Lecture 23

Lecturer: Madhu Sudan *Scribe: Rocco Servedio*

1 Introduction

In today's lecture we discussed some lower bounds on the complexity of computing various algebraic functions under several different nonuniform models of computation. Some useful references for today's material are Borodin and Munro's 1975 book "The Computational Complexity of Algebraic and Numeric Problems," Strassen's entry in the Handbook of Theoretical Computer Science on algebraic complexity theory, and von zur Gathen's survey paper.

2 Nonuniform Models

The models we will discuss are nonuniform in the sense that a different algorithm may be used for each different input size. Each of the models described below – straight-line programs, algebraic decision trees, and algebraic computation trees – fit into this framework.

2.1 Straight-line programs

Let R be an arbitrary ring and let $f : R^n \to R$ be some function. A straight-line program of length t for computing the value of $f(x_1, \ldots, x_n)$ is a sequence of t instructions, where the ith instruction is of the form

$$v_i \leftarrow y_1 \circ y_2.$$

Here each y_i is either some $v_j, j < i$, or is some input $x_j, 1 \le j \le n$, or is some scalar value (an element of R). The operator $\circ$ belongs to $\{+, -, \times\}$ (or may belong to $\{+, -, \times, \div\}$ if R is a field). The output of a straight-line program of length t on input $(x_1, \ldots, x_n)$ is the value of the final variable v_t, and the running time of the program is t. The complexity of such a program can be further broken down into the number of $+$ or $-$ operations performed and the number of $\times$ or $\div$ operations, and can be even further broken down into the number of scalar multiplications (of the form $r \times z$ with $r \in R$) versus nonscalar multiplications.

We will see that it is possible to prove some sharp bounds in this model. However, this is clearly a weak model since there is no way to do any kind of branching.

2.2 Algebraic decision trees

Unlike a straight-line program, which computes a function $f : R^n \to R$, an algebraic decision tree computes a boolean function $f : R^n \to \{0, 1\}$. An algebraic decision tree T is a rooted, complete (i.e. each node has either zero or two descendents) binary tree. Each leaf node is labelled with an element of $\{0, 1\}$ and each non-leaf node v is labelled with a predicate "$p_v(x_1, \ldots, x_n) = 0$," where p_v is a polynomial. (If p_v is restricted to be of total degree 1, then we that T is a *linear* decision tree, and if p_v is restricted to be of total degree at most d, we say that T is a *degree-d* algebraic decision tree.) If R is an ordered ring, the label of a non-leaf node can be of the form "$p_v(x_1, \ldots, x_n) \ge 0$." Evaluation proceeds down the tree from the root by going left if the predicate at a node is satisfied and going right if it is not. The complexity of a tree T is the depth of T.

This model clearly allows branching, and in that sense is stronger than the straight-line program model. However, the algebraic decision tree model is weaker in that straight-line programs can compute and use intermediate values, whereas algebraic computation trees can only maintain state information but cannot compute intermediate values.

2.3 Algebraic computation trees

Algebraic computation trees compute functions from R^n to R; they combine the branching ability of algebraic decision trees with the storage ability of straight-line programs. Unlike algebraic decision trees, algebraic computation trees can have non-leaf nodes of degree either 1 or 2. A node v of degree 1 is labelled with an instruction of the form "$v \leftarrow y_1 \circ y_2$," where each a_i is either some v' which was computed earlier on the path from the root to v or is some input x_j or is a scalar value, and $\circ$ is a basic arithmetic operation ($+, \times$,etc.) A node v of degree 2 is a branching node, labelled with a predicate of the form "$v' = 0$" where v' is an ancestor of v. (As with algebraic decision trees, predicates of the form "$v' \geq 0$" are allowed over ordered rings.) The complexity of an algebraic computation tree is the depth of the tree. Algebraic computation trees clearly generalize straight-line programs, and since polynomials can be built up using intermediate values and can be tested at branch nodes, they generalize algebraic decision trees as well.

3 Horner's Rule is Optimal

We begin by considering the straight-line program complexity of evaluating a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$. Horner's rule for evaluting such a polynomial is to write it in the following form:

$$p(x) = a_0 + x(a_1 + \ldots x(a_{n-1} + xa_n)\ldots).$$

This algorithm uses n additions and n multiplications. In 1954, A. Ostrowski posed the question of whether any other method of computing $p(x)$ could use fewer operations. He proved that n additions are necessary, but it was not until 1966 that V. Pan showed that n multiplications are necessary as well.

