CSC 2429 – Approaches to the P versus NP Question

Lecture #8: 12 March 2014

Lecturer: Joshua A. Grochow

Scribe Notes by: Joshua A. Grochow

Continuing from last time, our goal is to sketch a proof of:

Theorem 1 (Mulmuley [Mul99])**.** *In the parallel RAM model without bit operations, "*$\mathsf{P} \neq \mathsf{NC}$*."*

Last time we discussed the model and gave a proof that extracting the low-order bits cannot be done in "NC" in the PRAM model without bit operations. Toni asked a great question: if we have a strong lower bound for such a simple problem (extracting bits), aren't we done—why do we need to go on to prove *separately* lower bounds for problems that are seemingly much harder (like max flow)? The intuitive answer is that it's not clear how to reduce extracting bits to combinatorial optimization problems like max flow. If you consider the current best algorithms for these optimization problems, they all fall into the model of (sequential) RAM without bit operations—that is, nowhere do they need to extract bits. More concretely, by the techniques in the previous lecture and this one, it may be possible to prove unconditionally that there is no efficient parallel reduction from extracting bits to, say, max flow (in the PRAM model without bit operations), but I don't think this is known.

3.2 Dimension reduction via parametric complexity

The parametric complexity of a combinatorial optimization problem is a measure of how many times the optimum value can change when some integer parameters of the input are changed. A simple but useful formalization of this idea is to consider 1-parameter families of inputs. For example, for maximum flow, we might consider a 1-parameter family of inputs on a fixed graph, whose weights are linear functions of the one parameter λ. Then the optimum value for this particular one-parameter family is a function of λ alone; if the optimum value is piecewise-linear as a function of λ, then the complexity of this particular parametrization is the number of breakpoints. The parametric complexity of a problem such as maximum flow is then the maximum complexity of any parametrization.

To make this precise, recall that we consider combinatorial optimization problems that have a boolean part (e. g., for maximum flow, this would be the underlying graph) and a numeric part (e. g., the weights).

Definition 1 (see Definitions 3.1 and 3.2 of Mulmuley [Mul99])**.** A *linear parametrization for cardinality* n for a given combinatorial maximization problem is a map I from some interval $[\alpha_{min}, \alpha_{max}] \subset \mathbb{R}$ to instances with n numeric parameters such that 0) the boolean part of

$I(\lambda)$ is independent of λ; 1) each numeric parameter of $I(\lambda)$ is a $\mathbb{Q}$-linear function of λ; and 2) the optimum value of the instance $I(\lambda)$, denoted $OPT(I(\lambda))$ is a piecewise-linear and convex function of λ.

The *complexity of a linear parametrization* I as above is the number of bounded linear segments of the graph Γ of $OPT(I(\lambda))$. The *bit-size of a linear parametrization* is the maximum of the bit-sizes of the vertices of Γ and the bit-sizes of the coefficients in the linear functions of λ appearing in the numeric parameters of $I(\lambda)$.

The *parametric complexity* $\rho(n)$ of the optimization problem is a function of n, which is the maximum of the complexity of all linear parametrizations for cardinality n. Similarly, the parametric complexity $\rho(n, \beta(n))$ for cardinality n and bit-size $\beta(n)$ is the maximum complexity of all linear parametrizations of cardinality n and bit-size at most $\beta(n)$.

For a graph optimization problem such as maximum flow, the use of parametric complexity allows us to reduce the space of inputs we consider drastically, while still maintaining much of the complexity of the original problem. Naively, we have to consider all $\sim 2^{n^2}/n!$ n-vertex graphs, and for each graph with e edges, the e-dimensional space $\mathbb{Q}^e$ of possible weights. Instead, we may consider a single fixed graph on n vertices and a 1-dimensional space of inputs. This dimension reduction is quite crucial—without it, it seems quite difficult to bound the number of indistinguishability classes, as polynomials of even moderate degree in n variables can cut up $\mathbb{R}^n$ into too many pieces.

To state the lower bound for general optimization problems, we need one additional assumption, which is a property shared by many natural optimization problems: namely, that if we scale all the numeric parameters by α, then the optimum value should also be scaled by α. Such optimization problems are called *homogeneous*.

The next result is the key theorem that connects parametric complexity to computational complexity; we will spend the rest of this lecture discussing its proof.

Theorem 2 (Theorem 3.3 of Mulmuley [Mul99])**.** *The decision version of any homogeneous optimization problem cannot be solved in the PRAM model without bit operations in time* $t(n, N) = \sqrt{\log(\rho(n, \beta(n)))}/c$ *using* $2^{t(n)}$ *processors for any sufficiently large constant* c*, where* $\rho(n, \beta(n))$ *is the parametric complexity of the problem as in Definition 1. This holds even if the bit-length of the input is restricted to be at most* $c'\beta(n)$ *for some sufficiently large constant* c'*.*

Theorem 1 then follows from Theorem 2 together with previously known lower bounds on the parametric complexity of various P-complete optimization problems: the parametric complexity of combinatorial linear programming [Mur80], minimum-cost flow [Zad73], and maximum flow [Car83a, Car83b, Mul99] are all $2^{\Omega(n)}$. These lower bounds hold even for parametrizations of bit-length $O(n)$ for combinatorial linear programming and minimum-cost flow, and $O(n^2)$ for maximum flow.

In contrast, Mulmuley shows [Mul99, Theorem 3.10] that the parametric complexity of global minimum cuts in weighted undirected graphs is at most polynomial (that is, minimum weight cut over all cuts in the graph, not just over s-t cuts for some fixed s and t). This is in accord with the fact that this problem has fast parallel algorithms in the PRAM model without bit operations [Kar93, KM97]. Although a general theorem stating that low parametric complexity implies low computational complexity is not known, this last result and Theorem 2 together provide fairly strong evidence that parametric complexity closely tracks parallel computational complexity.

