CS 710: Complexity Theory 10/5/2017

Parallelism

Instructor: Dieter van Melkebeek Scribe: Joel Atkins and Zheming Wang

DRAFT

In the previous lecture we covered the notion of language compression, specifically the concept of Kernelization. We went on to establish kernelization lower bounds for the algorithmic problem Vertex Cover. In this lecture we will complete our coverage of Kernelization by proving the packing lemma and then begin the notion of Parallelism.

1 Kernelization

Our previous discussion of kernelization of Vertex Cover had the intent of establishing a compression algorithm for 3SAT. Specifically we showed that if VC has a kernelization with $k^{2-\epsilon}$ vertices then the order of 3SAT has such a compression. This means that $\overline{3SAT} \in$ NP/poly which would mean coNP $\subseteq$ NP/poly which would collapse the polynomial time hierarchy to the third level.

In the last lecture we utilized a packing lemma in order to compress the size of our instance of VC. We will now prove this lemma.

1.1 Packing Lemma Proof

Lemma 1 (Packing Lemma). *$\forall s, t \in \mathbb{Z}^+ \ \exists$ a graph $P_{s,t}$ on $O(st^{1/2+o(1)})$vertices such that:*

1. *The graph $P_{s,t}$ partitions into t cliques on s vertices each.*
2. *$P_{s,t}$ contains no other cliques of size s.*

Through the packing lemma we are trying to pack many cliques into a graph with relatively few vertices, this is intended to give a tight result up to the lower bound of $k^{2-\epsilon}$. Item 1 is satisfied because the t cliques use distinct edges and it is not necessary to use any more edges than those in the cliques so the graph can actually be partitioned into these t cliques. We will call these cliques $K_s^1, K_s^2, ..., K_s^t$ with the subscript s denoting the size of the cliques into which the edges partition. This construction satisfies item 2 because a clique of size s cannot be formed by combining the edges of 2 or more cliques, only by picking exactly one of the K_s^i.

To construct this graph we will construct an s-partite graph, namely there will be s partitions of the graph and none of the partitions will contain internal edges, edges will only go between distinct partitions. This is done by creating columns $1, 2, ..., s$. The rows of vertices will be numbered $1, 2, ..., p$. Each of the s columns will contain p vertices. Edges do not connect vertices within a column, only across distinct columns.

A possible construction of this graph is to pick $p = t$ and use each row to embed a particular clique, i.e. embed K_s^1 in row 1 and so forth. This corresponds to the disjoint union construction mentioned in the previous lecture. In this construction the number of vertices $= s \cdot t$ and is larger than stated in our Packing Lemma.

A better construction is necessary. When thinking about embedding a particular clique it means that in each of these columns we will select exactly one vertex. We can think of this as a function f which embeds the click by taking the column of the clique and map it to an element in $1, ..., p$.

$$f_i : \{1, 2, ..., s\} \mapsto \{1, 2, ..., p\} \text{ or, more succinctly, } f_i : [s] \mapsto [p]$$

For example, with our disjoint union construction $f_i(x) = i$. In other words, for each column we pick the same row for a given K_s^i. This will use p of the total p^2 edges, specifically the ones that travel straight within a row.

But there are p^2 possible edges and this allows for room to improve. The goal is to construct the graph with fewer vertices. This can be done by using more of the possible edges, ensuring it is done in a consistent and coherent way. An example of how to do this is as follows. We will think about p as a prime, this permits arithmetic within the range because $\mathbb{Z}/p\mathbb{Z}$ is a field. We will use the range $1, ..., p$ for consistency but this can also be thought of as $0, ..., p-1$. If the number of cliques in our problem is not a prime, p can be extended to a prime because for $x < t < 2x$ there exists a prime between and it will not affect the order of the function. Next, we will think about the construction of the graph as a linear function over the field $\mathbb{Z}/p\mathbb{Z}$, namely the function $f_i(x) = a_i x + b_i$. For every K_s^i we will associate a pair $(a_i, b_i) | a_i, b_i \in \{1, ..., p\}$. There are p^2 different possible pairs. If we select different pairs for every distinct clique then we realize item one of our lemma. This is because if we choose vertices x and y we have now chosen 2 points defining a line and there is exactly one particular choice for a_i, b_i which satisfies f_i for that line. This can also be thought of as solving a system of linear equations in 2 unknowns, this system is regular because we are looking at different x's.