Before proving these results, let us state the problem clearly. The input is a sequence $a_0, \ldots, a_n$ of coefficients and an argument x, all taken from some infinite field K. The desired output is the value of $p(x) = \sum_{i=0}^{n} a_i x^i$. The questions we are interested in are the following: suppose that A is a straight-line program with operations $\{+, -, \times\}$ which computes the desired output for all inputs.

1. What is the minimum number of additions or subtractions which A can have?
2. What is the minimum number of multiplications?

3.1 Horner's Rule is optimal for additions

To see that at least n additions/subtractions are required, we consider the restriction of algorithm A to inputs which have $x = 1$. This yields a new algorithm A' which, on input $a_0, \ldots, a_n$, computes $a_0 + \ldots + a_n$; we now use induction on n to prove that A' must perform at least n additions or subtractions.

For the base case $n = 1$, note that if a straight-line program cannot perform any additions or subtractions then it can only compute values of the form $c \cdot a_0^{b_0} \cdot a_1^{b_1}$, where $c \in K$, a_0 and a_1 are the two inputs to the algorithm, and b_0 and b_1 are non-negative integers. Since the equation $c \cdot a_0^{b_0} \cdot a_1^{b_1} = a_0 + a_1$ cannot hold for all values of a_0, a_1 over the infinite field K, it follows that at least one addition/subtraction must be required.

For the inductive step, suppose that A' is a straight-line program which computes $a_0 + \ldots + a_n$. Let

$$v_i \leftarrow y_1 \pm y_2$$

be the first addition/subtraction which occurs in A'; as in the last paragraph, each y_i must be of the form $c \cdot a_0^{b_0} \cdots a_n^{b_n}$. Without loss of generality we can assume that a_n appears with nonzero exponent in the expression for y_2. Now if we consider the restriction of algorithm A' to inputs which have $a_n = 0$, we can eliminate the above instruction and obtain a new algorithm A'' which, on input $a_0, \ldots, a_{n-1}$, computes $a_0 + \ldots + a_{n-1}$. By the induction hypothesis, algorithm A'' must use at least $n-1$ additions/subtractions, and hence A' must use at least n additions/subtractions.

3.2 Horner's Rule is optimal for multiplications

We use the so-called *method of linear independence* to show that at least n multiplications are required to evaluate $p(x)$. To motivate this approach, note that if we consider the space $S = \{\sum c_i a_i : c_i \in K\}$ as a vector space over K, then the n elements $\{a_1, \ldots, a_n\}$ are linearly independent. The method of linear independence works by analyzing the amount of linear independence that can exist among the coefficients of polynomials (in x) that are generated as intermediate computations.

Throughout this section $\ell(a_1, \ldots, a_n)$ denotes a linear combination $\sum_{i=1}^n c_i a_i$ where $c_i \in K$. We begin with the following claim (the proof is left as an exercise for the reader):

Claim 1 *Let $\ell_1(a_1, \ldots, a_n), \ldots, \ell_u(a_1, \ldots, a_n)$ each be a linear combination of $a_1, \ldots, a_n$ as described above, and let $\ell(a_2, \ldots, a_n)$ be some linear combination of $a_2, \ldots, a_n$. For $1 \leq i \leq u$, let $\ell'_i(a_2, \ldots, a_n)$ be $\ell_i(\ell(a_2, \ldots, a_n), a_2, \ldots, a_n)$. Then the following holds: if the collection*

$$\{\ell_1(a_1, \ldots, a_n), \ldots, \ell_u(a_1, \ldots, a_n)\}$$

has rank k over K, then the collection

$$\{\ell'_1(a_2, \ldots, a_n), \ldots, \ell'_u(a_2, \ldots, a_n)\}$$

has rank at least $k-1$ over K.

We also require the following definition:

Definition 2 *We say that a multiplication of two elements of $K[a_1, \ldots, a_n, x]$ is* insignificant *if either both multiplicands belong to $K[x]$, or else one belongs to $K[a_1, \ldots, a_n]$ and the other belongs to K.*

Now we can prove the following lemma, which immediately yields the desired lower bound of n multiplications:

Lemma 3 *Let A be a straight-line program over $\{+, -, \times\}$ which, on inputs $a_0, \ldots, a_n, x$, computes*

$$\sum_{i=1}^{u} \ell_i(a_1, \ldots, a_n)x^i + r(x) + \ell_0(a_1, \ldots, a_n)$$

where each ℓ_i is a nontrivial linear combination of the a_i's and $r(x)$ is a polynomial. Let k be the rank of $\{\ell_1, \ldots, \ell_u\}$. Then A must have at least k significant multiplications.