3.3 Lower bound in the linear model

We begin with the *arithmetic* linear PRAM model over $\mathbb{Q}$, which is a truly algebraic (and standard) model, in which the PRAM program is only allowed to depend on the number of rational inputs, but not their bit-length. This will exhibit in a very simple setting how a lower bound on parametric complexity yields a lower bound on computational complexity. Extending this to the PRAM model over $\mathbb{Z}$ without bit operations (program can depend on the bit-length) will then give us occasion to introduce the next two key ideas in the proof of Theorem 1.

Proposition 3 (implicit in Section 4 of Mulmuley [Mul99])**.** *If a homogeneous combinatorial optimization problem of parametric complexity $\rho(n)$ can be solved on an arithmetic linear PRAM in time $t(n)$ with $p(n)$ processors, then $t(n)(2p(n))^{t(n)+1} > \rho(n)$.*

In particular, minimum-cost flow can't be solved on an arithmetic linear PRAM in $\sqrt{n}/c$ time on $2^{\sqrt{n}/c}$ processors for any large enough constant c.

Note that for the arithmetic linear PRAM we get a fairly strong time-processor trade-off, which we'll also get in the linear PRAM without bit operations, but not in general. Just for safety, we emphasize again that this proposition is purely algebraic, unlike Theorem 1—no bits here!

Proof idea. Let $I(\lambda)$ be a linear parametrization for cardinality n of maximum complexity ρ ($=2^{\Omega(n)}$ for minimum-cost flow [Zad73]). Instances of the decision version of any combinatorial maximization problem consists of an instance of the combinatorial optimization problem together with a threshold τ, and the problem is to decide whether the optimum value is at least τ. We denote the threshold of $I(\lambda)$ by $\tau(\lambda)$. We change our arithmetic linear PRAM into an arithmetic linear PRAM with a single input λ that first computes $I(\lambda)$ and then runs the original PRAM on $I(\lambda)$. Let $[\alpha_{min}, \alpha_{max}] \subset \mathbb{R}$ denote the domain of definition of the parametrization, and let $\alpha_{min} < \alpha_1 < \ldots < \alpha_{\rho(n)} < \alpha_{max}$ denote the λ-coordinates of the breakpoints of the graph of $OPT(I(\lambda))$ (recall that it's piecewise-linear, by Definition 1). Intuitively, all a linear PRAM can do is determine which interval $[\alpha_i, \alpha_{i+1}]$ the input λ is in and then check whether the threshold $\tau(\lambda)$ is less than the optimum value $OPT(I(\lambda))$. Note that within any given interval $[\alpha_i, \alpha_{i+1}]$, $OPT(I(\lambda))$ is linear, so a linear PRAM can indeed check this condition. Furthermore, as the graph of the optimum value is convex (by Definition 1), the slopes of the lines of the optimum value over the interval $[\alpha_i, \alpha_{i+1}]$ are all distinct, so we can't exclude from consideration any α_i.

To make this idea precise, we return to the same type of reasoning as in the lower bound on extracting bits from the previous lecture. The first branch instructions are all fixed linear functions of λ, and there are at most $p(n)$ of these: one per processor. These branch instructions partition the input into at most $2p(n)+1$ classes: $p(n)+1$ intervals between the roots of these linear functions plus the $p(n)$ roots themselves. We define two inputs to be t-indistinguishable if they are indistinguishable by the branches executed at or before time t. For any fixed t-indistinguishability class, the branches at time $t+1$ are fixed linear functions of λ, and there are most $p(n)$ of these. These can subdivide a given t-indistinguishability class into $2p(n)+1$ $(t+1)$-indistinguishability classes. By induction, we find that the total number of indistinguishability classes at the end of the program is at most $\approx (2p(n))^{t(n)}$. Each indistinguishability class is a union of connected intervals that is determined by at most $p(n)t(n)$ linear inequalities, as this is the total number of inequalities that may be checked during a single execution. The total number of inequalities checked over all indistinguishability classes—$t(n)(2p(n))^{t(n)+1}$—must be at least the number of breakpoints in the graph of $OPT(I(\lambda))$. □

Remark 1. Because only linear functions can be considered by linear PRAMs, in the counting in the second paragraph of the above proof outline we don't get the extra factor of $2^{t(n)^2}$ that appears in the proofs of the lower bound on extracting bits and Theorem 1. More specifically, when multiplications are allowed, the value $(2p(n))^{t(n)}$ in the above proof is replaced by $(2p(n)2^{t(n)})^{t(n)}$ because in general the branch polynomials may have degree up to $2^{t(n)}$. This is why we get the time-processor trade-off for linear PRAMs but not for PRAMs allowing multiplication.

Now we extend this result from the arithmetic linear PRAM model to the linear PRAM model without bit operations. Since we're now keeping track of bits, we must keep in mind that rational numbers are represented by pairs of integers.

In the preceding proof, we were essentially considering a two-dimensional linear configuration over $\mathbb{Q}$, namely the graph $\Gamma \subset \mathbb{R}^2$ of $OPT(I(\lambda))$. In that simple setting, we very quickly reduced the issue to just the values of λ at which Γ changes slope, a configuration of points in $\mathbb{R}$ (see the last sentence of the first paragraph of the proof outline above). In our next setting, however, we'll consider the graph of the optimum value as a function of two integer parameters—the numerator and denominator of λ—so we'll be dealing with a configuration of planes in $\mathbb{R}^3$, which we cannot so easily reduce to a configuration of lines in $\mathbb{R}^2$, let alone a configuration of points in $\mathbb{R}$. This will naturally lead us to use a simple form of cylindrical decomposition.

