Monotone Circuits for Monotone Weighted Threshold Functions[*]

Amos Beimel[†] Enav Weinreb[*]

Abstract

Weighted threshold functions with positive weights are a natural generalization of unweighted threshold functions. These functions are clearly monotone. However, the naive way of computing them is adding the weights of the satisfied variables and checking if the sum is greater than the threshold; this algorithm is inherently non-monotone since addition is a non-monotone function. In this work we by-pass this addition step and construct a polynomial size logarithmic depth unbounded fan-in monotone circuit for every weighted threshold function, i.e., we show that weighted threshold functions are in $m\mathcal{AC}^1$. (To the best of our knowledge, prior to our work no polynomial monotone circuits were known for weighted threshold functions.)

Our monotone circuits are applicable for the cryptographic tool of *secret sharing schemes*. Using general results for compiling monotone circuits (Yao, 1989) and monotone formulae (Benaloh and Lechter, 1990) into secret sharing schemes, we get secret sharing schemes for every weighted threshold access structure. Specifically, we get: (1) information-theoretic secret sharing schemes where the size of each share is quasi-polynomial in the number of users, and (2) computational secret sharing schemes where the size of each share is polynomial in the number of users.

Keywords: Computational Complexity, Cryptography, Parallel Algorithms, Theory of Computation.

1 Introduction

A weighted threshold function with positive weights is described by a set of positive weights $w_1, \ldots, w_n$ and a threshold T. It computes a monotone Boolean function where $f(x) = 1$ iff $\sum_{i=1}^{n} x_i w_i \geq T$. Weighted threshold functions play an important role in complexity theory (e.g., [14, 6, 7]) and learning theory (e.g., [10, 9]). The unweighted threshold function has a polynomial size monotone formula [1, 19]. Weighted threshold functions are a natural generalization of threshold functions. However, to the best of our knowledge, prior to our work no polynomial monotone circuits were known for weighted threshold functions.

A weighted threshold function with positive weights is clearly a monotone function. However, the naive way of computing it is adding the weights of the satisfied variables and checking if the sum is greater than the threshold. Using *iterated addition* [13, 21], and the fact that the weights and the threshold can always be represented using at most $n \log n$ bits (See Claim 2.2), this algorithm can be implemented by

[*]A preliminary version of this paper appeared in [4].

[†]Dept. of Computer Science, Ben-Gurion University, Beer-Sheva, Israel. E-mail: {beimel,weinrebe}@cs.bgu.ac.il. The work of the second author was partially supported by a Kreitman Foundation Fellowship and by the Frankel Center for Computer Science.

an $\mathcal{NC}^1$ non-monotone circuit. However, this algorithm is inherently non-monotone since addition is a non-monotone function. This demonstrates the usefulness of non-monotone computational steps even for monotone functions. Our task in designing a monotone circuit for weighted threshold functions is to by-pass this addition step. In this work we construct a polynomial size logarithmic depth unbounded fan-in monotone circuit for every weighted threshold function, i.e., we show that weighted threshold functions are in $m\mathcal{AC}^1$.

Our monotone circuits are applicable for the cryptographic tool of *secret sharing schemes*. In a weighted threshold secret sharing scheme, a dealer shares a secret among a set of users, where each user has a positive weight and there is a certain threshold. A set of users can reconstruct the secret from their shares if and only if the sum of their weights is at least the threshold. Using general results for compiling monotone circuits [23] and monotone formulae [5] into secret sharing schemes, we get secret sharing schemes for every weighted threshold access structure. In the information theoretic setting, the size of each share is quasi-polynomial in the number of users. In the computationally secure setting, the share size is polynomial in the number of users. Secret sharing schemes for weighted threshold access structures were previously considered in [17, 3, 11, 15].

Our results complement a recent result of Servedio [16], who shows that every monotone weighted threshold function can be approximated by a polynomial size monotone formula. The question if every monotone weighted threshold function can be exactly computed by a polynomial size monotone formula remains open. The question of constructing monotone formulae for monotone weighted threshold functions is closely related to an open problem of Goldman and Karpinski [7]; they ask if every monotone weighted threshold function has a monotone constant depth polynomial size circuit with unweighted threshold gates (such simulations are known in the non-monotone case [6, 7, 2]). If such a monotone simulation is possible then using the formulae of [1, 19] we get an efficient monotone formulae for monotone weighted threshold functions.

