelization lower bound for vertex cover.

Theorem 3. *If* OR(L) *can be compressed to length* $l(s, t) = O(t)$ *for* $t \geq s^c$ *(for some constant* c*), then* $\overline{L} \in \text{NP/poly}$.

Note: Since we require that $l(s, t) = O(t)$, this means the amortized length per instance x_i is *constant*.

Proof (of Theorem 3). The idea of this proof is similar to the proof of Theorem 2. By assumption, we have a polynomial time mapping reduction f from OR(L) to some language L'. Note that if a tuple $\langle x_1, ..., x_n \rangle$ gets mapped to $y \in \overline{L'}$, then every element in the tuple must be in $\overline{L}$. We say that such a y vouches for the non-membership of x_i in $\overline{L}$. We wish to find some y that vouches for as many elements as possible, and give this as advice (as in the proof of Theorem 2). In order to do so, let us define a set of all the elements for which y can vouch. Let $V(y)$ denote the set of all elements that are in some tuple that maps to y under f. This is,

$$V(y) = \{x \mid (\exists j \in [t])(\exists x_1, ..., x_{j-1}, x_{j+1}, ..., x_t) \text{ s.t. } f(\langle x_1, ..., x_{j-1}, x, x_{j+1}, ..., x_t \rangle) = y\}$$

We can determine membership in $V(y)$ in nondeterministic polynomial time: guess some assignment to $x_1, ..., x_{j-1}, x_{j+1}, ..., x_t$, and verify that $f(\langle x_1, ..., x_{j-1}, x, x_{j+1}, ..., x_t \rangle) = y$ (f is polynomial time computable by assumption).

We are going to focus on strings x of a given length n. We want to find some y that vouches for many of the elements x of length n. Let X_0 denote all n-bit strings that are not in L. Let $V_0(y)$ denote the tuples of length n that map to y. Formally,

$$X_0 = \{0, 1\}^n \cap \overline{L}$$
$$V_0(y) = \{0, 1\}^n \cap V(y) = X_0 \cap V(y)$$

We wish to find some y_0^* such that the size of $V_0(y_0^*)$ is some constant fraction of the size of X_0. Then repeated application of this method will produce a sequence of $y_0^*, y_1^*, ..., y_k^*$ such that every element in X_0 is vouched for by some y_i^*. The following claim proves that such a y_0^* exists: (Note that X_0^t denotes the set of tuples of length t whose elements are in X_0).

Claim 1. *There exists some y_0^* such that the number of tuples from X_0^t that get mapped to y_0^* under f is $\geq \frac{|X_0^t|}{2^{at}}$ for some constant a.*

Proof. Let us compute the average number of tuples from X_0^t that map to a given y under f. This is equal to the number of tuples in X_0^t divided by the number of possible y's we can map to. The length of the output y is bounded by $l(s,t)$ by definition of our mapping. But $l(s,t) = O(t) \leq at$ for some constant $a > 0$. Thus $|y| \leq at$, and so the number of possible y is the number of binary strings of length at, which is 2^{at}. This means that the average number of tuples that map to a given y is $\geq \frac{|X_0^t|}{2^{at}}$. Clearly there must exist some y_0^* such that the number of strings that map to y_0^* is greater than or equal to the average. ∎

We have shown that some constant fraction of the tuples X_0^t get mapped to y_0^*, but we need to show that some constant fraction of X_0 is in some tuple that gets mapped to y_0^*. We furnish a new definition:

$$\tilde{V}_0(y) = \{x \in X_0 \mid (\exists j \in [t])(\exists x_1, ..., x_{j-1}, x_{j+1}, ..., x_t \in X_0) \text{ s.t. } f(\langle x_1, ..., x_{j-1}, x, x_{j+1}, ..., x_t\rangle) = y\}$$

Note that $\tilde{V}_0(y) \subseteq V_0(y)$, since we are just restricting the elements x_j to be in X_0. Thus $|\tilde{V}_0(y)| \leq |V_0(y)|$

If we have a tuple from X_0^t with some component $x \in \tilde{V}_0(y_0^*)$, then every component of the tuple must also be in $\tilde{V}_0(y_0^*)$. This means that:

$$\frac{|X_0^t|}{2^{at}} \leq \text{the number of tuples from } X_0 \text{ that map to } y_0^* \leq |\tilde{V}_0(y_0^*)|^t$$

And thus by taking the t^{th} root, we get $\frac{|X_0|}{2^a} \leq |\tilde{V}_0(y_0^*)| \leq |V_0(y_0^*)|$. This means that y_0^* vouches for some constant fraction of X_0, as desired.

