

Lecture 14 : 3.08.06

Lecturer: Richard Karp

Scribe: Juliet Holwill & Benjamin I. P. Rubinstein

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Last lecture saw the introduction of Groebner bases as a powerful tool – when used with the simple division algorithm – for membership testing in multivariate polynomial ideals. In particular we proved the Hilbert Basis Theorem for multivariate ideals, that every such ideal has a finite (Groebner) basis. We also explored construction of Groebner bases through Buchberger’s algorithm. In this lecture we complete the present discussion of systems of polynomials and polynomial equations with reduced Groebner bases and a convenient back-substitution method for solving systems of polynomial equations using Groebner bases. We then proceed to sorting networks, which consist of a sequence of layers of parallel pairwise comparisons that can be used for efficient sorting. After focusing on Batcher networks – sorting networks that are $\Theta(\log^2 n)$ deep – we introduce the AKS sorting network.

Recall that a *Groebner basis* of an ideal $I \subseteq K[x_1, \dots, x_n]$ is a basis $\{g_1, \dots, g_t\}$ of I such that $LT(I) = \langle LT(g_1), \dots, LT(g_t) \rangle$ where for any $f \in K[x_1, \dots, x_n]$ the leading term (taken with respect to the chosen lexicographic ordering on $\{x_1, \dots, x_n\}$) of f is denoted by $LT(f)$, and $LT(I) = \{LT(f) \mid f \in I\}$.

14.1 Reduced Groebner Bases

After running the Groebner basis algorithm, we may get more elements than we really require, so we would like to prune the returned basis in order to remove redundant elements. We can use the following lemma for this purpose.

Lemma 14.1 *Let G be a Groebner basis for the polynomial ideal I . Let $p \in G$ be a polynomial such that $LT(p) \in \langle LT(G - \{p\}) \rangle$. Then $G - \{p\}$ is also a Groebner basis for I .*

Proof: The result follows from noting that $I = \langle G - \{p\} \rangle$. ■

Definition 14.2 *A reduced Groebner basis for ideal I is a Groebner basis G of I satisfying two additional properties:*

- (i) $\forall p \in G$, the coefficient of $LT(p)$ is 1; and
- (ii) $\forall p \in G$, no monomial of p lies in $\langle LT(G) - \{p\} \rangle$.

Condition (i) simply requires multiplying through by an appropriate constant in the field K to attain leading coefficients of unity. To prove that a reduced Groebner basis exists for ideal I , it can be shown that a reduced basis can be generated from a Groebner basis G using the following two steps:

1. If for some $p \in G$, $LT(p) \in \langle LT(G) - \{p\} \rangle$, then remove p from G ; and

2. If some other (non-leading) monomial term in p is in $\langle LT(G) - \{p\} \rangle$, then replace p by $\bar{p}^{G-\{p\}}$ – the remainder on division of p by the set $G - \{p\}$ – thus removing such monomials generated by $LT(G) - \{p\}$.

Repeated application of these two steps always produces a reduced Groeber basis.

Theorem 14.3 *Every polynomial ideal has a unique reduced Groeber basis (relative to a fixed monomial ordering).*

A consequence of this result is that it provides a standard for computational algebra systems. That is, if you have a Maxima or Macsyma system¹ and a system under development, you can check the performance of your system against the presumably well-tuned established one.

14.1.1 Solving Polynomial Equations using Groeber Bases

Consider the following three polynomial equations.

$$\begin{aligned} f_1: & \quad x^2 - y - z - 1 = 0 \\ f_2: & \quad x - y^2 - z - 1 = 0 \\ f_2: & \quad x - y - z^2 - 1 = 0 \end{aligned}$$

To solve these equations we consider the ideal $I = \langle f_1, f_2, f_3 \rangle$.

$$I = \langle x^2 - y - z - 1, x - y^2 - z - 1, x - y - z^2 - 1 \rangle \quad (14.1)$$

It is easy to see that if we generate any other basis for this ideal, then the solution set of the equations in the new basis will be the same as the solutions for the original one. This is because each polynomial in the new basis is a combination of the polynomials in the original basis and vice versa. Computing a Groeber basis for I with respect to the ordering $x \succ y \succ z \succ w$, we get

$$\begin{aligned} g_1: & \quad x - y - z^2 - 1 = 0 \\ g_2: & \quad y^2 - y - z^2 - z = 0 \\ g_3: & \quad 2yz^2 - z^4 - z^2 = 0 \\ g_4: & \quad z^6 - 4z^4 - 4z^3 - z^2 = 0 \end{aligned}$$

The system of equations $g_1 = g_2 = g_3 = g_4$ must then be equivalent to (14.1).