2 Monotone Circuits for Every Weighted Threshold Function

In this section we describe a super-linear depth polynomial size monotone circuit for every weighted threshold function. We first define weighted threshold functions and discuss their basic properties. Then, we overview the monotone circuit construction. Thereafter, we define a universal weighted threshold function, and, finally, we design a super-linear depth polynomial size monotone circuit for the universal function.

Definition 2.1 (Weighted Threshold Function) *Let $w_1, \ldots, w_n \in \mathbb{N}$ be positive weights, and $T \in \mathbb{N}$ be a threshold. Define $f : \{0,1\}^n \rightarrow \{0,1\}$ in the following way: $f(x_1, \ldots, x_n) = 1$ if and only if $\sum_{i=1}^{n} w_i x_i \geq T$. Then f is called the weighted threshold function associated with $w_1, \ldots, w_n$ and T.*

As we deal with monotone circuits, all the functions in this paper are monotone weighted threshold functions, i.e., the weights are always positive. It is easy to see that Definition 2.1 does not restrict generality when assuming that the weights and the threshold are integers. In other words, given a weighted threshold function with real weights and threshold, there exist integer weights and threshold that induce the very same weighted threshold function. Furthermore, by a famous result of [12], the weights can be represented by $\lceil n \log n \rceil$ bits. Moreover, Håstad [8] proved that for some weighted threshold functions, this upper bound is tight. We summarize the upper bound in the next claim, where $\tau \overset{\text{def}}{=} \lceil n \log n \rceil$.

Claim 2.2 ([12]) *Let f be a monotone weighted threshold function with n variables. There exist positive integers $w_1, \dots, w_n$ and T of size smaller than 2^τ, such that f is the function associated with $w_1, \dots, w_n$ and T.*

We next describe the ideas of our super-linear depth polynomial size monotone circuit. By Claim 2.2, we can assume that the threshold and weights are integers that can be represented by binary numbers with at most τ bits, where $\tau \overset{\text{def}}{=} \lceil n \log n \rceil$. We define a universal weighted threshold function on $n\tau$ variables. In the sequence, it suffices to design a monotone circuit for the universal function to get a monotone circuit for every weighted threshold function. In the reduction from an arbitrary weighted threshold function to the universal threshold function, each variable is replaced by a set of at most τ variables, such that the weight associated with every new variable is a power of 2, and their sum equals the weight of the original variable. Hence, we will have τ levels of variables, where in the ith level we will have n variables of weight 2^i. By adding additional variables we change the threshold to be 2^τ. To conclude, in this universal function the weights and the threshold are powers of 2, which makes the design of the circuit easier.

We now consider the variables of each level, and sort them such that the 1's in the level appear before the 0's. If the number of 1's in level i is a_i, then the sum of the n numbers is $\sum_{i=0}^{\tau} a_i 2^i$. As explained, we cannot compute this sum monotonically, and we will only check if this sum is greater than the threshold. We use the fact that the sum of weights of any two variables in a given level equals the weight of one variable in the level above. Thus, we iteratively propagate half of the bits of any level to the level above. In the end, we accept only if the $\tau - 1$ level propagates at least one bit to the τth level. We show that since the threshold is a power of 2, we can handle the slight lose of accuracy caused when the number of bits in a level is odd.

We now define a *universal* weighted threshold function u_n on $n\tau$ variables. That is, $u_n : \{0,1\}^{n\tau} \to \{0,1\}$ is a weighted threshold function defined over the variables $\{x_{i,j}\}_{0 \le i < \tau, 1 \le j \le n}$. The $n\tau$ variables of u_n are partitioned into τ levels, with n variables in each level. The weight assigned to each variable in the ith level is 2^i, and the threshold is 2^τ. The function u_n is universal in the sense that every weighted threshold function on n variables is monotonically reducible to it. Therefore, it suffices to construct a monotone circuit for u_n, in order to get a monotone circuit for every weighted threshold function.

Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone weighted threshold function. We now show how to reduce f to u_{n+1}, using a projective reduction. First, by Claim 2.2, we can assume that the threshold and weights are integers that can be represented by binary numbers with at most τ bits. Second, we associate each variable of f to a column of variables of u_n, according to its binary representation. That is, if x_j is a variable with weight $w = \sum_{i=0}^{\tau-1} w_i 2^i$, we replace it by at most τ new variables where if $w_i = 1$ then we add a variable $x_{i,j}$ with weight 2^i. More formally, if $w_i = 1$ we connect the input variable x_j of f by a wire to the variable $x_{i,j}$ of u_n, and if $w_i = 0$, we connect the constant 0 to $x_{i,j}$. Third, by adding at most τ dummy variables which are always connected to the constant 1, we can change the threshold to be 2^τ. To conclude, we have proved the following claim.

Claim 2.3 *Let $f : \{0,1\}^n \to \{0,1\}$ be a weighted threshold function. If u_{n+1} has a polynomial monotone circuit, then f has a polynomial monotone circuit as well.*

Constructing the Circuits. The first step in every circuit that we construct for the universal weighted threshold function is sorting the inputs at every level. To do this, we use a sorting network which has n Boolean inputs $x_1, \dots, x_n$ and returns n Boolean outputs $z_1, \dots, z_n$ such that $\sum_{i=1}^{n} z_i = \sum_{i=1}^{n} x_i$ and if $i < j$ then $z_i \leq z_j$. That is, the network sorts $x_1, \dots, x_n$. There are a few possible monotone implementations for a sorting network using Boolean AND/OR gates, e.g., Ajtai, Koml´os, and Szemerédi (AKS) sorting network [1] of depth $O(\log n)$ or Valiant's construction of monotone formula for majority [19]. From now on we will assume that the input variables at each of the τ input levels of u_n are sorted.

We now describe two tools we use to construct our circuit. Our first tool is a simple monotone circuit which we call a *halving circuit*. It has m inputs $x_1, \dots, x_m$ and $\lfloor m/2 \rfloor$ outputs $y_1, \dots, y_{\lfloor m/2 \rfloor}$ such that if the inputs are sorted then the output satisfies $\sum_{i=1}^{\lfloor m/2 \rfloor} y_i = \left\lfloor \frac{\sum_{i=1}^{m} x_i}{2} \right\rfloor$. The fact that we first sort the input variables of each level enables us to use the halving circuit. The implementation of the halving circuit is trivial and does not contain a single gate. It simply takes every second input x_{2i} and wires it to y_i. A property of the halving circuit is that the output is sorted as well. Our second tool is a monotone circuit called a *merge circuit*. It has $2n$ inputs $x_1, \dots, x_n, y_1, \dots, y_n$ and $2n$ outputs $z_1, \dots, z_{2n}$, such that if the inputs $x_1, \dots, x_n$ are sorted and the inputs $y_1, \dots, y_n$ are sorted, then $\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} y_i = \sum_{i=1}^{2n} z_i$ and the outputs are sorted. To compute z_ℓ, for $1 \leq \ell \leq n$, we simply implement the following OR of ANDs: $z_\ell = \bigvee_{i+j=\ell} x_i \wedge y_j$. Thus, the merge circuit is implemented by a monotone circuit of depth 2.

We are ready to construct the polynomial-size circuit using the halving and the merge circuits. Each level will propagate a carry to the level above it, determined by the number of satisfied variables in the level, and by the carry it got from the level below. The sum of weights of every two variables in the ith level equals the weight of one variable in the $i+1$ level. Thus, each level sums the number of its satisfied variables and the carry it got from the level below, and then propagate half of it as carry. If the number to propagate is not even, the value of the carry will be slightly smaller then the actual weight it represents. However, since the threshold is a power of 2, we will manage with this certain loss of accuracy.