There are p^2 distinct choices for (a_i, b_i), as we need to embed t cliques we want $p^2 = t$ so $p = O(t^{1/2})$. There are s columns so the number of vertices is $s \cdot t^{1/2}$. Our claim was that the number of vertices is $s \cdot t^{1/2+o(1)}$, this implies that our current construction is incorrect. This is because constraint 2 of our lemma has been violated. With our current construction we do not have only t cliques but rather every possible clique in the s-partite graph exists. In order to correct this we must analyze how additional cliques have been created. Namely, lets consider a clique, K_s, which does not fall entirely into one of K_s^i, $K_s \neq K_s^i$ $(\forall i \in [t])$.

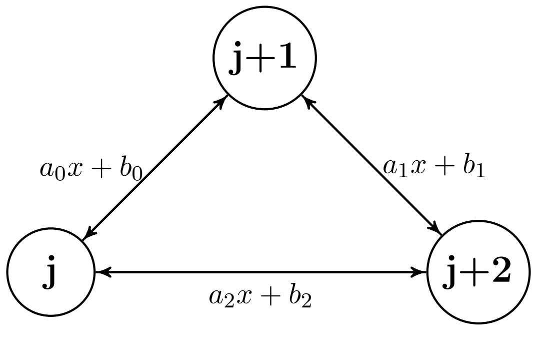

Figure 1: Nodes in K_s cols j, j_1, j_2

There must exist 3 consecutive columns $\{j, j+1, j+2\}$ such that in each column we pick a particular vertex belonging to K_s and each of the edges connecting those vertices belongs to one of the cliques which we put into the graph, specifically $a_0x + b_0, a_2x + b_1, a_2x + b_2$ denoted by the function. Moreover, these pairs $(a_0, b_0), (a_1, b_1), (a_2, b_2)$ are not all the same, if they were then there would be some index by which the vertices could be shifted and be in a K_s^i and if we repeated this then all edges of K_s would be in the corresponding clique which we put into the graph, but our assumption is that it isn't so that means there have to be 3 consecutive columns where this pattern happens which means these 3 edges do not all belong to the same clique.

Traversing this subgraph from point to point there are 2 ways to go from column j to column $j+2$. The direct way and the indirect way through $j+1$. How does the y value change? When moving from j to $j+1$ it changes by a_0. When going from $j+1$ to $j+2$ it changes by a_1. And when going from j directly to $j+2$ it changes by $2a_2$. From these we see $a_o + a_1 = 2a_2$. This is the same thing as saying (a_0, a_2, a_1) form an arithmetic sequence of length 3 and, moreover, it is a nontrivial arithmetic sequence because they're not all the same. They're not all the same because if $a_0 = a_1 = a_2$ we have stated we are no longer looking at our disjoint union construction of cliques entirely on one line. Our goal at this point is to remove all instances of this, in order to do so we will restrict a_i to come from a subset $A \subseteq \mathbb{Z}/p\mathbb{Z}$ such that A is AP3-free. An AP3-free set does not contain any nontrivial arithmetic progressions of length 3, trivial progressions can never be excluded. We will leave b_i unrestricted.

Is this construction too constrictive? Do there remain enough integers to prevent making p too large? A mathematical finding from 1946 [1] proves this is the case. Behrend's Construction of sets of integers shows that $\exists A\ s.t. |A| = p^{1-o(1)}$. This gives $p^{1-o(1)}$ possible values for a_i and p values for b_i and therefore $\#(a_i, b_i) = p^{2-o(1)} = t$ and $p = \sqrt{t}^{1+o(1)}$ which is an improvement over our previous construction and results in a number of vertices $= s \cdot p^{1/2+o(1)}$ $\square$

1.2 Behrend's Construction

An interesting construction gives us the required density in our AP3-free set. The full proof will not be given, but rather a general overview. The required density is arrived at by geometric construction in high dimensional space. We will look at all integer points (quotient points) in higher dimensions where all coefficients are integers within some small range, specifically $\mathbb{Z}_q^d$. The idea is that for each of these points their distance from the origin can only take on a relatively small number of values, $\{x | x = q^2 \cdot d\} \Rightarrow x \in (0, q^2 \cdot d)$. This means that each of these points will fall onto shells of certain radii. There will be at least one shell which contains a large number of these points and there are a relatively small number of distinct shells. The unique feature of these points which applies is that for any two points on a shell their midpoint will not fall on the shell. Thought of as vectors the points on a given shell form a set which does not contain an arithmetic progression of length 3. To then convert these vectors into the numbers required for our packing lemma one can consider the components of the vectors as elements in the expansion of a number which will define a linear mapping from these vectors to the integers up to a certain range and linear mappings preserve these relations. The end result is a large subset of integers in $\mathbb{Z}/p\mathbb{Z}$ which is AP3-free. It is necessary to pick p, d, q optimally to make a large number of points, but an optimal choice will give the required $|A|$.