Although this looks more complicated than the original set of equations, it has the attractive property that is similar to the property that gaussian elimination has for solving linear equations. Namely, we can solve the equations using back substitutions. Looking at g_4 , we notice that it is the only g_i depending on variable z only. So we can solve $g_4(z) = 0$ for z ; then for each solution \tilde{z} attained substitute and solve the now single-variable equations $g_2 = 0, g_3 = 0$ in y and finally for each solution (\tilde{y}, \tilde{z}) solve $g_1 = 0$ for x . We can generate the whole solution set by this back substitution elimination process – at each step solving only single-variate equations. It turns out that this is always possible with Groeber bases, as shown by the following theorem.

¹Both are popular computational algebra systems, the former is free and based on the latter commercial one.

Definition 14.4 Given an ideal $I = \langle f_1, f_2, \dots, f_s \rangle \subset K[x_1, x_2, \dots, x_n]$, the l^{th} elimination ideal of I is defined as $I_l = I \cap K(x_1, \dots, x_n)$. In words, we are picking out just those polynomials in I within which only the variables x_1, \dots, x_n are present.

Theorem 14.5 Let I be an ideal with Groebner basis G with respect to the ordering $x_1 \succ \dots \succ x_n$. Then

- (i) I_l is an ideal for each $l \in \{1, \dots, n\}$; and
- (ii) $G_l = G \cap K[x_1, \dots, x_n]$ is a Groebner basis for I_l .

With this theorem we can back-solve by repeating:

1. Solve the system $g_i = 0$ for $g_i \in G_n$;
2. For each solution, solve the system $g_i = 0$ for $g_i \in G_{n-1}$;
3. Continue for $g_i \in G_{n-2}$ then G_{n-3} , down to $G_1 = G$.

This theorem makes the Groebner basis very convenient for solving systems of equations: you can always work backwards provided that a lexicographic ordering is used. Groebner bases are not a cureall, however. They can be very ‘ugly’ so it is not always the case that they give a great advantage. In particular, there is very little in the way of complexity theory for Groebner bases.

14.2 Sorting Networks

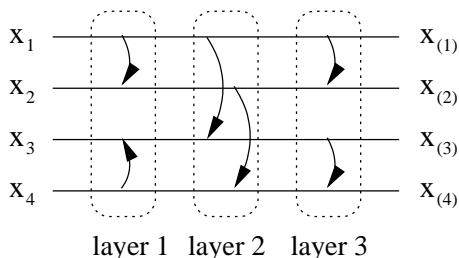


Figure 14.1: An example sorting network. The input sequence x_1, x_2, x_3, x_4 enters the network on the left, directed comparator segments indicate swaps (each head indicated the destination of the comparison’s maximum), swaps are collected into independent layers, and after the last layer is complete the sorted output $x_{(1)}, x_{(2)}, x_{(3)}, x_{(4)}$ is returned. This network has depth 3 and a size of 6.

A *sorting network* (see Figure 14.1) consists of n horizontal lines, with directed vertical segments joining pairs of lines. These vertical segments represent *comparators*. Elements x_1, \dots, x_n enter at the left and they flow from left to right along the lines. At each comparator, they get compared and the larger of the two entering elements comes out at the head of the arrow, and the smaller of the two exists at the tail. For all sorting networks, it is a requirement that regardless of the order in which the elements enter on the left, they emerge at the right in increasing order from top to bottom.

The *size* of the network is defined as the number of comparators. The *depth* of a sorting network is the maximum number of comparators any element can pass through. The depth is equal to the number of layers.

Starting from the left, each *layer* consists of a maximal cardinality set of comparators that pairwise do not have any lines in common. Note that a network can always be transformed to an equivalent network where all comparators point downwards.

Natural parameters to consider here are the minimum size of a network needed to sort, and the minimum depth. Given that any comparison-based sorting algorithm must perform at least $\Omega(n \log n)$ comparisons, the size of any sorting network must be $\Omega(n \log n)$, and therefore its depth must therefore be $\Omega(\log n)$.

14.2.1 Batcher Networks

Batcher showed that one can achieve a sorting network of size $\mathcal{O}(n \log^2 n)$ and depth $\mathcal{O}(\log^2 n)$ [B68]. To analyze these sorting networks, we require the *0-1 principle*.

Lemma 14.6 *A comparator network is a sorting network iff it correctly sorts all binary sequences.*

Proof: Let N be a comparator network with input sequence (x_1, \dots, x_n) . Then the following are equivalent:

- (i) N correctly sorts (x_1, \dots, x_n) ;
- (ii) $\forall y, \{x_i | x_i \leq y\}$ precedes $\{x_i | x_i > y\}$ in top-to-bottom order of output; and
- (iii) $\forall y$, the network correctly sorts $(a_1(y), \dots, a_n(y))$, where $a_i(y) = \mathbf{1}[x_i > y]$.