We denote by a_i, for $0 \leq i < \tau$, the number of satisfied variables in the ith level, that is, $a_i \stackrel{\text{def}}{=} \sum_{j=1}^{n} x_{i,j}$. We denote by c_i the carry the ith level propagates to the $i+1$ level. To propagate the carry from the ith level, we first use the merge circuit on the level's variables and on the carry wires from the level below, to get the data of the level sorted. The halving circuit is now used in order to produce the carry to the level above. The number of bits propagated are $c_i \stackrel{\text{def}}{=} \left\lfloor \frac{a_i + c_{i-1}}{2} \right\rfloor$. We define $c_{-1} \stackrel{\text{def}}{=} 0$ because the bottom level gets no carry. Finally, we accept if $c_{\tau-1} > 0$. The circuit is described in Figure 1.

We now prove that the circuit correctly computes u_n. For every i, where $0 \leq i < \tau$, define v_i to be the total weight of the satisfied variables in levels 0 to i, that is, $v_i \stackrel{\text{def}}{=} \sum_{j=0}^{i} a_j 2^j$. Clearly, it holds that $v_i = v_{i-1} + 2^i a_i$, where $v_{-1} \stackrel{\text{def}}{=} 0$. Using this notation, we have to show that the circuit accepts if $v_{\tau-1} \geq 2^\tau$ and rejects otherwise. The following claim relates the value v_i to the carry c_i propagated by the ith level.

Claim 2.4 *For every i, where $-1 \leq i < \tau$, it holds that $v_i - 2^{i+1} < 2^{i+1} c_i \leq v_i$.*

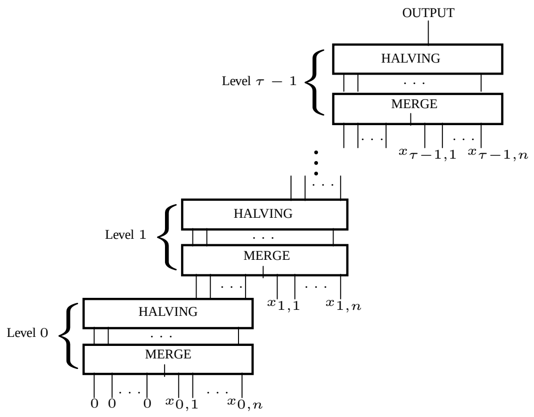


Figure 1: Polynomial monotone circuit for the universal weighted threshold. The sorting stage is omitted.

Proof: The proof is by induction. By inspection, the claim is correct for $i = -1$. We now turn to the induction step. We first show that the second inequality holds:

$$2^{i+1}c_i = 2^{i+1}\left\lfloor \frac{a_i + c_{i-1}}{2} \right\rfloor \leq 2^{i+1}\frac{a_i + c_{i-1}}{2} = 2^i a_i + 2^i c_{i-1} \leq 2^i a_i + v_{i-1} = v_i,$$

where the last inequality is by the induction hypothesis and the last equality is by the definition of v_i. We now turn to the first inequality:

$$2^{i+1}c_i = 2^{i+1}\left\lfloor \frac{a_i + c_{i-1}}{2} \right\rfloor \geq 2^{i+1}\frac{a_i + c_{i-1} - 1}{2} = 2^i a_i + 2^i c_{i-1} - 2^i > 2^i a_i + (v_{i-1} - 2^i) - 2^i = v_i - 2^{i+1},$$

where the last inequality is by the induction hypothesis and the last equality is by the definition of v_i. □

We now prove the correctness of the circuit. If $v_{\tau-1} \geq 2^\tau$, then, by the first inequality of Lemma 2.4, the value of $c_{\tau-1}$ must be greater than 0, and thus the circuit accepts. On the other hand, if $v_{\tau-1} < 2^\tau$, then, by the second inequality of Lemma 2.4, the value of $c_{\tau-1}$ must be 0, causing the circuit to reject. To conclude, as we have $\tau \overset{\text{def}}{=} \lceil n \log n \rceil$ sorting networks and τ levels of computation, each of polynomial size, the size of the entire circuit is polynomial in n. We summarize the construction in the next theorem.