Theorem 4 (Theorems 4.1 and 4.2 of Mulmuley [Mul99])**.** *If a homogeneous combinatorial optimization problem of parametric complexity $\rho(n, \beta(n))$ can be solved for integer parameters of bit-length at most $b\beta(n)$ on a linear PRAM without bit operations in time $t(n, b\beta(n))$ with $p(n, b\beta(n))$ processors, then $(t(n, b\beta(n))(2p(n, b\beta(n)))^{t(n,b\beta(n))+1})^3 > \rho(n, \beta(n))$, for any large enough constant b.*

In particular, such a problem cannot be solved on a linear PRAM without bit operations in time $t(n, b\beta(n)) = \sqrt{\log(\rho(n, \beta(n)))}/c$ on $2^{t(n,b\beta(n))}$ processors, even if each numeric parameter in the input is restricted have at most $b\beta(n)$ bits, for any large enough constants b, c. It follows that minimum-cost flow on n-node graphs with every cost and capacity of bit-length at most bn can't be solved on a linear PRAM without bit operations in $\sqrt{n}/c$ time on $2^{\sqrt{n}/c}$ processors, for any large enough constants b, c.

We will outline the proof of the above theorem with the exponent 3 replaced by 12, as we use a very naive argument at one point; Mulmuley shows that 3 is in fact achievable.

Proof idea. We begin as in the proof of Proposition 3, and maintain the notation from that proof. In particular, $I(\lambda)$ is a linear parametrization of maximum complexity $\rho(n, \beta(n))$, defined on the interval $[\alpha_{min}, \alpha_{max}] \subset \mathbb{R}$, and $\alpha_{min} < \alpha_1 < \cdots < \alpha_{\rho(n,\beta(n))} < \alpha_{max}$ are the λ-coordinates of the breakpoints of the graph $\Gamma \subset \mathbb{R}^2$ of the optimum value $OPT(I(\lambda))$.

The basic idea is this: in the arithmetic model, we considered all rational inputs, which are dense in the plane $\mathbb{R}^2$. In order to decide whether a point was in the set $\{(\lambda, t) : OPT(I(\lambda)) \geq t\}$, the density of the rational points in $\mathbb{R}^2$ essentially implied that an arithmetic linear PRAM *had* to be able to compute the entire graph Γ of $OPT(I(\lambda))$, which had ρ breakpoints; thus over all possible inputs the PRAM had to make at least ρ distinct branches. In the model without bit operations, we do not get to consider all rational inputs, but only rational inputs whose numerator and denominator both have bit-size bounded by $b\beta(n)$. This set of inputs is no longer dense in the plane, so it is *a priori* conceivable that a linear PRAM without bit operations could decide whether such a bit-bounded point was a yes-instance without having to compute enough branches to determine the entire graph Γ, especially since the PRAM can depend on the bit-size of the input. The basic idea is that if we allow the constant b to be large enough, then because

of the integer nature of the inputs, the inputs of bit-size at most $b\beta(n)$ are "dense enough" to make the preceding argument work. In order to make this idea precise, we separate the input parameter λ into its integer numererator and denominator and then consider a set of integer points in $\mathbb{R}^3$. Actually, we have to do this anyways because the model of computation forces it on us, but it also turns out to be useful in making this idea work. □

Proof outline. As mentioned above, the input to the modified "one-input" PRAM is no longer the single rational value λ, but consists of the integer numerator $\mathfrak{n}$ and denominator $\mathfrak{d}$ of λ, that is, $\lambda = \mathfrak{n}/\mathfrak{d}$. We also need to change our parametrization from being $\mathbb{Q}$-linear to being $\mathbb{Z}$-linear, which we do as follows. As $I(\lambda)$ is a linear parametrization, each numeric parameter is a function of the form $u\lambda + v$, where $u, v \in \mathbb{Q}$. To make the u's and v's integers, we can multiply all of them by the least common multiple of their denominators; since the optimization problem is homogeneous, this multiplies the optimum value by the same factor, but otherwise leaves the structure of the problem—including the corresponding geometry—unchanged. Next, to replace the input parameter λ by the pair $(\mathfrak{n}, \mathfrak{d})$, we multiply every numeric parameter by $\mathfrak{d}$; again, since the problem is homogeneous this also multiplies the optimum value by $\mathfrak{d}$. The resulting numeric parameters now look like $u\mathfrak{n} + v\mathfrak{d}$ with $u, v \in \mathbb{Z}$; we denote the resulting parametrization by $\widetilde{I}(\mathfrak{n}, \mathfrak{d})$, which is just a scaled version of $I(\mathfrak{n}/\mathfrak{d})$. We will now also need to consider not just the graph $\Gamma \subseteq \mathbb{R}^2$ of $OPT(I(\lambda))$, but also the graph $\widetilde{\Gamma} \subseteq \mathbb{R}^3$ of $OPT(\widetilde{I}(\mathfrak{n}, \mathfrak{d}))$; for the definition of this graph, $\mathfrak{n}$ and $\mathfrak{d}$ may take any real values.

We identify the $\mathbb{R}^2$ in which Γ sits with the $\mathfrak{d} = 1$ plane in $\mathbb{R}^3$ via $(\lambda, OPT) \leftrightarrow (\lambda, 1, OPT)$. This follows the general idea that $\lambda = \mathfrak{n}/\mathfrak{d}$: in fact, for any d, we can identify any $\mathfrak{d} = d$ plane with the $\mathfrak{d} = 1$ plane, and hence the (λ, OPT)-plane, via $(\mathfrak{n}, \mathfrak{d}, OPT) \leftrightarrow (\mathfrak{n}/\mathfrak{d}, 1, OPT/\mathfrak{d}) \leftrightarrow (\mathfrak{n}/\mathfrak{d}, OPT/\mathfrak{d})$. Under these identifications, the intersection of $\widetilde{\Gamma}$ with the $\mathfrak{d} = 1$ plane is exactly Γ, and the intersection of $\widetilde{\Gamma}$ with the $\mathfrak{d} = d$ plane is the d-scaled version of Γ (the image of Γ under the map $(\lambda, OPT) \mapsto (d\lambda, d, dOPT)$).

