Scribe Notes for *Algorithmic Number Theory*
Class 7—May 27, 1998

Scribes: Scott A. Guyer, Duxing Cai, and Degong Song

Abstract

Today's first topic is parallel complexity, which is a continuation of the last day's topic. The second topic is calculating the greatest common divisor. Some results about the greatest common divisor are given including the Euclidean algorithm.

1 Parallel Complexity

Parallel algorithms use more hardware simultaneously to reduce time complexity. We will use boolean circuits as a model.

Example 1.1 Consider the following binary operation.

```
  c1    c0
        a1    a0
    +   b1    b0
-----------------
  s2    s1    s0
```

Writing this in terms of boolean expressions, we have:

$$\begin{aligned} s_0 &= (a_0 \wedge \sim b_0) \vee (\sim a_0 \wedge b_0), \\ c_0 &= a_0 \wedge b_0, \\ s_1 &= (a_1 \wedge b_1 \wedge c_0) \vee (a_1 \wedge \sim b_1 \wedge \sim c_0) \vee (\sim a_1 \wedge b_1 \wedge \sim c_0) \vee (\sim a_1 \wedge \sim b_1 \wedge c_0), \\ s_2 &= c_1 = (a_1 \wedge b_1) \vee (a_1 \wedge c_0) \vee (b_1 \wedge c_0). \end{aligned}$$

We can represent these kind of relations with directed acyclic graphs (cf. Figure 1). Each vertex has indegree 0, 1, or 2. Vertices of indegree 0 are called inputs. The other vertices are labeled with boolean operations chosen from the set $\{\sim, \vee, \wedge\}$. Also, some vertices are distinguished as outputs.

Such a graph is called a boolean circuit. If it has n inputs and m outputs, it will represent a function from $\{0,1\}^n$ to $\{0,1\}^m$. Given an input, signal propagation occurs in the circuit until an output is obtained.

Usually we can measure the cost of a boolean circuit by two parameters, size and depth. *Size* is the number of nodes in graph, it describe the space complexity. *Depth* is the longest path from an input to an output. It reflects the propagation delay of gates in the circuit. For example, the circuit in Figure 1 has size 28 and depth 5.