Theorem 2.5 *There exists a constant c such that every monotone weighted threshold function with n inputs has a monotone circuit of size n^c. Furthermore, given the weights and the threshold this circuit can be constructed efficiently.*

3 Shallow Monotone Circuits for Weighted Threshold Functions

In this section we transform the super-linear depth monotone circuit constructed in Section 2 into a logarithmic depth circuit. In Section 3.1, we use balancing to get a logarithmic depth quasi-polynomial size circuit. In Section 3.2, we use dynamic programming to construct the logarithmic-depth polynomial-size circuit.

3.1 Balancing the Circuit

We next apply balancing ideas to reduce the depth of the circuit. Note that the circuit described above has τ levels, each one is of constant depth, and a preprocessing sorting layer of depth $O(\log n)$. The improvement follows the fact that the output of the halving circuit is sorted, thus there are only $n+1$ possibilities for the carry that level i gets from level $i-1$ (i.e., the value of c_{i-1}). Therefore, the circuit can compute, in parallel, the value of c_i for each of the $n+1$ possible values of c_{i-1}. In parallel, it computes the real value of c_{i-1} and accordingly chooses which of the $n+1$ values of c_i to output. The idea of the construction is similar to formulae balancing (see [18] for non-monotone formulae balancing and [22] for monotone formulae).

Formally, our construction is recursive. We denote by $C_{i,j}$ a component whose inputs are the carry c_{i-1} and the inputs to levels i to j. The component produces the corresponding output of the jth level, namely c_j. The basis of this recursion is a component where $i = j$, that is, computing c_i given c_{i-1} and the input variables of the ith level. This is done simply by connecting a halving circuits to the output of a merge circuit, and thus can be done in depth 2.

We now show how to construct $C_{i,j}$ for some $0 \leq i < j < \tau$, from the components $C_{i,k}$ and $C_{k+1,j}$, where $k = \lfloor (i+j)/2 \rfloor$. The inputs of $C_{i,j}$ are c_{i-1} and the input variables of levels i to j, and the output is c_j. We use $n+1$ copies of $C_{k+1,j}$ to be evaluated in parallel. For every $0 \leq \ell \leq n$, the copy $C^{(\ell)}_{k+1,j}$ computes the output of the jth level in the original circuit, assuming that $c_k = \ell$, that is, the carry to the $k+1$ level equals ℓ. In parallel, we compute the circuit $C_{i,k}$ with c_{i-1} as input, to produce the correct value of c_k. See Figure 2 (the description of the selector circuit is explained in the sequel).

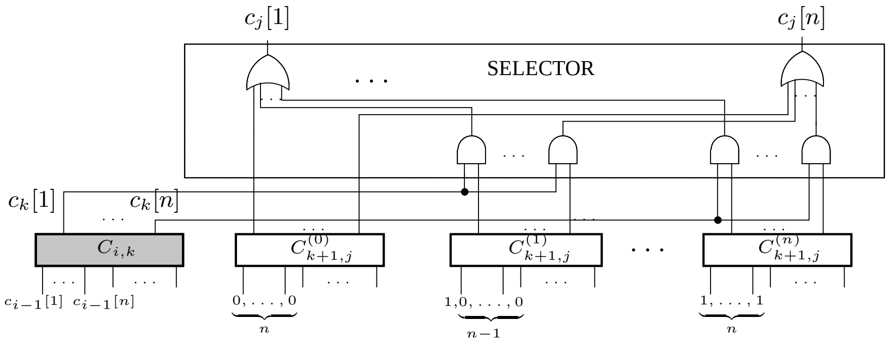
To choose the correct output of $C_{i,j}$, we use a monotone circuit we call a *selector circuit*. Consider the output of a component $C^{(\ell)}_{k+1,j}$, for some $0 \leq \ell \leq n$. If $\ell = c_k$, the output of the component is exactly the correct value of c_j, which we want $C_{i,j}$ to output. However, if $\ell > c_k$, then the output may be larger than the correct value of c_j. On the other hand, the output of $C_{i,k}$ is a unary representation of c_k. That is, for every $1 \leq e \leq n$, the eth bit of the output is 1 if and only if $c_k \geq e$. Thus, in the first layer of the selector, for every $0 < \ell \leq n$, we AND all the output bits of $C^{(\ell)}_{k+1,j}$ with the ℓth bit of the output of $C_{i,k}$. For every $0 < \ell \leq n$, denote the n output bits of these AND gates as $B^{(\ell)}$. That is, $B^{(\ell)}[e] = c_k[\ell] \wedge C^{(\ell)}_{k+1,j}[e]$, for every $1 \leq e \leq n$. As the output of $C^{(0)}_{k+1,j}$ is never bigger than c_j, we define $B^{(0)}[e] = C^{(\ell)}_{k+1,j}[e]$.