If we can prove the equivalence of these three, we have proved the property because we have shown that the network always sorts correctly. We have shown that 1 is equivalent to 2, now to prove 2 is equivalent to 3, you only need to keep track of which numbers are smaller or larger, the actual values don't matter. This is equivalent to treating them as 0s or 1s, so 2 is equivalent to 3. ■

Definition 14.7 *A sequence of 0's and 1's is called bitonic if it is of the form $0^a 1^b 0^c$ or $1^a 0^b 1^c$, where $a, b, c \geq 0$.*

The next results can be proven with a small case analysis.

Lemma 14.8 *Let $(a_1, \dots, a_{2m}) \in \{0, 1\}^{2m}$ be a bitonic sequence of even length. For $i \in \{1, \dots, m\}$, let $b_i = \min(a_i, a_{m+i})$ and $c_i = \max(a_i, a_{m+i})$. Then (b_1, \dots, b_m) and (c_1, \dots, c_m) are bitonic, and for all i, j , $b_i \leq c_j$.*

Example 14.9 *Let $m = 8$, so that the length of the sequence is 16. Suppose the sequence is of the $1^a 0^b 1^c$. Note that the a_i sequence can be split over 2 lines so that the min and max of each position (over the two lines) give the b_i and c_i . For example*

(a_i)	1	1	1	1	1	0	0	0	
	0	0	1	1	1	1	1	1	
(b_i)	0	0	1	1	1	0	0	0	min
(c_i)	1	1	1	1	1	1	1	1	max

Observe that the max is all 1's and the min sequence is bitonic. This effect will always occur: if at least half the a_i are all 1's then the upper half of the output is all 1's, and if at least half are 0's then the lower half will be all 0's.

Given this result, we can sort the bitonic sequence (a_1, \dots, a_{2^k}) by divide and conquer as follows (see Figure 14.2):

- (1) For $i = 1, \dots, 2^{k-1}$, compare a_i with $a_{2^{k-1}+i}$, and let $b_i = \min(a_i, a_{2^{k-1}+i})$ and $c_i = \max(a_i, a_{2^{k-1}+i})$.
- (2) Recursively, in parallel, sort the two bitonic sequences $(b_1, \dots, b_{2^{k-1}})$ and $(c_1, \dots, c_{2^{k-1}})$.

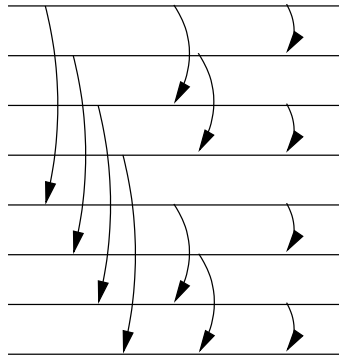


Figure 14.2: A network for sorting a bitonic sequence of length 8 by the recursive method described in the text. The network has depth 3.

It is immediate that the number of layers required to sort a bitonic sequence of length 2^k using this method is k . Now that we know how to sort a bitonic sequence of length 2^k using a network of depth k , we can construct a sorting network that operates on any input as follows:

- (1) Transform the input sequence into a bitonic sequence by sorting its first half into increasing order, and concurrently sorting its second half into decreasing order.
- (2) Sort the resulting bitonic sequence.

This algorithm is known as Batcher's algorithm [B68].

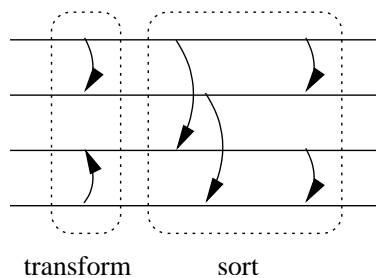


Figure 14.3: A Batcher network for sorting an arbitrary sequence of 4 elements, beginning with a phase that transforms the sequence into a bitonic form and then a recursive sorting phase.

Lemma 14.10 *The depth (or time) required by Batcher's algorithm to sort an arbitrary sequence of $n = 2^k$ elements is $\frac{\log n(\log n + 1)}{2} = \Theta(\log^2 n)$.*

Proof: Let $T(k)$ be the depth required to transform an input sequence of length 2^k into a bitonic sequence, $S(k)$ be the depth required to sort an arbitrary sequence of length 2^k , and $B(k)$ be the depth required to sort a bitonic sequence of length 2^k . Then $B(k) = k$, $T(k) = S(k - 1)$, $S(k) = T(k) + B(k)$ and $S(0) = 0$. Solving gives $S(k) = S(k - 1) + k$, $S(0) = 0$, which implies that $S(k) = \frac{k(k+1)}{2}$ for all $k \geq 0$. ■

14.2.2 AKS Networks

Batcher’s algorithm is the best choice in practice, but its discovery left the question of whether there exist sorting networks of depth $\mathcal{O}(\log n)$. Ajtai, Komlós, and Szemerédi found a network (the AKS network) that did have this property [AKS83], but it is not practical to use because the constant hidden by the big-O is around 6100. We shall instead look at a refinement of their result due to Paterson [P90].