We can repeat this process with the remaining, unvouched for members of X_0. That is, let $X_1 = X_0 - V_0(y_0^*)$. Then $|X_1| = |X_0| - |V_0(y_0^*)| \leq (1 - \frac{1}{2^a}) \cdot |X_0|$. We can then produce an element y_1^* that vouches for some constant fraction of X_1. Repeating this k times gives us the inequality $|X_k| \leq (1 - \frac{1}{2^a})^k \cdot |X_0|$, and a sequence of $y_1^*, ..., y_k^*$ that together vouch for everything not in $X_0 - X_k$. We want to find some value of k such that the final iteration contains no elements, i.e. $|X_k| < 1$. If $|X_k| \leq (1 - \frac{1}{2^a})^k \cdot |X_0| < 1$, we will cover all the elements of X_0 in k repetitions. We can then compute a suitable value of k that satistfies this inequality.

$$\left(1 - \frac{1}{2^a}\right)^k \cdot |X_0| < 1 \tag{1}$$

$$k \cdot \log(1 - \frac{1}{2^a}) + \log(|X_0|) < 0 \tag{2}$$

$$\frac{\log(|X_0|)}{-\log(1 - 1/2^a)} < k \tag{3}$$

And so choosing $k > \frac{\log(|X_0|)}{-\log(1-1/2^a)}$ suffices. Note that $k = O(n)$, since $|X_0| \leq 2^n$.

This means that if we take $k = O(n)$ many repetitions, there are polynomially many y_i^* that together vouch for all of X_0. We use this sequence of y_i^*'s as advice for our nondeterministic machine. In order to test whether $x \in \overline{L}$, just nondeterministically guess a tuple $\langle x_1, ..., x_{j-1}, x, x_{j+1}, ...x_t\rangle$ containing x, and then test whether $f(\langle x_1, ..., x_{j-1}, x, x_{j+1}, ...x_t\rangle) = y_i^*$ for some y_i^* (there are only polynomially many of them). If we find some such y_i^*, then we accept x. This is clearly a nondeterministic polynomial machine that decides $\overline{L}$ with polynomial advice. Thus $\overline{L} \in$ NP/poly.■

Note that Theorem 3 immediately implies the following corollary:

Corollary 1. *If* OR(3SAT) *can be compressed to length* $l(s,t) = O(t)$ *for* $t \geq s^c$*, then* PH $= \Sigma_3^{\mathrm{P}}$.

Proof. Suppose OR(3SAT) can be compressed to length $l(s,t) = O(t)$ for $t \geq s^c$. Then by Theorem 3, $\overline{\text{3SAT}} \in$ NP/poly, and thus coNP $\subseteq$ NP/poly. But then Theorem 1 tells us that PH $= \Sigma_3^{\mathrm{P}}$. ■

3 Kernelization for Vertex Cover

In this section, we introduce methods to analyze the difficulty of NP-hard problems by making use of paramterized complexity concepts. In parameterized complexity, we measure the complexity of a problem not only as a function of the input size, but also as a function of one or more additional parameters denoted as k. Our goal is to achieve a running time of $O(h(k) \cdot n^c)$. If we are able to achieve this running time, we can describe our algorithm as *fixed-parameter tractable* for L. We formalize the definition of fixed-parameter tractibility below:

Definition 3 (fixed-parameter tractable). *A parameterized problem is fixed-parameter tractable (FPT) if there exists some algorithm* $\mathcal{A}$ *whose running time is* $O(h(k) \cdot n^c)$ *for some constant* c *and some function* $h : \mathbb{N} \to \mathbb{N}$*. A problem* $\mathcal{A}$ *is fixed-parameter tractable (FPT) if there exists some* k *such that* $\mathcal{A}$ *runs in time* $O(h(k) \cdot n^c)$ *for some constant* c *and some function* $h : \mathbb{N} \to \mathbb{N}$*.*

In practice these parameters k only take on small values, and thus the corresponding function, denoted as h, is allowed to be quite large. Even if $h(k)$ is a very large value, it only contributes a multiplicative factor to our running time.