1.3 Communication Game

In the last lecture we stated that if we have a language L such that OR(L) has this compression then $\overline{L} \in$NP/poly. It is possible to generalize this result to the following exercise. We are going to think of a communication game between 2 players, Alice and Bob, with and underlying language L. It is an asymmetric game in which Alice is given the input x and Bob does not know anything about the input. Alice is restricted to running in P and Bob is unrestricted. The goal is for them to communicate and for Alice to know if $x \in L$. One trivial way to do this if for Alice to just send x to Bob. Regardless of the complexity of the language L, Bob, because he is unrestricted, figures it out and sends one bit back to Alice and then Alice knows. This is trivial, but the game is modified so that the cost of the protocol is the number of bits that are sent from Alice to Bob. In the trivial case $x \in \{0,1\}^n$ the cost is $|x|$. For every language there exists a protocol of cost $|x|$.

For VC the trivial protocol involves sending the adjacency matrix. The size of the adjacency matrix is $O(n^2)$ for a matrix with n edges and therefore the protocol cost is n^2. With kernelization we can improve on this cost by performing a mapping reduction from the problem onto itself, thereby reducing the size of the problem. This kernelization gives a cost of at most $n^{2-\epsilon}$ because the problem has $k^{2-\epsilon}$ edges which can be written using $k^{2-\epsilon} logn$ bits. This is sent to Bob, Bob calculates the answer and sends it back to Alice. This is not possible unless the Polynomial Time Hierarchy collapses. In fact a protocol of cost $O(n^{2-\epsilon})$ is impossible unless coNP $\subseteq$ NP/ poly.

Exercise 1. Prove that Vertex Cover does not have protocol of cost $n^{2-\epsilon}$ ($\epsilon > 0,\ n = \#\text{vertices}$) unless coNP $\subseteq$ NP/ poly.

There are other ways in which protocols can be established. There can be more interaction between Alice and Bob. It is possible that Alice takes the lead by taking x and does some compression, asks a question of Bob and Bob responds. Our packing lemma proof generalizes to this. The result of this exercise is that languages L, such that NP$\leq_m^p$ L, have to contain $2^{n^{o(1)}}$ strings of length n unless coNP $\subseteq$ NP/ poly. This is a statement about the density of NP complete languages. If we look at all strings of length n, there are 2^n of them. This statement says that a fairly large number of them have to be in NP complete, namely $2^{n^{o(1)}}$. For a language that is NP complete that has density less than this use this to construct a protocol for VC of cost less than $n^{2-\epsilon}$ and that would imply the collapse. To construct this protocol Bob will play an active role and try to figure out enough information about the input x such that he can decide if $x \in L$. This is a relatively recent result. What was known for a longer time is that if you restrict the density to be polynomial, namely languages that only contain a polynomial number of strings of length n, or very sparse languages, they cannot be NP complete unless P=NP. And then there would actually be an equivalence as all but two of the languages are NP complete.

Exercise 2. Consider any language L of subexponential density, i.e., such that

$$(\forall \epsilon > 0)\,(\exists N \in \mathbb{N})\,(\forall n \geq N)\,|L \cap \{0,1\}^n| \leq 2^{n^\epsilon}.$$

Show that L cannot be NP-hard under $\leq_m^p$ unless coNP $\subseteq$ NP/ poly.

2 Parallelism

Parallelism means we can have multiple processors that work together to solve the same problem. In practice, multiple processors refers to some constant number of processors. But in complexity theory, if we only have a constant number of processors, the best speedup over a standard processor is a constant one. Hence, parallelism hereafter indicates multiple processors working together whose number is allowed to grow with input size. Although the divergence from reality may be criticized, it still gives us several meaningful results.

Consider the standard model of computation, Turing Machines, it means that we look at a growing number of Turing Machines instead of a fixed number, which induces two issues. The first one is about uniformity. Uniformity is hard to define due to the additional program needed to generate the increasing number of Turing Machines. The other one is that the communication between different Turing Machines is hard to model. Particularly, how Turing Machines interact with each other is specific to the underlying architecture implemented, which contradicts our anticipation.

For these reasons, we discuss parallelism using circuits rather than Turing Machines. In circuits, we consider gates as processors and layers as computation steps. Then we have a family of boolean circuits, for each length n there exits circuits C_n modeling parallelism. The communication can be specified as values conveyed between gates of different layers so the second issue is solved. As to the issue of uniformity, we still need to confirm the complexity of the algorithm that the given circuit is generated by input of length n. The standard uniformity is the logarithmic space one.