Noting that a natural way to sort is by repeated splitting, the idea is to construct a complete binary tree with $n = 2^k$ leaves and send the input sequence down this tree so that the i^{th} smallest element reaches the i^{th} leaf from the left – reading off from left-to-right then produces the sorted sequence. Upon reaching a node, a set of elements is split with the ‘smaller’ (‘larger’) elements being sent down the left (right) subtree. A perfect splitter that split each set to the smallest (largest) half would require depth $\Omega(\log n)$ and an entire structure of depth $\Omega(\log^2 n)$, so the AKS network instead employs an approximate halver (of constant depth) that makes a mistake on a small proportion of its input which are later sent to the correct child. The algorithm based on such a halver would need of course need to be changed to include a recovery mechanism taking misdirected keys to the path they were following.

Definition 14.11 Let $0 < \epsilon < 1$ be a design parameter. Given (even) m elements an ϵ -halver splits the elements into two sets each of size $\frac{m}{2}$, L and R , such that for all $k \leq \frac{m}{2}$ at most ϵk of the k smallest (largest) keys incorrectly go to R (L).

An ϵ -halver can be realized in $D = \mathcal{O}(1)$ layers (constant in m). We will use an (m, ϵ) -bipartite expander of degree D : a bipartite graph with $\frac{m}{2}$ D -degree vertices in each partition such that for all $k \leq \frac{m}{2}$ every set of $\lceil \epsilon k \rceil$ vertices contained in one part has at least $(1 - \epsilon)k + 1$ neighbors in the other part. The existence of such graphs can either be proven with the probabilistic method or explicitly produced with a known explicit construction.

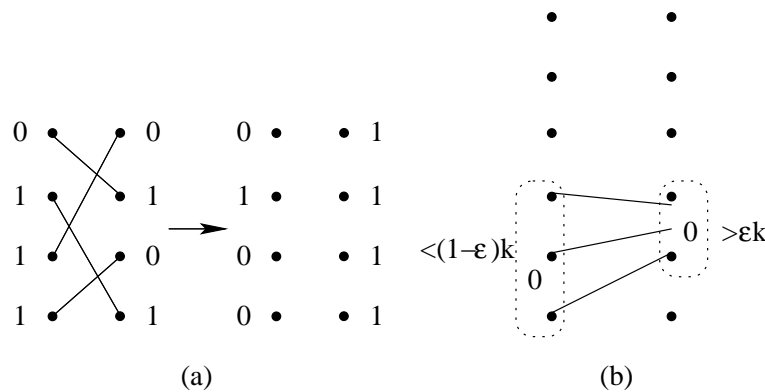


Figure 14.4: Paterson’s variant on the AKS sorting network: (a) by setting up a matching we can compare and better sort the L and R outputs of a halver; and (b) the ‘bad’ event in a graph to be matched. Avoiding this event guarantees an ϵ -halver.

Placing the elements in the left and right part of the bipartite graph, a good D matching of the graph is

one for which 1's in L are matched with 0's in R – these matched vertices are then compared and swapped (see Figure 14.4(a)). Provided that not too many keys can be incorrectly binned a D matching will produce the desired ϵ -halver. The bad event is illustrated in Figure 14.4(b) which is exactly the opposite of the (m, ϵ) -bipartite expander property.

Next time we will see further details of Paterson's version of the AKS sorting network.

References

- [AKS83] M. AJTAI, J. KOMLÓS, AND E. SZEMERÉDI, “An $O(n \log n)$ Sorting Network,” *Combinatorica* **3**(1) (1983), pp. 1–19.
- [B68] K.E. BATCHER, “Sorting Networks and their Applications,” *AFIPS Proc. Spring Joint Computer Conference* **32** (1968), pp. 307–314.
- [CLRS01] T.H. CORMEN, C.E. LEISERSON, R.L. RIVEST AND C. STEIN, *Introduction to Algorithms*, MIT Press, Second Edition (2001).
- [P90] M.S. PATERSON, “Improved Sorting Network with $O(\log n)$ Depth,” *Algorithmica* **5**(1) (1990).