1 Polynomial bounds for circuit complexity

It is a well-known result (citation needed here) from classical circuit complexity that any symmetric Boolean function that depends on polynomially-weighted inputs can be enacted in constant depth and polynomial sizein a circuit that allows threshold gates. This is surprising, since there are exponentially many rows in a Boolean function. The method from Siu and Bruck declares the existence of intervals within $[-N, N]$, where N is the sum of the magnitudes of all the weights, and represents the entire range of the weighted sum, where 0 is the threshold over the whole circuit.

$$N = \sum_{i=0}^{n} |w_i| \tag{1}$$

That is, the final threshold gate in the circuit is the sign function of the weighted sum of the inputs.

$$f(X) = \operatorname{sgn}\left(w_0 + \sum_{i=0}^{n} w_i x_i\right) \tag{2}$$

In their proof, Siu and Bruck find that at least two layers of threshold gates are needed to enact any symmetric boolean function. In the first layer, they calculate y_k's which are the sum of the differences of each weighted input $w_i x_i$ with one of ℓ intervals $[k_j, \tilde{k}_j]$. If the sum of the weighted inputs are within a particular interval, then that particular y_k will be 1, otherwise, all the y_k's will be zero. The second layer sums up the y_k's. If at least one of them is 1 then their sum is 1, and the circuit outputs 1. Otherwise, none of the y_k's are one, and the circuit outputs 0. Thus, the trick then is to find the intervals $[k_j, \tilde{k}_j]$ that correspond to a given symmetric Boolean function and to guarantee that the number of intervals ℓ is polynomially bounded by n, i.e. $\ell \leq n^c$ for some $c > 0$.

The existence of such an ℓ derives from the theory of $GF(q)$. However, this is non-constructive. Explicit constructions for the addition gate ADD and the comparison gate $COMP$ are given in [citation needed], using the majority gate MAJ, which is just a special case for of the threshold gate where the weights are either $+1$ or -1. Equivalent results can be found for the canonical majority gate where all the weights are +!.

The ADD gates depend on expressing each of the $2\log_2 n$ bits of the block-sum s as a threshold function on a polynomial number of inputs. In the given construction, $n \times n$-bit numbers are summed together, and each bit s_i of the sum, for $0 \leq i < n$, depends not only on the n bits of significance 2^i but also on any carry bits generated and propagated for $0 \leq j < i$. Each bit of the sum depends not only on the n

The original construction says it suffices to show the final high-order carry bit c_n, which potentially depends on the most number of inputs, that is, all $n \log_2 n$ bits of the entire block. However, in this thesis, we give a general

formula for each individual sum and carry bit. The odd and even blocks are summed in parallel, with alternating blocks set to zero.

Therefore, the existence of these polynomials provides an upper bound on the number of terms in the weighted sum, and therefore on the number of intervals in the threshold gate. This is in contrast to the polynomial method used to lower bound quantum query complexity, as in Beals et al. and Ambainis.

The construction depends on *delta polynomials*, which in analogy to the Dirac delta function, has much of its weight on a particular input, namely, all $+1$'s or all -1's, and a much smaller weight on all other inputs. Furthermore, we are only interested in polynomials with a polynomially bounded number of monomials, We say that a polynomial $P(x_n, x_{n-1}, \ldots, x_1, x_0)$ is a (c, ϵ) sparse delta-polynomial if for a particular $\epsilon = \{\epsilon_n, \ldots, \epsilon_0\}$

$$P(\epsilon) \geq a \tag{3}$$

and for all other input vectors ϵ'

$$P(\epsilon') < a/c \tag{4}$$

In particular, the input ϵ signifies the condition where we want the threshold gate to trigger, that is, we want its output to be 1. We can alter the weights w_i so that this triggering happens for the right input vector ϵ, in a sense, "programming" the polynomial.

However, this suffices just for the addition function. In general, we would like to solve the harder problem of multiplication, and finally, of greatest interest to factoring, modular exponentiation, or powering, or one n-bit number by another. Therefore, finding the explicit construction may rely on similar techniques of calculating the character of a finite field.