2.1 Concrete Model

With above factors, we usually focus on NC-Circuits to model parallel computation. The main classes we are looking at is denoted as follows:

Definition 1 ($\mathbf{NC}^{\boldsymbol{k}}$). *For $k \in N$, $NC^k = \{$all languages decidable by families of circuits of size poly(n) and $O(\log^k(n))$ depth$\}$, where circuits have gates of conjunction, disjunction, or negation.* Uniform *NC^k is decidable by a family of uniform circuit where circuit for input of size n can be computed in space $O(\log(n))$.*

2.2 Languages in Various NC^k

To get a closer look at various NC^k and get a feel for the NC hierarchy, we focus on several NC^k and see how efficiently some basic tasks can be accomplished in parallel.

The class NC^0 contains, by definition, only those languages decidable by circuits with constant depth and constant fan-in gates. This is equivalent to saying that these problems must be decidable by checking only a constant number of bits of the input. So the high asymptotic level performance of NC^0 is quite restrictive. Even the problem of parity is not in NC^0.

The class NC^1 instead, can solve some nontrivial problems. Circuits deciding this kind of language have logarithmic depth. We can easily prove that addition of binary number is in this class. Iterated addition, which means addition of several numbers, is also in NC^1. It can be proved by using an operation realized within constant-depth circuits which outputs two numbers with the same sum as three binary numbers input. By repeating this operation, we can reduce the number of remaining numbers to add by 1/3 each several constant layers. Within logarithmic number of layers, this problem can be reduced to binary addition which is in NC^1. Obviously, binary multiplication is also in NC^1. Iterated multiplication is also known to be NC^1 computable, though the proof is more complex.

The class NC^2 is decidable by circuits with $O(\log^2(n))$ depth and polynomial size. Polynomial size is not emphasized in NC^0 and NC^1 as they are automatically in it with corresponding depth limitation. NC^2 is not since $2^{O(\log^2(n))}$ is $O(n^{\log(n)})$, so we add a polynomial restriction on size. Many linear algebra problems can be done with NC^2. Matrix multiplication, solving systems of linear systems are examples of them.

The class NC equals $\cup_{k\geq 0} NC^k$. It refers to problem sets that are efficient to be computable on parallel computer. Then a big open question is whether $P \subseteq$ NC? The conjecture is no with completeness for P under logarithmic space reduction and the closely related circuit evaluating problem.

2.3 Complexity of NC

There is a close connection between parallel computing and space bounded computing.

Theorem 1. *Uniform $NC^1 \subseteq L \subseteq$ Uniform NC^2.*

Proof. First, we show that Uniform $NC^1 \subseteq L$. Suppose that a uniform family F of NC^1-circuits decides language A. Then, given an input x, we can simulate F in logarithmic space as follows:

1. Compute the circuit C appropriate for $|x|$. More precisely, compute each bit of the description of C as it is needed. We do not have enough space to store the entire description of the circuit, but we can compute each part as we need it in logarithmic space because F is uniform. We only have to keep track of the path from the root to where we are, which is logarithmic because F has logarithmic depth and bounded fan-in gates.

2. From the output node of C, compute the values of each gate in C recursively. Since the depth of C is in $O(\log |x|)$, we can do this computation in logarithmic space.

3. If the output of C is 1, accept. Else, reject.

The proof of $L \subseteq$ Uniform NC^2 is similar to that of $NSPACE(s) \subseteq DSPACE(s^2)$.

1. We can look at the computation tableau and recursively guess the intermediate configuration from a polynomially many number of possibilities. As accepting the language is equivalent to existence of accepting configurations, we could write a formula like that there exists an intermediate configuration, for each of the two blocks there exists next level of intermediate configurations, so on so forth. The corresponding circuit can be generated in logarithmic space.

2. As each level of the recursion reduces the number of configurations to 1/2, there exists $O(\log |x|)$ quantifiers in the formula. When this formula is converted into a uniform circuit, each quantifier generates another $O(\log |x|)$ depth circuit with binary input gates. Hence, the language can be decided by a uniform circuit with polynomial size and $O(\log^2 |x|)$ depth.

This theorem gives us a fairly tight connection between space bounded computation and log depth circuits.

3 Next Time

Next lecture we will continue our coverage of parallelism by discussing different ways of thinking about NC^1 and strengthening the connection with space bounded computation. Specifically we will characterize NC^1 in terms of branching programs of polynomial size where the branching programs have constant width. In this characterization we will show that the width of the branching program can be brought down to 5 and this depends upon the fact that the symmetric group on 5 elements, S_5, is not solvable.

References

[1] Felix A. Behrend. *On the sets of integers which contain no three in arithmetic progression*. Proc. Nat. Acad. Sci., 23:331–332, 1946.