Since the computation of $C_{k+1,i}$ is monotone, the eth bit in $B^{(\ell-1)}$ is smaller or equal to the eth bit of $B^{(\ell)}$, for every $0 \leq \ell \leq c_k$. Hence, we compute $c_i[e] = \bigvee_{j=0}^{n} B^{(j)}[e]$. If the correct value of $c_i[e]$ is 1, then the value of $B^{(c_k)}[e]$ is 1, and thus we output the correct value. On the other hand, if the correct value of $c_i[e]$ is 0, then the value of $B^{(c_k)}[e]$ is 0, and thus the value of $B^{(\ell)}[e]$ is 0, for every $0 \leq \ell < c_k$. On the other hand, if $\ell > c_k$ all the bits of $B^{(\ell)}$ are 0, and thus so is $B^{(\ell)}[e]$. Therefore, the value of the OR gate is 0 in this case. Hence, we are able to compute $C_{i,j}$ using $C_{i,k}$ and $C_{k+1,j}$ with a depth 2 monotone circuit. The description of the selector circuit and the recursive step is described in Figure 2.

To compute the function u_n we construct $C_{0,\tau-1}$, hard-wiring 0 as c_{-1}. This will take $O(\log \tau) = O(\log n)$ steps, and thus we get a monotone circuit of depth $O(\log n)$ for u_n, and in general for every monotone weighted threshold function. However, the size of the circuit is $n^{O(\log n)}$, which is quasi-polynomial.

3.2 Constructing $m\mathcal{AC}^1$ Circuits for Weighted Threshold Functions

In this section we use dynamic programming to reduce the size of the monotone circuit constructed in Section 3.1 to polynomial while maintaining logarithmic depth. The waste in size in the circuit from Section 3.1 can be demonstrated looking at the recursive step. Consider the $n+1$ circuits $C^{(0)}_{k+1,j}, \ldots, C^{(n)}_{k+1,j}$. Although



$c_j[1]$
$c_j[n]$
SELECTOR
$c_k[1]$
$c_k[n]$
$C_{i,k}$
$C^{(0)}_{k+1,j}$
$C^{(1)}_{k+1,j}$
$C^{(n)}_{k+1,j}$
$c_{i-1}[1]$ $c_{i-1}[n]$
$\underbrace{0,\ldots,0}_{n}$
$\underbrace{1,0,\ldots,0}_{n-1}$
$\underbrace{1,\ldots,1}_{n}$

Figure 2: The construction of $C_{i,j}$ from $C_{i,k}$ and $C_{k+1,j}$ using a selector circuit.

they all assume different values of c_k, in the next recursive step, each of them will use an identical set of $n+1$ copies of $C_{k',j}$ for $k' = \lfloor (k+1+j)/2 \rfloor$.

We show that by correct wiring of the circuit, we can save in the size of the circuit and reduce it to be polynomial. Let $0 \le i, j \le \tau$ be two indices of input rows. Next, we design a monotone circuit $D_{i,j}$ that gets the input variables of levels i to j as inputs, and outputs $n+1$ unary strings of length n. For every $0 \le \ell \le n$, the ℓth output string, denoted $c_j^{(\ell)}$, will be a unary representation of c_j, assuming that the value of c_{i-1}, the carry propagated to the ith level, is equal to ℓ.