To simplify the ensuing discussion, we introduce the relevant geometric term: for any point $x \in \mathbb{R}^3$, we define the *fan* over or through x as the line in $\mathbb{R}^3$ passing through the origin and x. The fan of a set S of points is then the union of the fans of the points in S. The above paragraph can then be rephrased succinctly as $\widetilde{\Gamma} = \text{fan}(\Gamma)$ (at least this is true away from the $\mathfrak{d} = 0$ plane, but we don't ever need to consider that plane; $\widetilde{\Gamma}$ is well-defined even when $\mathfrak{d} = 0$, because the numeric inputs are $u\mathfrak{n} + v\mathfrak{d}$).

Because we only care about integer values of $\mathfrak{n}, \mathfrak{d}$ of bit-length at most $b\beta(n)$, we need only consider the situation with a certain box in $\mathbb{R}^3$. We're happy to exclude the possibility of $\mathfrak{d} = 0$, and we may assume without loss of generality that $\mathfrak{d}$ is always positive. We thus consider the *bounding box* defined by $1 \leq \mathfrak{d} \leq 2^{b\beta(n)}$ and $-2^{b\beta(n)} \leq \mathfrak{n} \leq 2^{b\beta(n)}$.

At this point—before we've actually proved anything—it's already worth remarking on how this geometric picture will help implement the proof idea discussed above. Integer points in $\mathbb{R}^3$ with higher values of $\mathfrak{d}$ correspond to rational points in $\mathbb{R}^2$ with larger denominators. The idea is that if we take the constant multiplier b of the bit-size bound to be large enough, then there are enough integer points within the bounding box that are sufficiently close to $\text{fan}(\Gamma) = \widetilde{\Gamma}$ to force the linear PRAM without bit operations to have to compute at least a significant fraction of the bounded linear segments of Γ. Here, "sufficiently close" means that, if the point is in the $\mathfrak{d} = d$ plane, then within the $\mathfrak{d} = d$ plane the distance from the point to $\widetilde{\Gamma} \cap \{\mathfrak{d} = d\}$ in the OPT-direction is at most 1. If we translate this back to the $\mathfrak{d} = 1$ plane, this means the point is closer than $1/\mathfrak{d}$ to Γ, capturing the idea that the points are "sufficiently dense" suggested above.

To implement this idea, we start from the other direction and assume that there *is* a linear

PRAM that solves the problem for $b\beta(n)$-bit-bounded inputs in time $t(n)$ on $p(n)$ processors. (In fact, $t(n)$ and $p(n)$ could also depend on $b\beta(n)$, but the lower bound is strong enough not to need this.) We define t-indistinguishability classes as in the proof of Proposition 3. Within each t-indistinguishability class, the branch instructions at time $(t+1)$ introduce additional planes into $\mathbb{R}^3$. The counting is the same as in the previous proposition: there are at most $t(n)(2p(n))^{t(n)+1}$ planes defined by all branch instructions over all inputs.

Let δ denote the number of planes. The bounding box is sliced up by these δ planes into at most $O(\delta^3)$ cells. A standard result from computational geometry (see, e. g., [Mul93]) lets us refine this decomposition further to get cells with a particular shape. In particular, we want each cell in the resulting to decomposition to be a "cylinder:" it should have six sides, a well-defined floor—a unique facet that is hit first by any ray from the origin—and similarly a well-defined ceiling. The procedure for this is as follows: consider the projection of every intersection of planes to the $\mathfrak{d} = 1$ plane. This gives an arrangement lines in the plane. Add a line parallel to the OPT-axis through every intersection point of this line arrangement. Then add to the original δ planes the fans through the resulting line arrangement in the plane. Even a very naive argument shows that the resulting arrangement has at most $O(\delta^4)$ planes: there are at most $O(\delta^2)$ pairs of planes to intersect, and then after projection these intersections to the plane there are at most $O(\delta^4)$ pairs of lines to intersect. The resulting arrangement of planes carves the bounding box into at most $O(\delta^{12})$ cells. In fact, one can do much better than this, but any constant exponent would have sufficed. This implements the standard technique from computational geometry, so the resulting cells are all cylinders, as desired. Let Δ denote the number of cells in the resulting arrangement.

Finally, we use a counting argument to show that the average number of bounded linear segments of Γ that are "close" to a given cell of the resulting arrangement is sufficiently large. Comparing the total number of sides of cells (6Δ) to the number of linear segments these sides must be close to (a nontrivial fraction of $\rho(n, \beta(n))$), gives the desired result.

Suppose a cell contains an integer point of $\widetilde{\Gamma}$. Note that points on $\widetilde{\Gamma}$ are yes-instances of the optimization problem; also, if $(\mathfrak{n}, \mathfrak{d}, opt)$ is a point of $\widetilde{\Gamma}$ then $(\mathfrak{n}, \mathfrak{d}, opt - 1)$ is a no-instance. As our original PRAM program correctly solves the optimization problem, the cells of the original partition cannot contain both a point $(\mathfrak{n}, \mathfrak{d}, opt) \in \widetilde{\Gamma}$ and $(\mathfrak{n}, \mathfrak{d}, opt - 1)$. As the new partition refines the original partition, the same is true of the cylindrical cells in our new partition. This suggests the following definition: given a point $p \in \Gamma \subset \{\mathfrak{d} = 1\}$, say that a cell is *good for* p if it contains some integer point on the ray through p (which is necessarily a point of $\widetilde{\Gamma}$). If ℓ is a bounded linear segment of Γ, say that a cell C is *good for* ℓ if C is good for a $1/\Delta$ fraction of δ^{13} points on ℓ whose $\mathfrak{n}$-coordinates are equally spaced (recall that Δ is the total number of cells). Note that if C is good for ℓ, then some bounding hyperplane of C must be close to $\text{fan}(\ell)$, as otherwise C would contain both yes- and no-instances. By choosing the constant b in the bit-bound $b\beta(n)$ sufficienty large, a simple counting argument, using the fact that the vertices of Γ have bounded bit-length, shows that there is a cell of the arrangement which is not only good for a single linear segment ℓ, but good for a $1/\Delta$ fraction of such linear segments of Γ. This is only possible if the number of bounding hyperplanes of C—namely six—is greater than the number of such linear segments. In other words, we need that $6\Delta \geq \rho(n, \beta(n))$. As Δ is at most $(t(n)(2p(n))^{t(n)+1})^{12}$, this completes the proof. □

3.4 Lower bound in the general model: from linear geometry to algebraic geometry

Finally, we give the idea of how to extend the proof from linear PRAMs without bit operations to general PRAMs without bit operations. The basic idea and outline is the same as in Theorem 4. The catch is that the branching functions will no longer be linear, but may be higher-degree polynomials.