Proof The proof is by induction on k. If $k = 1$, then $u \geq 1$ and then since $\ell_1 \neq 0$ at least one significant multiplication must be performed (without a significant multiplication we can only compute elements of the form $\ell_0(a_1, \ldots, a_n) + r(x)$).

For the inductive step, let A be as described in the statement of the lemma and let k be the rank of $\{\ell_1, \ldots, \ell_u\}$. The first significant multiplication performed by A must be of the form

$$\left[c_0(x) + \sum_{i=1}^{n} c_i a_i\right] \times \left[d_0(x) + \sum_{i=1}^{n} d_i a_i\right]$$

where $c_0(x), d_0(x) \in K[x]$ and $c_i, d_i \in K$ with some $c_i \neq 0$ and some $d_i \neq 0$. Without loss of generality we may assume that $c_1 \neq 0$. Now fix c to be some element of K, and consider the restriction of algorithm A to inputs $(a_0, a_1, \ldots, a_n, x)$ which satisfy

$$a_1 = \frac{c - c_0(x) - \sum_{i=2}^n c_i a_i}{c_1}.$$

Let us write $\frac{c-c_0(x)-\sum_{i=2}^n c_i a_i}{c_1}$ as $\ell(a_2,\ldots,a_n)+p(x)$. Then under this restriction, the program A computes

$$\sum_{i=1}^{u} \ell_i(\ell(a_2,\ldots,a_n)+p(x),a_2,\ldots,a_n)x^i + r(x) + \ell_0(\ell(a_2,\ldots,a_n)+p(x),a_2,\ldots,a_n) =$$

$$\sum_{i=1}^{u} \ell'_i(a_2,\ldots,a_n)x^i + r'(x) + \ell'_0(a_2,\ldots,a_n)$$

for $\ell'_1,\ldots,\ell'_u$ as defined in Claim 1 and some r', ℓ'_0. Since $c_0(x)+\sum_{i=1}^n c_i a_i = c$, under this restriction we obtain a new program A' on inputs $a_0, a_2, \ldots, a_n, x$ which has one fewer significant multiplication than A (since this first significant multiplication is no longer significant). By Claim 1 the rank of $\{\ell'_1,\ldots,\ell'_u\}$ is at least $k-1$, and thus by the induction hypothesis A' must have at least $k-1$ significant multiplications, which implies that A must have at least k significant multiplications. ■

4 Complexity of evaluating specific polynomials

In the last section, the inputs to our polynomial evaluation algorithm were the coefficients a_i of the polynomial as well as the argument x. A natural question is whether fewer operations are required for evaluating *specific* polynomials; so now the input to the algorithm is only the value x, and the output is the desired value of $p(x)=\sum_{i=0}^n a_i x^i$. (We can view this setup as one in which the algorithm is allowed to do "precomputation" using the coefficients a_i.) Clearly, there are specific polynomials which can be evaluated using fewer than n additions or multiplications, such as the polynomial $p(x)=x^n$ which can be computed using $\log n$ multiplications and no additions. In this section we will see that every polynomial can be evaluated in this framework using approximately $n/2$ multiplications and that there are polynomials for which any algorithm requires $n/2$ multiplications.

4.1 $n/2$ multiplications suffice

For a complete proof of this result, see the book by Borodin and Munro or Volume II ("Seminumerical Algorithms") of Knuth's series; here we provide only an outline. The basic idea is to use an alternative representation (other than a list $a_0, a_1, \ldots$ of coefficients) of the polynomial $p(x)$. More concretely, given any $b_1 \in K$, we can write $p(x)$ as

$$p(x) = (x^2 - b_1)p_1(x) + R_1(x),$$

where $\deg p_1 = \deg p - 2$ and $\deg R_1 \leq 1$. Continuing in this fashion, we can write

$$p_1(x) = (x^2 - b_2)p_2(x) + R_2(x)$$

and so on. The crux of the method is to find $b_1, b_2, \ldots$ so that $R_1, R_2, \ldots$ are each of degree 0 (i.e. $R_1 = c_1, R_2 = c_2, \ldots$). Then we can write

$$\begin{aligned} p(x) &= (x^2-b_1)p_1 + c_1 \\ &= (x^2-b_1)((x^2-b_2)p_2 + c_2) + c_1 \\ &\vdots \\ &= (x^2-b_1)(\ldots(x^2-b_{n/2}) + c_{n/2}) + \ldots + c_2) + c_1 \end{aligned}$$

Since we only need to compute x^2 once, this representation requires only $n/2+1$ multiplications.