We show how to construct $D_{i,j}$, recursively, for every $0 \le i, j < \tau$. If $i = j$, to compute the output $c_j^{(\ell)}$ for every $0 \le \ell \le n$, we simply merge the variables of the ith level with ℓ constant 1's, and use the halving circuit on the result. If $i < j$, let $k = \lfloor (i+j)/2 \rfloor$, as in Section 3.1, and assume we constructed $D_{i,k}$ and $D_{k+1,j}$. We need to compute $c_j^{(\ell)}$ for every $1 \le \ell \le n$, which is the output of the jth level in the original circuit, assuming the carry propagated to the ith level was $c_{i-1} = \ell$. Thus, for computing $c_j^{(\ell)}$, we will only use the ℓth output of $D_{i,k}$, denoted $c_k^{(\ell)}$, that also assumes $c_{i-1} = \ell$. We now need to combine the value from $D_{i,k}$ with the computation of $D_{k+1,j}$. However, this can simply be done by using the selector circuit described in Section 3.1. An illustration of the recursive step appears in Figure 3. Therefore, we are able to compute the values of $c_j^{(\ell)}$ for every $0 \le \ell \le n$ using $D_{i,k}$, $D_{k+1,j}$, and a depth 2 monotone circuit.

To compute the universal weighted threshold function u_n, we will construct the circuit $D_{0,\tau-1}$, and will check if the output $c_{\tau-1}^{(0)}$ is bigger than 0. Therefore, the depth of the resulted circuit $D_{0,\tau-1}$ is $O(\log \tau) = O(\log n)$. The size of the circuit remains polynomial, since we use exactly one copy of $D_{i,k}$ and one copy of $D_{k+1,j}$ in the recursive step. We have τ sorting circuits, each of polynomial size, and $O(\tau)$ components implementing the recursive step, each of polynomial size, and thus the size of the circuit is polynomial in n.

Theorem 3.1 *Every weighted threshold function is in $m\mathcal{AC}^1$, that is, it has a logarithmic depth polynomial size unbounded fan-in monotone circuit.*[1]

[1]In fact, since we use only fan-in 2 AND gates, our circuit is a $m\mathcal{SAC}^1$ circuit.

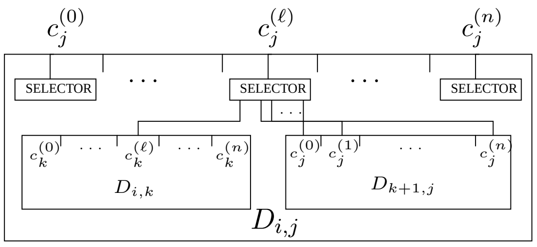


Figure 3: The construction of $D_{i,j}$ from $D_{i,k}$ and $D_{k+1,j}$ using dynamic programming.

4 Secret Sharing for Weighted Threshold Access Structures

A secret sharing scheme involves a dealer who has a secret, a set of n users, and a collection Γ of subsets of the users called the access structure. A secret-sharing scheme for Γ is a method by which the dealer distributes shares to the users such that any subset in Γ can reconstruct the secret from its shares, and any subset not in Γ cannot reveal any partial information about the secret from their shares.

In this work we construct both information-theoretic and computational schemes for weighted threshold access structures. In a *weighted threshold access structure* each user is assigned a positive weight. A set of users can reconstruct the secret only if the sum of its weights is at least a given threshold. Weighted threshold access structures are natural for implementing policies in hierarchical organizations. Shamir [17], in his work defining threshold secret sharing, also considered weighted threshold access structures and proposed a simple secret sharing scheme for such access structures. In Shamir's scheme the size of the share of each user is its weight. Therefore, if the weights are big then the scheme is highly inefficient, i.e., the size of the shares can be exponential in the number of users n. Moreover, prior to this work no efficient secret-sharing schemes for weighted threshold access structures were known.

We use our monotone circuits, together with known compilers, to construct secret sharing schemes for weighted threshold access structures. First, we show an information-theoretic secret sharing scheme with quasi-polynomial size shares. This is done using the scheme Benaloh and Leichter [5] realizing the access structures which have small monotone formulae.