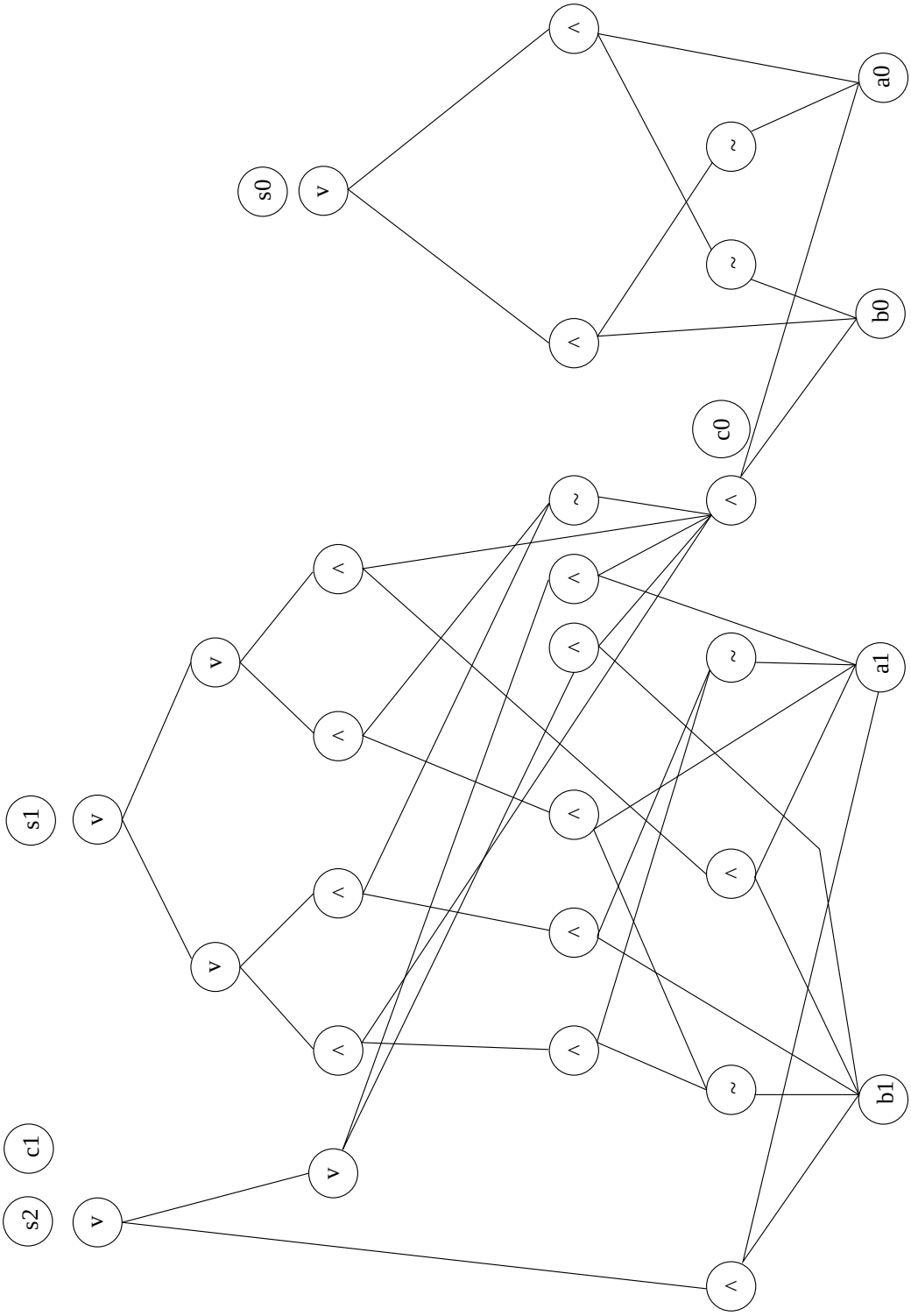

Figure 1: Boolean circuit for Example 1.1.

To discuss functions whose arguments have varying length, what we need is a family of circuits. This is a sequence of circuits C_0, C_1, C_2, ..., where C_n takes n inputs. This family of circuits defines a function:

$$f : \{0,1\}^* \longrightarrow \{0,1\}^*.$$

A circuit family $C_0, C_1, C_2, \ldots$ is *log-space uniform* if there is a deterministic (Turing machine) algorithm that takes inputs 1^n and returns a representation of C_n using only $O(\log n)$ space.

The complexity class NC consists of functions $f : \{0,1\}^* \longrightarrow \{0,1\}^*$ such that for each f there is a log-space uniform family of circuits to compute it and there are polynomials p and q such that C_n has size $\leq O(p(n))$ and depth $\leq O(q(\lg n))$.

Example 1.2 Addition can be accomplished in linear size and $O(\lg n)$ depth. This is based on a technique called carry look-ahead. The proof of this is left as an exercise (see Exercise 27 in [1]).

It is believed by complexity theorists that NC is strictly contained in NP.

2 Greatest Common Divisor

The *greatest common divisor* of two integers u and v, written $d = \gcd(u, v)$, is defined by these properties:

1. If $u \neq 0$ and $v \neq 0$, then $d > 0$;
2. $d|u$ and $d|v$;
3. If $e > 0, e|u$, and $e|v$, then $e|d$; and
4. $\gcd(u, 0) = \gcd(0, u) = |u|$.

The *least common multiple* of two integers u and v, written $f = \mathrm{lcm}(u, v)$, is defined by these properties:

1. If $u \neq 0$ and $v \neq 0$, then $f > 0$;
2. $u|f$ and $v|f$;
3. If $e > 0, u|e$, and $v|e$, then $f|e$; and
4. $\mathrm{lcm}(u, 0) = \mathrm{lcm}(0, u) = 0$.

It is interesting to note the relationship between the greatest common divisor and the least common multiple. In particular, if $(u, v) \neq (0, 0)$, then

$$\mathrm{lcm}(u, v) = \frac{|uv|}{\gcd(u, v)}.$$

Theorem 1 *For every $c, u, v \in \mathbb{Z}$, $\gcd(u, v) = \gcd(u, v + cu)$.*

Proof. If $u = 0$, then obviously $\gcd(u, v) = \gcd(u, v + cu)$.