3.1 A Kernelization Algorithm for Vertex Cover

One method of constructing a fixed parameter tractible algorithm is through the use of the *kernelization*. The idea is that we wish to map instances of some language L to "small enough" instances of that same language. We take some instance x of a language L, and reduce it to another instance $f(x)$ of "small enough" size $g(k)$. We can then solve the reduced instance $f(x)$ in time $h(k)$ (for example, by brute force). We formalize the notation here:

Definition 4 (Kernelization). *We say that* f *is a kernelization of a parameterized language* L *if* f *is a polynomial time mapping reduction from* L *to itself such that* $|f(x)| \leq g(k)$ *for some* $g : \mathbb{N} \to \mathbb{N}$*.*

Note that the size of the reduced instance is bounded by some function of the parameter k. This formalizes our desired notion of a "small enough" instance.

Vertex Cover (VC) is a problem that naturally admits a fixed-parameter tractable algorithm. Recall that VC is the problem of determining for a given graph $G = (V, E)$ whether there exists a set $S \subseteq V$ of vertices such that each edge in E is adjacent to some member of S. We can trivially determine if there exists a vertex cover of size k by examining every possible subset of size k. Since k is not a constant in this algorithm, there are $O(n^k)$ such subsets. Thus the running time of this trivial algorithm is not $O(h(k) \cdot n^c)$ revealing that the algorithm is not fixed-parameter tractible. However, fixed-parameter tractable algorithms do exist for VC. We will now prove the intuition that a kernelization implies an FPT algorithm.

Claim 2. *If L has a kernelization f, then L also has a fixed-parameter tractable algorithm $\mathcal{A}$.*

Proof. We explicitly describe the algorithm $\mathcal{A}$:

1. On input x, compute the kernelization $f(x)$.
2. Use any algorithm to determine whether $f(x) \in L$. Output whatever this algorithm outputs.

Our algorithm declares that x is in L if and only if $f(x) \in L$. But since f is a mapping reduction, $x \in L \iff f(x) \in L$, and thus our algorithm always correctly decides x. Step 1 takes polynomial time, since f was a polynomial mapping reduction. Step 2 takes time $h(|f(x)|)$, for some function $h : \mathbb{N} \to \mathbb{N}$. But $|f(x)| \leq g(k)$ for some function $g : \mathbb{N} \to \mathbb{N}$ since f is a kernelization, and so $h(|f(x)|) \leq h(g(k)) = p(k)$, where $p = h \circ g$. So step 2 takes time less than $p(k)$ for some $p : \mathbb{N} \to \mathbb{N}$. Thus the entire algorithm takes time $O(p(k) + n^c)$, which is clearly $O(p(k) \cdot n^c)$. Therefore this algorithm is fixed-parameter tractable. ∎

We have shown above that if a problem has a kernelization of size $g(k)$, it also has a fixed-parameter tractable algorithm. In order to prove the VC has a fixed-parameter algorithm, we will explicitly describe a kernelization.

Theorem 4. VC *has a kernelization with $O(k^2)$ edges.*

Proof. We explicitly describe the kernelization. If any vertex v in G has more than k neighbors, it must be in our vertex cover. (Otherwise we would need to pick all of its neighbors, of which there are more than k, resulting in a vertex cover of size greater than k.) If there were more than k such vertices, then there can be no vertex cover of size k. We can thus assume our graph had $\ell \leq k$ vertices of degree more than k. We have removed all ℓ of these vertices (and their incident edges), resulting in a graph G' on $k - \ell$ vertices. What remains is the find a vertex cover of size $k - \ell$ in this graph G'.

Every vertex in G' has degree at most k, because otherwise it would have been removed from G at the beginning. This means that any vertex in G' can cover at most k edges. This means that any vertex cover of size $k - \ell$ of G' will cover at most $k(k - \ell)$ edges. If G' has more than this number of edges, we know that no vertex cover exists. Otherwise, G' has no more than $k(k - \ell) \leq k^2$ edges that we need to cover with $k - \ell$ vertices. This is a problem in VC with $O(k^2)$ edges. Thus this entire algorithm is a kernelization that reduces a problem with n vertices to a problem on $O(k^2)$ edges, and thus $2k^2 = O(k^2)$ vertices. ∎

The best known kernelization algorithm for VC produces a kernel with $O(k)$ vertices and $O(k^2)$ edges. The natural question is whether this is the best we can do. That is, does there exists some kernelization with fewer vertices or edges? In the next section, we will use all the tools at our disposal to prove that no kernelization of VC can have $k^{2-\epsilon}$ edges for some constant ϵ unless the polynomial time hierarchy collapses to the third level.