4.2 $n/2$ multiplications can be necessary

To prove that $n/2$ multiplications can be required to evaluate certain polynomials of degree n, we use the *method of algebraic independence,* which is similar to the method of linear independence. Recall the following definitions:

Definition 4 *Let K, G be fields with $K \subseteq G$. Elements $g_1, \ldots, g_m \in G$ are said to be* algebraically dependent over K *if there is a nonzero polynomial $p \in K[y_1, \ldots, y_m]$ for which $p(g_1, \ldots, g_m) = 0$. A maximal set of algebraically independent elements of G is called a* transcendence basis *and the size of such a set is the* transcendence degree *of G over K.*

The following lemma is well known:

Lemma 5 *Let $H = K(\alpha_1, \ldots, \alpha_t)$ and let $h_1, \ldots, h_m \in H$. If $m > t$ then the elements $h_1, \ldots, h_m$ are algebraically dependent over K.*

We use this lemma to prove the following:

Theorem 6 *Let $p(x) = \sum_{i=0}^{n} a_i x^i \in K[x]$ where K is a field of characteristic 0. If there is a straight-line program over $\{+, -, \times\}$ which evaluates $p(x)$ using fewer than $(n+1)/2$ multiplications, then $a_0, \ldots, a_n$ are algebraically dependent over Q (the field of rational numbers).*

We note that this theorem immediately implies that certain polynomials require $n/2$ multiplications, e.g. by taking $a_0, \ldots, a_n$ to be algebraically independent real numbers.

Proof of Theorem 6: Suppose that A is a straight-line program which evaluates $p(x)$ using k multiplications. For $1 \le i \le k$ let S_i denote the result of the ith multiplication in the program. Since the program can only do additions and subtractions between computing S_i and S_{i+1}, we have that (taking $S_0 = x$)

$$S_1 = [c_1 + n_1 \cdot S_0] \times [d_1 + m_1 \cdot S_0]$$

where $c_1, d_1 \in K$ and n_1, m_1 are integers. More generally, we have that

$$S_i = [c_i + \sum_{j=0}^{i-1} n_{j,i} \cdot S_j] \times [d_i + \sum_{j=0}^{i-1} m_{j,i} \cdot S_j]$$

where each $c_i, d_i \in K$ and each $n_{j,i}, m_{j,i}$ are integers. It follows that

$$p(x) = \sum_{i=0}^{n} a_i x^i \in Q[x, c_1, \ldots, c_k, d_1, \ldots, d_k],$$

and hence that each $a_i \in Q(c_1, \ldots, c_k, d_1, \ldots, d_k)$. By Lemma 5, if $n + 1 > 2k$ then $a_0, \ldots, a_n$ are algebraically dependent over Q, which proves the theorem. ■

Specific polynomials with integer coefficients are known which require $n/2$ multiplications. In 1980, Strassen showed that any straight-line program which computes

$$p(x) = \sum_{i=0}^{n} 2^{2^i} x^i$$

must use at least $n/2$ multiplications. This is an impressive lower bound, since it is exponential in the number of bits required to specify n.

5 Lower Bounds for Algebraic Computation Trees

Thus far we have been interested in the complexity of evaluating a polynomial at a particular argument value x. In the problem of computing the Fourier transform of a fixed polynomial $a_0, \ldots, a_n$, the inputs are $n+1$ argument values $x_0, \ldots, x_n$, and the goal is to evaluate the value of the polynomial at all $n+1$ argument values, i.e. to compute

$$\{\sum_{i=0}^{n} a_i x_0^i, \sum_{i=0}^{n} a_i x_1^i, \ldots, \sum_{i=0}^{n} a_i x_n^i\}.$$

Fast algorithms are known which can solve this problem in $O(n \cdot \log n \cdot poly(\log\log n))$ steps for arbitrary inputs x_i, and in the special case when the x_i's are the complex $(n+1)$th roots of unity, an $O(n \log n)$ algorithm is known.