Extending the proof from the linear to the general PRAM model without bit operations is where all of the deep results from real algebraic geometry get used. In particular:

1. The $O(m^3)$ upper bound on the number of faces in an arrangement of planes in $\mathbb{R}^3$ gets replaced by the Milnor–Thom bound on the number of connected components of a (hyper)*surface* arrangement [Mil64, Tho65], where the surfaces may be defined by polynomials of a given degree;

2. The easy cylindrical decomposition for hyperplane arrangements gets replaced by the Collins cylindrical decomposition [Col75] for real semi-algebraic varieties, with similar properties of the resulting decomposition. The process of constructing this decomposition—projecting onto the plane, adding additional lines, and then taking fans—is similar in outline but a bit more complicated in detail than in the linear case, so we discuss it further below;

3. In order to make all of this work—especially the way in which the cylindrical decomposition is used—we have to be careful about any singularities the hypersurfaces may have and how such hypersurfaces may intersect. In the linear case, hyperplanes are always smooth, and the intersection of any two distinct but intersecting hyperplanes is always a transverse intersection.[1] To ensure this, classical results of differential geometry (see, e. g., [GG73, Chapter 2]) imply that it is enough for the hypersurfaces to be in "general position," which can be achieved by an arbitrarily small perturbation of their coefficients. For technical reasons, we'll also want to ensure that no point of interest actually lands squarely on a hypersurface; this can be achieved by adding $\pm\varepsilon$ to the constant term of every polynomial, and at most doubling the number of branch instructions, as in the proof of the lower bound on extracting bits from last lecture.

Note how the use of multiplication in the computation naturally leads to the use of algebraic geometry in the lower bound.

To get a cylindrical decomposition in the general case, we first project onto the $\mathfrak{d} = 1$ plane. In the linear case, we could project the intersection of any two planes. In the general case, we not only project all the pairwise intersections of surfaces—which will be 1-dimensional curves of potentially high degreee—but also the *silhouette* of every surface: essentially the outline of the intersection of the fan of the surface with the $\mathfrak{d} = 1$ plane. We then add new OPT-axis-parallel lines in the $\mathfrak{d} = 1$ plane through every intersection point of the resulting set of curves and through any point at which the curve becomes tangent to a OPT-axis-parallel line. As

[1] An intersection of two surfaces is transverse if it locally looks like an intersection of two planes. More formally, the intersectionf two surfaces S, S' is transverse at the point $x \in S \cap S'$ if the only vectors based at x that are tangent to S and S' are those which are tangent to $S \cap S'$. An example of a non-transverse intersection of curves in $\mathbb{R}^2$ is the intersection of the sets $S = \{(x, y) : y = x^3\}$ and $S' = \{(x, y) : y = -x^3\}$. The intersection is the origin, which has no tangent vectors, as it is zero-dimensional. However, because of the inflection point at the origin in the graphs of x^3 and $-x^3$, the vector $(1, 0)$ is tangent to both S and S' at the origin.

before, we take the fan of the resulting arrangement of curves in the $\delta = 1$ plane. We then take the additional step of adding horizontal planes parallel to the $\delta = 1$ plane at sufficiently many equally spaced intervals in the bounding box.

The proof then proceeds essentially as before, with one more ingredient. Rather than merely bounding the *number* of bounding hyperplanes of each cell, the boundary components of a cell may be surfaces defined by higher-degree equations. Counting the number of sides is replaced by a more complicated argument using the degree of the equations defining the bounding surfaces. This is then used to get a bound on the number of inflection points of these bounding surfaces, which can be seen as a rough measure of how many planes would be needed to approximate the surface. Then essentially the same idea as before is used: a counting argument shows that some cell must be good for many segments of Γ, which is impossible because there aren't enough inflection points in its bounding hypersurfaces.

4 Towards $\mathsf{P} \neq \mathsf{NC}$ and Geometric Complexity Theory

To put this result in a larger context and relate it to P versus NC and the Permanent versus Determinant Conjecture, we rephrase it in terms of a variant of determinantal complexity. The *determinantal complexity* of a polynomial $f(\vec{x})$ is the smallest m such that f is an affine projection of $\det_m$.

Recall that in the boolean world, NC is the same as the set of those languages whose characteristic functions have quasi-polynomial-size (i. e., $2^{\mathrm{poly}(\log n)}$) boolean formulas. In the algebraic world, a function has a quasi-polynomial-size algebraic formula if and only if it has quasi-polynomial determinantal complexity (equivalently, the determinant is complete under qp-projections for $\mathsf{VQP} = \mathsf{VQP}_e$). The definition and conjecture below can be seen as bridging the formal gap between these analogous facts.

The second half of the following definition is implicit in Mulmuley [Mul99, Definition 7.2]. Let $B(\beta)$ denote the set of sequences of arbitrarily many integers in which each integer has bit-length at most β; thus for any n, $\mathbb{Z}^n \cap B(\beta)$ consists of all n-tuples of integers each with bit-length at most β.