3.2 A Lower Bound for Kernelizations of VC

We will prove the following lower bound for the number of edges in the kernelization of VC:

Theorem 5. *If VC has a kernel with $k^{2-\epsilon}$ edges (for some $\epsilon > 0$), then* coNP $\subseteq$ NP/poly.

This theorem taken with Theorem 1 immediately imply the following corollary:

Corollary 2. VC *has no kernels with $k^{2-\epsilon}$ edges for any $\epsilon > 0$ unless* PH $= \Sigma_3^{\mathrm{P}}$.

This means that unless the polynomial hierarchy collapses (an unlikely assumption), VC has no kernelization with $\leq k^{2-\epsilon}$, and thus our kernelization in the proof of Theorem 4 is tight (in the number of edges).

We will assume that VC has a kernel with $k^{2-\epsilon}$ edges, and use this fact to prove that OR(3SAT) has a compression of short length. We can then use Theorem 3 to show that $\overline{3SAT} \in$ NP/poly, which in turn proves that coNP $\subseteq$ NP/poly, and thus that the polynomial time hierarchy collapses to the third level (by Theorem 1).

Proof (of Theorem 5). Suppose that VC has a kernel with $k^{2-\epsilon}$ edges for some $\epsilon > 0$. We use this kernel to establish a compression algorithm for 3SAT. Rather than dealing with VC directly, we will instead consider the closely related problem of Clique. Recall that Clique consists of pairs (G, k) such that the graph G has a clique of size k. A clique is a subset of vertices with all possible edges present.

We make use of the following fact about the relationship between VC and Clique. Let us denote by $\overline{E}$ the set of edges that are not in E.

Claim 3. *If S is a set of edges in a graph $G = (V, E)$, then S is a vertex cover for G if and only if $\overline{S}$ is a clique in $G' = (V, \overline{E})$.*

The hypothesis that VC has a kernel of $k^{2-\epsilon}$ edges implies that Clique has a compression of size $O(n^{2\epsilon} \cdot \log n)$, where n is the number of vertices. Suppose we have instance (G', k') of Clique. We can transform this into an instance of $(G', n - k')$ of VC. By assumption, this instance of VC has a kernelization of size $(n - k')^{2-\epsilon} \leq n^{2-\epsilon}$. This process exactly describes a compression of Clique into an instance of VC of length $n^{2-\epsilon}$. This means Clique can be compressed to length $O(n^{2-\epsilon} \log n) = O(n^{2-\epsilon_1})$ for any $\epsilon_1 < \epsilon$.

We will now exploit this compression of Clique to develop a compression for OR(3SAT). If OR(3SAT) has a compression to length $l(s, t) = O(t)$ for $t \geq s^c$, then by Theorem 3, coNP $\subseteq$ NP/poly. We can represent an instance of OR(3SAT), as a tuple $\langle \varphi_1, \varphi_2, ..., \varphi_t \rangle$ of t 3CNF formulas. Note that since $|\varphi_i| = s$, the number of variables in φ_i and the number of clauses in φ_i are both at most s. W.l.o.g. we assume that there are exactly s clauses (if not, just add duplicates).

Using the standard reduction from 3SAT to Clique (see Sipser p.284 [2]), we can transform each 3SAT instance φ_i into an equivalent Clique instance (G_i, s). By applying this reduction to each φ_i in the tuple $\langle \varphi_1, \varphi_2, ..., \varphi_t \rangle$, we can reduce this tuple to a new tuple $\langle (G_0, s), (G_1, s), ...(G_t, s) \rangle$ that is an instance of OR(Clique). This gives us a polynomial time reduction from OR(3SAT) to OR(Clique).

Now we would like to reduce this tuple of Clique instances to a single Clique instance (G, s) with few vertices. If we can do so, then we know 3SAT also has a compression of a small length by our reduction above. This will allow us to finish our proof by applying Theorem 3.

Method 1: Disjoint Union

A first attempt to produce a single instance (G, s) of Clique from the tuple of instances (G_i, s) is by taking the disjoint union of the vertices of all of the graphs G_i. In establishing the disjoint union, we note that each G_i has $3s$ vertices and we note that there are t graphs. Thus, the disjoint union has $3st$ vertices. By applying our compression, we get a compression length of $n^{2-\epsilon} = O((st)^{2-\epsilon})$.

This is not quite good enough. To apply Theorem 3, the compression length must be $O(t)$ for a sufficiently large t (polynomial in s). Even if we set $\epsilon = 1$ in $O((st)^{2-\epsilon})$ we will not be able to reduce this to $O(t)$ as s will always be a polynomial fraction. Since we cannot use disjoint union to reduce the compression length to $O(t)$, we will need to identify a different way to pack these t Clique instances G_t into one Clique instance.