Motivated by these fast algorithms, Strassen studied the question of whether even faster algorithms could exist for this problem. He showed an $\Omega(n \log n)$ lower bound in the straight-line model, which can be extended to an $\Omega(n \log n)$ lower bound in the algebraic computation tree model. In this section we outline the proof of the $\Omega(n \log n)$ lower bound in the algebraic computation tree model. The proof requires some machinery from algebraic geometry, which we review briefly below.

5.1 Preliminaries

For the rest of today's lecture we will assume that the ground field K is algebraically closed. In future lectures we will consider fields such as $\mathbf{R}$ which are not algebraically closed.

Recall that if $f_1, \ldots, f_s \in K[x_1, \ldots, x_n]$, then the *variety* generated by these polynomials is

$$V(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in K^n \mid f_1(a_1, \ldots, a_n) = \cdots = f_s(a_1, \ldots, a_n) = 0\}.$$

The *dimension* of a variety V, written $\dim(V)$, is the maximum dimension of a linear subspace $L \subseteq K^n$ such that $|L \cap V|$ is finite. The *degree* of a variety V, written $\deg(V)$, is the maximum number of points than can occur in any such intersection. It is easy to verify that if V, W are varieties then $V \cap W$ is a variety as well; Bezout's theorem tells us that

$$\deg(V \cap W) \leq \deg(V) \cdot \deg(W).$$

In our informal exposition here, we will use the above definitions and will use varieties over K^n. However, for a rigorous proof of Strassen's result, one should work over the projective space P^n rather than K^n (we will see why this is necessary in the next section). The space P^n has as its points equivalence classes of points in $K^{n+1}/\{(0, \ldots, 0)\}$ under the equivalence relation $\equiv$, where $(x_0, \ldots, x_n) \equiv (y_0, \ldots, y_n)$ iff for some $0 \neq \lambda \in K$ it is the case that $(x_0, \ldots, x_n) = (\lambda y_0, \ldots, \lambda y_n)$. We write $\langle x_0, \ldots, x_n \rangle$ to represent the equivalence class to which $(x_0, \ldots, x_n)$ belongs. It would take us too far afield to develop all of the necessary machinery for discussing varieties over P^n; a more thorough exposition can be found in the book by Borodin and Munro.

5.2 Algebraic computation trees and varieties

In this section we will establish the following bound, which relates the depth of an algebraic computation tree to the degree of the variety of the function which the tree computes.

Lemma 7 *Let $f_1, \ldots, f_m \in K[x_1, \ldots, x_n]$ and let $f : K^n \to K^m$ be defined by $f(\overline{x}) = (f_1(\overline{x}), \ldots, f_m(\overline{x}))$. Let $V \subseteq K^{n+m}$ be the variety*

$$V = \{(\overline{x}, f(\overline{x})) \mid \overline{x} \in K^n\}.$$

If $\deg(V) \geq D$, then any algebraic computation tree for f must have depth at least $\log_4 D$.

Proof sketch: Let T be an algebraic computation tree of depth h which computes f, and let P_1 be some path through T from the root to a leaf. A given node on such a path is:

- either a predicate of the form "$v_i = 0$" or "$v_i \neq 0$",
- or an assignment statement of the form "$v_i = v_j \circ v_k$" where $\circ$ is some operation from $\{+, -, \times, \div\}$ and $i, j, k \leq h$.

The key point is that each node on this path corresponds to some polynomial equation of degree at most 2 over the variables $x_1, \ldots, x_n, v_1, \ldots, v_h$. This is clearly the case for a statement of the form "$v_i = 0$;" for a statement of the form "$v_i \neq 0$" we can employ Rabinovich's trick, i.e. introduce a new variable w_i and use the equation $w_i \cdot v_i = 1$ (so in fact our new set of variables will be $\{x_1, \ldots, x_n, v_1, \ldots, v_h, w_1, \ldots, w_h\}$). For an assignment statement "$v_i = v_j \circ v_k$" we use the equation $v_i - v_j \circ v_k = 0$ if $\circ \in \{+, -, \times\}$ and we use the equation $v_i \cdot v_k - v_j = 0$ if $\circ$ is $\div$.