Definition 2 (compare Definition 7.2 of Mulmuley [Mul99])**.** Let L be a subset $\mathbb{Z}^n$. The *bit-bounded determinantal complexity* of L over $\mathbb{C}$, denoted $\mathrm{bdc}_{\mathbb{C},\beta}(L)$ or just $\mathrm{bdc}_\beta(L)$, is the least m such that the characteristic function of L, 1_L, agrees with a projection of of $\det_m$ on inputs each of whose parameters has bit-length at most β. More verbosely, $\mathrm{bdc}_\beta(L)$ is the least m such that there is an $m \times m$ matrix $M(x_1, \ldots, x_n)$ whose entries are affine $\mathbb{C}$-linear combinations of the x_i such that $1_L(\vec{x}) = \det_m(M(\vec{x}))$ for all $\vec{x} \in \mathbb{Z}^n \cap B(\beta)$.

The *positive bit-bounded determinantal complexity* of L over $\mathbb{C}$, denoted $\mathrm{bdc}^+_{\mathbb{C},\beta}(L)$ or $\mathrm{bdc}^+_\beta(L)$, is the least m such that there is an $m \times m$ matrix M as above such that $\vec{x} \in L$ if and only if $\det_m(M(\vec{x})) = 1$ for all $x \in \mathbb{Z}^n \cap B(\beta)$.

Mulmuley introduced a conjecture on the positive bit-bounded determinantal complexity of the decision version of minimum-cost flow, restricted to inputs with parameters of bit-size at most $\beta = bn$. If the conjecture holds for all values of b, then $\mathsf{P} \neq \mathsf{NC}$; using the same techniques as in Theorem 1 he shows that the conjecture holds for all sufficiently large b. This naturally leads to an idea of the limitations of the techniques of this lower bound, and how they might be overcome. In the remainder of this section we discuss this conjecture and related results, and how it could lead one naturally to the current GCT Program.

Let L_n denote the language consisting of the yes-instances of the decision version of the minimum-cost flow problem on n-vertex graphs. L_n is an infinite subset of $\mathbb{Z}^{2\binom{n}{2}+2}$, as this is how many integer parameters are needed, but the exact number of such parameters isn't crucial.

Conjecture 5 (Section 7 of Mulmuley [Mul99])**.** *The positive bit-bounded determinantal complexity of minimum-cost flow with weights of bit-size $\leq bn$ is exponential. More precisely, with notations as above, for all constant $b > 0$, for a sufficiently large constant c, and for all sufficiently large n,*

$$bdc^{+}_{\mathbb{C},bn}(L_n) > 2^{n/c}.$$

A few remarks are in order:

- Of course this conjecture could also be made for any sufficiently algebraic problem that is not expected to be in NC; for problems in P, any problem in strongly polynomial time is likely to be "sufficiently algebraic," but the conjecture could also be made for problems outside of P.
- It is natural to wonder why one should allow $\mathbb{C}$-linear combinations in Conjecture 5, when we are only concerned with integer inputs. The analogous definition with $\mathbb{Z}$-linear combinations would indeed be quite reasonable. The complex numbers are used primarily because algebraic geometry is much nicer over $\mathbb{C}$; it is plausible that this doesn't strengthen the conjecture much (if at all), as it seems unlikely that the conjecture would hold for $\mathbb{Z}$-linear combinations but not $\mathbb{C}$-linear combinations.
- We could have written the conjecture above as $(\forall b > 0)\mathrm{bdc}^{+}_{\mathbb{C},bn}(L_n) > 2^{\Omega(n)}$, but, as we'll see in the next few results, the relationship between the bit-size bn and the constant $1/c$ in the exponent of the conjectured lower bound turns out to be crucial. Furthermore, although Mulmuley only made the conjecture for *positive* bit-bounded determinantal complexity, we'll see as we go along that this may not be crucial—the analogous conjecture for usual bit-bounded determinantal complexity seems to have the same implications, both formal and informal.

Before we get to those results, we'll try to give some intuition for this conjecture.

There are at least two intuitive motivations for Conjecture 5. We first give our own after-the-fact motivation for the conjecture, and then discuss the intuitive *a priori* explanation given by Mulmuley.

Since NC corresponds to quasi-polynomial boolean formula size, and quasi-polynomial algebraic formula size corresponds to quasi-polynomial determinantal complexity, a natural conjecture which one would guess is nearly equivalent to $\mathsf{P} \neq \mathsf{NC}$ would be that the determinantal complexity of some function in P is more than quasi-polynomial. To make this precise, since P is a boolean class, the conjecture uses bit-bounded determinantal complexity instead. The bound $2^{\Omega(n)}$ can be seen simply as a natural choice for a super-quasi-polynomial function; indeed, one could also make the conjecture that $\mathrm{bdc}^{+}_{\mathbb{C},bn^{\varepsilon}} > 2^{n^{\varepsilon}/c}$ for some $\varepsilon > 0$, and the remaining results in this section would still be true (in particular, this conjecture would still imply $\mathsf{P} \neq \mathsf{NC}$).

Mulmuley [Mul99, pp. 1504–1505] motivates Conjecture 5 by noting that it's weaker than run-time of the currently best-known parallel method of solving minimum-cost flow under the assumption that maximum matching lies in NC^k for some $k > 1$ (not necessarily an integer). The method involves "exploding" the bits from the $\mathrm{poly}(n)$ input parameters of bit-size β describing a weighted graph to get $\mathrm{poly}(n)2^{\beta}$ boolean parameters describing an unweighted graph, in such a

way that minimum-cost flow is reduced to maximum-cardinality matching. One may then apply any parallel algorithm for maximum matching, whence it is called the "explode-and-match" method. If maximum matching is in NC^1, then the conjecture is false for sufficiently small b, but it is widely-believed that maximum matching is not in NC^1, which is roughly (but not formally) equivalent to the belief that the determinant is not in NC^1 (does not have polynomial-size formulas). Even if maximum matching is in $\mathsf{NC}^{1+\varepsilon}$ for any $\varepsilon > 0$, the run-time of the explode-and-match method is $2^{O(n^{1+\varepsilon})}$. The conjecture is then motivated by the possibility that one cannot do much better than this method even for inputs of small bit-length (which would in fact suggest a slightly stronger conjecture with a lower bound of $2^{\Omega(n^{1+\varepsilon})}$).

