In this talk, Eric Blais presented joint work with Johan Håstad, Rocco Servedio and Li-Yang Tan, concerned with Boolean functions. More precisely, "the simplest representations of functions": DNF (Disjunctive Normal Form) formulas.

For a little bit of background, recall that a Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ defined on the hypercube [2] is a DNF if it can be written as

$$f(x) = \bigvee_k \bigwedge_{i \in S_k} \ell_i$$

that is as an OR of AND's of literals. One can also see such functions as being eactly those taking value 1 on an "union of subcubes" (if One prefers. I will not argue with One).

A nice property of DNF formulas is that they are arguably amongst the simplest of all representations of Boolean functions; while formulas of depth ≥ 3 are a nightmare, DNFs have been extensively studied, and by now "Everything is known about them". Well, *almost* everything.

Indeed, amidst other facts, we have that

Theorem 1 (Folklore)**.** Every Boolean function can be computed by a DNF of size 2^{n-1}.

Theorem 2 (Lupanov '61)**.** This is tight (PARITY_n needs that much).

Theorem 3 (Korshunov '81, Kuznetsov '83)**.** A random Boolean function can be computed by DNFs of size $\Theta(2^n/(\log n \log\log n))$ (and requires that much).

So... are we done yet? The mere presence of Eric in the auditorium was a clear hint that all was not settled. And as it turns out, if the picture is well understood for *exact* computation of Boolean functions by DNFs, what about *approximate* representation of a function? That is, what about the size required to approximate a Boolean function by a DNF, if one allows error ε (as a fraction of the inputs).

This leads to the notion of **DNF approximator complexity**; and here again some results – much more recent results:

Theorem 4 (Blais–Tan '13)**.** Every Boolean function can be approximated by a DNF of size $O(2^n/\log n)$. Furthermore, our old friend PARITY_n only needs DNF size $O(2^{(1-2\varepsilon)n})$.

That's *way* better than 2^{n-1}. So, again – are we done here? And, again... not quite. This brings us to the main point of the paper, namely: what about *monotone* functions? Can they be computed more efficiently? Approximated more efficiently? (Recall that a Boolean function f is monotone if $x \preceq y$ implies $f(x) \leq f(y)$, where $\preceq$ is the coordinate-wise partial order on bit-strings.)
As a starter: no.

Theorem 5 (Folklore)**.** Every monotone Boolean function can be computed by a DNF of size $O(2^n/\sqrt{n})$ (using subcubes rooted on each min-term); and again, this is tight – for the majority function MAJ_n.

Furthermore, and quite intuitively, using negations does not buy you anything to compute a monotone function (and why should it, indeed?):

Theorem 6 (Quine '54)**.** To compute monotone Boolean functions, monotone DNFs are the best amongst DNFs.

Not surprising, I suppose? Well... it's a whole new game when one (one, again!) asks only for approximations; and that's the gist of the paper presented here. First of all, drastic savings in the size of the formulas!

Theorem 7 (Blais–Hastad–Servedio–Tan '14)**.** For $\varepsilon = 0.1$ (or any fixed constant), every monotone Boolean function can be ε-approximated by a DNF of size $O(2^n/2^{\Omega(\sqrt{n})})$.

Eric gave a high-level view of the proof: again, it works by considering the subcubes rooted on each min-term, but in two steps:

- Regularity lemma: the world would be much simpler if all subcubes were rooted on the same level of the hypercube; so first, reduce it to this case (writing $f = f_1 + \cdots + f_k$, each f_i has this property)
- then, approximate independently each f_i, using a probabilistic argument (via random sampling), to prove there exists a good approximator for all f_i's, and then stitching them together.

And they also show it is tight: this time with the majority function MAJ_n. The proof goes by a counting argument and concentration of measure on the hypercube (every or almost every input is on the middle "belt" of the hypercube; but each subcube thus has to be rooted there, and each cannot cover too much... so many are needed)

So, approximation *does* buy us a lot. But clearly, using negations shouldn't, should it? Why would allowing non-monotone DNF's to approximate monotone functions *ever* help? (**Hint:** it does.) (Yep.)

Theorem 8 (Blais–Hastad–Servedio–Tan '14)**.** For every n, there exists ε_n and $f\colon \{0,1\}^{6n} \to \{0,1\}$ such that

- f can be ε_n-approximated by DNFs of size $O(n)$;
- any *monotone* DNF ε_n-approximating f must have size $\Omega(n^2)$.

(Take that, intuition!)

The upshot: *exact* computation and *approximate* computation have intrinsically **very** different properties!

Eric then concluded with an open question: namely, how to improve/better understand the gap between approximating functions with monotone DNF vs. approximating them with general DNF's (the currently known gap in the size being quite huge – almost exponential). Additionally, how to get a separation as in the mind-boggling theorem above, but changing the quantifiers – that is, for a constant ε independent of n?

Also, can someone fix my intuition? I think it's broken.

[1] `http://www.cs.columbia.edu/~rocco/papers/icalp14.html`

[2] Not this one. `http://en.wikipedia.org/wiki/Cube_2:_Hypercube`