Thus, the set of those points $(x_1, \ldots, x_n, v_1, \ldots, v_h, w_1, \ldots, w_h)$ which correspond to a particular path through the tree will be the variety generated by a set of at most h polynomials:

$$\begin{aligned} p_1(x_1, \ldots, x_n, v_1, \ldots, v_h, w_1, \ldots, w_h) &= 0 \\ &\vdots \\ p_h(x_1, \ldots, x_n, v_1, \ldots, v_h, w_1, \ldots, w_h) &= 0. \end{aligned}$$

By Bezout's theorem, the degree of this variety can be at most 2^h (since the degree of each polynomial is at most 2). However, this is a variety over the $(n+2h)$-dimensional space of points $(x_1, \ldots, x_n, v_1, \ldots, v_h, w_1, \ldots, w_h)$, which is a larger space than we are really interested in. We would like to consider the set of points

$$V_{P_1} = \{(\overline{x}, f(\overline{x})) \mid (x_1, \ldots, x_n) \text{ goes down path } P_1\}.$$

This is the projection of the original variety onto the coordinates $x_1, \ldots, x_n, v_{i_1}, \ldots, v_{i_m}$, where $i_1, \ldots, i_m$ are such that $(v_{i_1}, \ldots, v_{i_m}) = f(\overline{x})$. To analyze such a variety we require the following fact:

Fact 8 *Let $\Pi : P^n \to P^{n-1}$ be the mapping which takes $\langle x_0, \ldots, x_n \rangle$ to $\langle x_0, \ldots, x_{n-1} \rangle$. Let V, W be varieties over P^n, P^{n-1} respectively. Then $\Pi(V), \Pi^{-1}(W)$ are varieties over P^{n-1}, P^n respectively and*

$$\begin{aligned} \deg(\Pi(V)) &\leq \deg(V) \\ \dim(V) - 1 &\leq \dim(\Pi(V)) \leq \dim(V) \\ \deg(\Pi^{-1}(W)) &= \deg(W) \\ \dim(\Pi^{-1}(W)) &= \dim(W) + 1. \end{aligned}$$

(We note that projections of varieties over K^n are not necessarily themselves varieties; this is why we must work over P^n.) Using this fact, we have that the degree of V_{P_1} is also at most 2^h. But recall that V_{P_1} is the variety generated by a single path P through the computation tree. Since the tree T has depth h, there are at most 2^h different paths through the computation tree, and hence we can write

$$V = V_{P_1} \cup \cdots \cup V_{P_{2^h}} = \{(\overline{x}, f(\overline{x}))\}.$$

Since $\deg(A \cup B) \leq \deg(A) + \deg(B)$ for varieties A, B, we have that $\deg V \leq 2^h \cdot 2^h = 4^h$. ■

5.3 A lower bound for the Fourier transform

Using Lemma 7 we can show the following, which yields the desired $\Omega(n \log n)$ complexity lower bound for the Fourier transform.

Claim 9 *Let $p \in K[x]$ be a univariate polynomial of degree m. Then any algebraic computation tree which evaluates $p(x_1), \ldots, p(x_n)$ must have depth $\Omega(n \log m)$.*

Proof sketch: Let

$$V = \{(x_1, \ldots, x_n, p(x_1), \ldots, p(x_n)) \mid (x_1, \ldots, x_n) \in K^n\}.$$

By Lemma 7, it suffices to show that $\deg(V) \geq m^n$. Let $c \in K$ be such that the equation "$p(x) - c = 0$" has m distinct roots $\alpha_1, \ldots, \alpha_m$ (we leave it as an exercise for the reader to show that such a c must exist). For $1 \leq i \leq n$, define

$$H_i = \{(y_1, \ldots, y_n, z_1, \ldots, z_{i-1}, c, z_{i+1}, \ldots, z_n) \mid y_j \in K, z_j \in K\}.$$

Each H_i is a variety of degree 1. Let $V' = V \cap (\cap_{i=1}^n H_i)$; by Bezout's theorem,

$$\deg(V') = \deg(V \cap (\cap_{i=1}^n H_i)) \leq \deg(V) \cdot \left(\prod_{i=1}^n \deg(H_i)\right) = \deg(V).$$

However,

$$V' = \{(r_1, \ldots, r_n, c, \ldots, c) \mid r_i \in \{\alpha_1, \ldots, \alpha_m\}\}.$$

Hence $|V'| = m^n$. Since V' is contained a linear subspace of K^{2n}, we have that $\deg(V') = m^n$, which proves the claim. ■