Now we move onto the consequences of Conjecture 5.

Proposition 6 (Proposition 7.3 of Mulmuley [Mul99])**.** *If Conjecture 5 holds for some positive $b < 1/(2c)$, then $\mathsf{P} \neq \mathsf{NC}$.*

We give the proof here because it is short and illuminating, and also so we may see how it goes through even for bdc instead of bdc^+, and furthermore, even if we conjectured only $\text{bdc}_{\mathbb{Z},bn^\varepsilon}(L_n) > 2^{n^\varepsilon/c}$.

Proof (Mulmuley [Mul99]). Suppose $\mathsf{P} = \mathsf{NC}$. Then there is a boolean formula of size $2^{\text{poly}(\log n)}$ in the input bits such that the formula evaluates to 1 if and only if the corresponding instance of minimum-cost flow is a yes-instance, and 0 otherwise (this is unconditionally true of any function in NC). Treat this boolean formula as a formula over the integers via the usual translation $\neg x \mapsto 1 - x$ and $x \wedge y \mapsto xy$. However, the inputs to this integer formula are still the *bits* of the integer parameters in the original instance. To get an integer formula whose inputs are the input integer parameters themselves, we must get integer formulas that extract the bits of the integer inputs. This can be done for all inputs in $L_n \cap B(bn)$ by Lagrange interpolation; the resulting formula for each bit of each input has size at most $O((2^{bn})^2)$. Since inputs may be used more than once in the $2^{\text{poly}(\log n)}$ formula above, we may need to use $2^{\text{poly}(\log n)}$ copies of the bit-extraction formulae, resulting in a $\mathbb{Z}$ formula in the integer inputs of size at most $O(2^{2bn+\text{poly}(\log n)})$. By the efficient reduction of Liu and Regan [LR06], this formula is a $\mathbb{Z}$-projection of a determinant of basically the same size (just one larger). When $b < 1/(2c)$, this contradicts Conjecture 5. □

Remark 2. There is only one small change needed for the above proof to apply to the bit-bounded determinantal complexity analogue (instead of bdc^+) of Conjecture 5. Namely, rather than only applying Lagrange interpolation to the points of $L_n \cap B(bn)$ (those points with value 1), apply Lagrange interpolation to every input of bit-length at most bn, some of which will have value 1 (those in L_n) and some 0 (those not). Since the proof only used the bound $|L_n \cap B(bn)| \leq 2^{bn} = |B(bn)|$, the rest of the proof goes through unchanged. ◁

The same ideas that are used to prove Theorem 1 also show:

Theorem 7 (Theorem 7.4 of Mulmuley [Mul99])**.** *Conjecture 5 holds for all large enough b.*

Remark 3. Although we didn't give the proof here, the proof of Theorem 7 also goes through for the bdc-analogue (rather than bdc^+) of Conjecture 5 with only one minor change. The difference is that instead of only considering the hypersurface $\{X : \det(X) = 1\}$, one must also consider the hypersurface $\{X : \det(X) = 0\}$; this at most doubles the total degree of the hypersurface arrangement, and the rest of the proof goes through verbatim. ◁

Thus we have a single conjectured lower bound which:

1. Morally seems nearly equivalent to $\mathsf{P} \neq \mathsf{NC}$;

2. Holds true for sufficiently large—but still $O(n)$—bit-size by the methods of Mulmuley [Mul99]; and

3. For sufficiently small linear bit-size implies $\mathsf{P} \neq \mathsf{NC}$.

To approach $\mathsf{P} \neq \mathsf{NC}$, it is thus crucial to see what the difference is between these two settings—large and small (linear) bit-size—and what the exact limitations of these techniques are. We'll also see that similar reasoning will show us the limitations of these techniques for other problems as well, such as permanent versus determinant, and these limitations will in turn suggest a way forward.

The limitation which stands out the most, both for $\mathsf{P} \neq \mathsf{NC}$ and permanent versus determinant, is the extreme reliance of this technique on *degree*. In particular, the proof of the conjecture for large bit-size (Theorem 7) doesn't depend on the determinant at all, but only on its degree: the same proof also applies to the analogue of Conjecture 5 where projections of the determinant are replaced by projections of *any* function of the same degree. Such techniques can't work to prove the conjecture in its full strength, because one can construct in a straightforward fashion, using the techniques of Proposition 6, a function g of the same degree as the determinant, yet for which the conjecture fails when bit-bounded determinantal complexity is replaced by bit-bounded g-complexity [Mul99, Proposition 7.5]. This may be taken in the same spirit as the algebraic relativization or "algebraization" barrier [AW08] (see also [For94]).

When applied to the Permanent versus Determinant Conjecture, one could consider Conjecture 5 with the permanent in place of minimum-cost flow; the resulting conjecture is nearly equivalent to $\#\mathsf{P} \neq \mathsf{NC}$, and thus would imply the Permanent versus Determinant Conjecture (which is essentially $\#\mathsf{P} \neq \mathsf{NC}^2$). Here we'll see a different facet of the degree-limitation of these techniques.

The decision version of computing the permanent is to decide whether or not $\operatorname{perm}(X) \geq t$ where t is an input threshold. In tracing through the proof of Theorem 1, but with the permanent decision problem in place of minimum-cost flow, one eventually gets to a point where one needs to know that none of the polynomials computed at the branch instructions of the PRAM can be the permanent. But this is essentially the problem we began with: to show that a PRAM without limited bit operations can't compute the permanent efficiently. And indeed, we still don't know how to rule this out: PRAMs with limited bit operations can computed polynomials of the same degree as the permanent in time $O(\log n)$ on even a single processor. This naive argument may seem a bit circular, but we can lend more weight to it by delving a little further into the geometry involved in the proof.