Method 2: Packing Construction

The general idea behind this more effective method of reducing a tuple of clique instances to a single clique instance is that we can view each graph G_i as being embedded in an independent set of size 3 tensor with some clique of size s.

$$G_i \subseteq I_3 \otimes K_s^{(i)}$$

We can think of these independent sets (each of size 3) as super-vertices, with our goal being to combine the cliques into a graph with fewer vertices. The edges of each clique $K_s^{(i)}$ we can think of as super-edges, where one edge within a clique represents all nine possible edges between two independent sets I_3. This allows us to embed t cliques of size s into a graph with significantly fewer than st vertices.

We introduce the following lemma which is key to our construction. We will prove this lemma in the next lecture.

Lemma 1 (Packing Lemma). *There exists a graph $P_{s,t}$ with $s \cdot \sqrt{t}^{1+o(1)}$ vertices such that:*

1. *The edges of $P_{s,t}$ partition into t cliques $K_s^{(i)}$ of size s.*
2. *$P_{s,t}$ does not contain any cliques of size s other than the ones than the ones we already have (i.e. the $K_s^{(i)}$).*

By meeting the first condition of the packing lemma, we ensure the completeness of our compression. This implies that if there exists a clique $K_s^{(i)}$ of size s (a "yes" instance) in our graphs G_i, our packing construction also contains a clique $K_s^{(i)}$ of size s.

The second condition ensures the soundess of our construction. Namely, if our construction has a clique of size s, then one of the given graphs also has a clique of size s. We can imply this because if we have a clique of size s, it has to come from one the established cliques in $K_s^{(i)}$. If we let these cliques share vertices outside of the cliques in $K_s^{(i)}$, then we would no longer be performing a mapping reduction from OR(Clique) to clique.

Our prior disjoint union construction satisfies both conditions of the packing lemma. However, as noted above, the problem with the disjoint union construction is that it uses too many vertices. To avoid this issue in our packing construction, we make our clique graphs share vertices. Note that our graphs will not share edges - the edges must remain disjoint. Then we will embed the

instance that we have into this graph. This ensures that if at least one of the graphs G_i has a clique of at least size s, then there will be a clique of size at least s in our packing construction. If we look at the edges of graph $P_{s,t}$, they partition into t cliques, and there are no cliques other than those in the graph.

Since we can provide $P_{s,t}$ as advice, we can use $P_{s,t}$ as a compressed OR(Clique) instance. This compressed instance $P_{s,t}$ has $n = s \cdot \sqrt{t}^{1+o(1)}$ vertices, resulting in a compression length $n^{2-\epsilon} = O(s^{2-\epsilon} \cdot t^{1-(\epsilon/2)+o(1)})$. Note that this running time is dominated by t if it is a large enough polynomial in s. That is, there exists some constant c such that if $t > s^c$ then $s^{2-\epsilon} \cdot t^{1-(\epsilon/2)+o(1)} = O(t)$. If we set $t = s^c$, then:

$$\begin{aligned}(s \cdot \sqrt{t}^{1+o(1)})^{2-\epsilon} &= s^{2-\epsilon} \cdot s^{c(1-(\epsilon/2)+o(1))} \\ &= s^{c(1-(\epsilon/2)+((2-\epsilon)/c)+o(1)} \\ &= O(s^c)\end{aligned}$$

for $1 - (\epsilon/2) + ((2-\epsilon)/c) < 1$.

Since our compression of OR(Clique) has size is linear in t, we also have a compression of OR(3SAT) whose size is linear in t. This means we can apply Theorem 3 to show that $\overline{3SAT} \in$ NP/poly, and thus coNP is in NP/poly. This completes our proof. ∎

4 Next Time

In the next lecture, we will complete the proof of our kernelization lower bound for vertex cover by proving the packing lemma. In order to do so, we will use the well-known fact that there exist high density subsets of $\mathbb{Z}$ with no arithmetic progression of length 3. This is complemented to Szemerédi's theorem, which states that for every positive integer k and there exists a real $\delta \in [0, 1)$, there exists a positive integer n such that every subset of $\{1, 2, ..., n\}$ of size greater then δn contains an arithmetic progression of length k.

References

[1] J. Flum and M. Grohe. *Parameterized Complexity Theory*, 2006.

[2] M. Sipser. *Introduction to the Theory of Computation* (2nd ed.) 2006.