Theorem 4.1 ([5]) *Let Γ be an access structure. If the characteristic function of Γ has a monotone formula of size s, then there is an information-theoretic secret sharing scheme realizing Γ with shares of size s.*

As any unbounded fan-in monotone circuit of depth $O(\log n)$ can trivially be converted to an $n^{O(\log n)}$ size monotone formula, using Theorem 3.1 and Theorem 4.1, we get

Theorem 4.2 *There exists a constant c such that every weighted threshold access structure with n inputs has an information-theoretic secret sharing scheme with shares of size $n^{c \log n}$.*

Second, we show that there is a computational secret-sharing scheme realizing all weighted threshold access structures with polynomial-size shares. This is done using a computational scheme of Yao [23] realizing the access structures which have polynomial-size monotone circuits. This scheme was never published; a description of the scheme has recently appeared in [20].

Theorem 4.3 ([23]) *Let $\langle \Gamma_i \rangle$ be a sequence of access structures, and $\langle f_i \rangle$ be their characteristic functions. If one-way functions exist and there exists a sequence of polynomial-size monotone circuits computing $\langle f_i \rangle$, then there is a computational secret-sharing scheme realizing $\langle \Gamma_i \rangle$.*

Theorem 4.4 *If one-way functions exist, then there is a computational secret-sharing scheme realizing all the weighted threshold access structures.*

Acknowledgments. We thank Yinnon Haviv for his comments on an earlier draft of this paper.

References

[1] M. Ajtai, J. Koml´os, and E. Szemerédi. An $O(n \log n)$ sorting network. In *15th STOC*, pages 1–9, 1983.

[2] K. Amano and A. Maruoka. Better simulation of exponential threshold weights by polynomial weights. Technical Report TR04-090, Electronic Colloquium on Computational Complexity, 2005.

[3] A. Beimel, T. Tassa, and E. Weinreb. Characterizing ideal weighted threshold secret sharing. In *Proc. of the second Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 600–619, 2005.

[4] A. Beimel and E. Weinreb. Monotone circuits for weighted threshold functions. In *Proc. of 20th Annu. IEEE Conf. on Computational Complexity*, pages 67-75, 2005.

[5] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO '88*, volume 403 of *LNCS*, pages 27–35, 1990.

[6] M. Goldmann, J. Håstad, and A. A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

[7] M. Goldmann and M. Karpinski. Simulating threshold circuits by majority circuits. *SIAM J. on Comp.*, 27:230–246, 1998.

[8] J. Håstad. On the size of weights for threshold gates. *SIAM J. on Discrete Mathematics*, 7(3):484–492, 1994.

[9] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[10] M. L. Minsky and S. A. Papert. *Perceptrons*. MIT Press, 1968. Expanded edition 1988.

[11] P. Morillo, C. Padró, G. Sáez, and J. L. Villar. Weighted threshold secret sharing schemes. *IPL*, 70(5):211–216, 1999.

[12] S. Muroga. *Threshold Logic and Its Applications*. Wiley-Interscience, 1971.

[13] Y. Ofman. On the algorithmic complexity of discrete functions. *Sov. Phys. Dokl.*, 7:589–591, 1963. English translation.

[14] I. Parberry and G. Schnitger. Parallel computation with threshold functions. *JCSS*, 36(3):278–302, 1988.

[15] G. Sáez. Some results on the dual of weighted threshold access structures. Manuscript, 2004.

[16] R. Servedio. Monotone Boolean formulas can approximate monotone linear threshold functions. *Discrete Applied Mathematics*, 142(1-3):181–187, 2004.

[17] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[18] P. M. Spira. On time hardware tradeoffs for Boolean functions. In *4th Hawaii Symp. on System Sci.*, pages 525–527, 1971.

[19] L. G. Valiant. Short monotone formulae for the majority function. *J. of Algorithms*, 5(3):363–366, 1984.

[20] V. Vinod, A. Narayanan, K. Srinathan, C. Pandu Rangan, and K. Kim. On the power of computational secret sharing. In *Indocrypt 2003*, volume 2904 of *LNCS*, pages 162–176, 2003.

[21] C. S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. on Computers*, 13:14–17, 1964.

[22] I. Wegener. Relating monotone formula size and monotone depth of Boolean functions. *IPL*, 16(1):41–42, 1983.

[23] A. C. Yao. Unpublished manuscript, 1989. Presented at Oberwolfach and DIMACS workshops.