Whereas in the proof for minimum-cost flow the set of yes-instances was defined as the integer points bounded by the graph of the optimum value of a hard parametrization, here the yes-instances are defined by the graph of the permanent. So we replace an arrangement of $2^{\Omega(n)}$ (bounded) hyperplanes with the single hypersurface $\{X : \operatorname{perm}(X) = t\}$. Although each hyperplane is of degree 1, an arrangement of $2^{\Omega(n)}$ hyperplanes has total degree $2^{\Omega(n)}$, and it is this geometry that cannot be mimicked by the polynomials computed by an efficient PRAM with limited bit operations. In contrast, in the case of the permanent, the hyperplane arrangement of total degree $2^{\Omega(n)}$ gets replaced by the single hypersurface $\{X : \operatorname{perm}(X) = t\}$ which has (total) degree n, since it's defined by the vanishing of the single degree-n polynomial $\operatorname{perm}(X) - t$. Although this explanation may (or may not!) have sounded more convincing, it is

also not quite enough, because it doesn't explain *why* the technique fails. However, I hope that it provides a pleasing and helpful intuitive geometric picture.

The ultimate reason these techniques don't work to rule out the geometry of a low-degree hypersurface like $\{X : \mathrm{perm}(X) = t\}$ is that all of the key geometric results—the Milnor–Thom bound, the Collins decomposition, and the general position argument—depended at most on the degree of the polynomials comptued at the branch instructions, and nothing else (it's in fact this degree, rather than the degree of the hypersurfaces considered in the previous paragraph, that's the key limitation). But this degree can be $2^{t(n)}$ (where $t(n) = O(\sqrt{n})$ is the bound on the parallel running time), larger than the degree of the permanent.

The question—both for P versus NC and permanent versus determinant—is then what other (geometric?) properties of NC or (nearly-equivalently, as in the motivation for Conjecture 5) the determinant can we use? In the context of permanent versus determinant, this point becomes even finer: what (geometric?) properties of the determinant distinguish it from the permanent?

One natural property which immediately suggests itself is that the *symmetries* of the determinant are linear-algebraic in nature, whereas the symmetries of the permanent are combinatorial in nature. This leads to the notion of symmetry-characterization, a conjecturally crucial phenomenon in GCT (the phenomenon is real, its cruciality is conjectured). Also, essentially all known lower bounds on the permanent use some *geometric* property of the permanent hypersurface beyond just its degree. But the geometric properties used so far have their limitations: most of them apply equally well to prove the same lower bound on the determinant, and the only one that doesn't (the famous $n^2/2$ bound [MR04, LMR10]) can easily be shown to be the optimal bound achievable using that property. Using algebraic geometry and representation theory, the GCT Program suggests a more systematic and global way of understanding the possible geometric properties of the determinant, and suggests a way of using this global understanding to prove lower bounds.

References

[AW08] Scott Aaronson and Avi Wigderson, *Algebrization: a new barrier in complexity theory*, STOC '08: 40th Annual ACM Symposium on Theory of Computing, ACM, New York, 2008, pp. 731–740.

[Car83a] Patricia J. Carstensen, *Complexity of some parametric integer and network programming problems*, Math. Programming **26** (1983), no. 1, 64–75.

[Car83b] Patricia June Carstensen, *The complexity of some problems in parametric linear and combinatorial programming*, Ph.D. thesis, University of Michigan—Ann Arbor, Ann Arbor, MI, 1983.

[Col75] George E. Collins, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, Automata theory and formal languages (Second GI Conf., Kaiserslautern, 1975), Lecture Notes in Computer Science, vol. 33, Springer, Berlin, 1975, pp. 134–183.

[For94] Lance Fortnow, *The role of relativization in complexity theory*, Bull. Eur. Assoc. Theoret. Comp. Sci. (1994), no. 52, 229–244.

[GG73] M. Golubitsky and V. Guillemin, *Stable mappings and their singularities*, Springer-Verlag, New York, 1973, Graduate Texts in Mathematics, Vol. 14.

[Kar93] David R. Karger, *Global min-cuts in* RNC*, and other ramifications of a simple min-cut algorithm*, SODA '93: 4th ACM–SIAM Symposium on Discrete Algorithms (New York), ACM, 1993, pp. 21–30.

[KM97] David R. Karger and Rajeev Motwani, *An* NC *algorithm for minimum cuts*, SIAM J. Comput **26** (1997), no. 1, 255–272.

[LMR10] J. M. Landsberg, Laurent Manivel, and Nicolas Ressayre, *Hypersurfaces with degenerate duals and the Geometric Complexity Theory Program*, arXiv:1004.4802 [math.AG], 2010.

[LR06] Hong Liu and Kenneth W. Regan, *Improved construction for universality of determinant and permanent*, Inform. Process. Lett. **100** (2006), no. 6, 233–237.

[Mil64] John Milnor, *On the Betti numbers of real varieties*, Proc. Amer. Math. Soc. **15** (1964), 275–280.

[MR04] Thierry Mignon and Nicolas Ressayre, *A quadratic bound for the determinant and permanent problem*, Int. Math. Res. Not. (2004), no. 79, 4241–4253.

[Mul93] Ketan D. Mulmuley, *Computational geometry: an introduction through randomized algorithms*, Prentice Hall, 1993.

[Mul99] Ketan Mulmuley, *Lower bounds in a parallel model without bit operations*, SIAM J. Comput **28** (1999), no. 4, 1460–1509 (electronic).

[Mur80] Katta G. Murty, *Computational complexity of parametric linear programming*, Math. Programming **19** (1980), no. 2, 213–219.

[Tho65] René Thom, *Sur l'homologie des variétés algébriques réelles*, Differential and Combinatorial Topology (A Symposium in Honor of Marston Morse), Princeton Univ. Press, Princeton, N.J., 1965, pp. 255–265.

[Zad73] Norman Zadeh, *A bad network problem for the simplex method and other minimum cost flow algorithms*, Math. Programming **5** (1973), 255–266.