Now suppose $u \neq 0$. Let $d = \gcd(u, v)$. If $v = 0$, then

$$d = \gcd(u, v) = |u| = \gcd(u, cu) = \gcd(u, v + cu).$$

Finally, suppose $v \neq 0$. In this case, $d > 0$, $d|u$ and $s|(v + cu)$. Suppose $e > 0$, $e|u$ and $e|(v + cu)$. Then $e|(v + cu - cu)$, i.e., $e|v$. From $e|u$ and $e|v$ we see that $e|d$. So, by definition, we have $d = \gcd(u, v + cu)$. □

Corollary 1 ([1, p. 67]) *For all* $u, v \in \mathbb{Z}$, $\gcd(u, v) = \gcd(v, u \bmod v)$.

Proof. If $v = 0$, then $u \bmod v = |u|$ and the conclusion is obviously true. If $v \neq 0$, then $u \bmod v = u - v\lfloor \frac{u}{v} \rfloor$. From the above theorem, it is easy to get this corollary. □

The above Corollary implies a method for calculating $\gcd(u, v)$ known as the Euclidean algorithm.

```
EUCLID(u, v)
1  if v = 0
2     then return |u|
3     else  return EUCLID(v, u mod v)
```

Example 2.1 EUCLID(216,183).

$$\begin{aligned} (183, 216 \bmod 183) &= (183, 33) \\ (33, 183 \bmod 33) &= (33, 18) \\ (18, 33 \bmod 18) &= (18, 15) \\ (15, 18 \bmod 15) &= (15, 3) \\ (3, 15 \bmod 3) &= (3, 0) \end{aligned}$$

EUCLID(3,0) returns 3, so $\gcd(216, 183) = 3$. In this example, we get the following sequence:

$$216 > 183 > 33 > 18 > 15 > 3 > 0.$$

The time complexity of the Euclidean algorithm is closely related to the speed at which this sequence approaches 0.

2.1 Crude Complexity Analysis

We can devise a crude complexity analysis of the Euclidean algorithm as follows. First, we claim that after any two consecutive divisions, the number in the second slot decreases by at least a factor of $\frac{1}{2}$.

Each division in the Euclidean algorithm requires $O((\lg u)(\lg v))$ bit operations. The number of divisions is $O(\max\{\lg u, \lg v\})$. Hence, the total bit complexity is no more than

$$O(\max\{\lg u, \lg v\})(\lg u)(\lg v)).$$

2.2 Refined Complexity Analysis

Assume $u > v > 0$. We want to keep track of the numbers occurring during the execution of the algorithm.

Let $u_0 = u$, $u_1 = v$, and

$$\begin{aligned} u_0 &= a_0 u_1 + u_2 \\ u_1 &= a_1 u_2 + u_3 \\ u_2 &= a_2 u_3 + u_4 \\ &\vdots \\ u_{n-2} &= a_{n-2} u_{n-1} + u_n \\ u_{n-1} &= a_{n-1} u_n. \end{aligned}$$

Obviously, $u_0 > u_1 > u_2 > \cdots > u_{n-1} > u_n$, and we have n division steps. First we bound n as a function of u. Fibonacci numbers (which will be introduced next) gives the worst case time complexity.

Fibonacci Numbers. Fibonacci numbers can be defined by the recursive relation:

$$F_n = F_{n-1} + F_{n-2},$$

with initial values $F_0 = 0$, $F_1 = 1$. It is known that

$$F_n = \frac{\alpha^n - \beta^n}{\sqrt{5}},$$

where $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1-\sqrt{5}}{2}$. Since $|\beta| < 1$, we have $|\beta^n| \longrightarrow 0$ and hence,

$$F_n \sim \frac{\alpha^n}{\sqrt{5}} = \Theta(\alpha^n).$$

Lemma 1 (4.2.1 in [1]) *Given the notation above, we have $u \geq F_{n+2}$ and $v \geq F_{n+1}$.*

We will use this lemma to obtain our refined bit complexity of the Euclidean algorithm next class meeting.

References

[1] E. Bach and J. Shallit, *Algorithmic Number Theory*, The MIT Press, Cambridge, Massachusetts, 1996.