A view of complexity theory, with a "concrete" open problem

Kenneth W. Regan

July 11, 2007

Computational complexity theory is the study of information flow and the *effort* required for it to reach desired conclusions. Computational models like cellular automata, Boolean or algebraic circuits, and other kinds of fixed networks exemplify this well, since they do not have "moving parts" like Turing machine tape heads, so the flow's locations are fixed. Measures of effort include the time for the flow, the amount of space or hardware needed, and subtler considerations such as time/space to prepare the network, or energy to overcome possible dissipation during its operation. These models and measures have fairly tight relations to Turing machines and their familiar complexity measures.

For an example and open problem, consider the general task of moving all "garbage bits" to the end of a string, leaving the "good bits" in their original sequence. We can model this as computing the function $f : \{ 0, 1, 2 \}^* \longrightarrow \{ 0, 1, 2 \}^*$ exemplified by $f(1020212) = 1001222$, $f(2200) = 0022$, $f(101) = 101$, etc., with $0, 1$ as "good bits" and 2 as "garbage." A rigorous inductive definition, using e for the empty string, is $f(e) = e$, $f(0x) = 0f(x)$, $f(1x) = 1f(x)$, and $f(2x) = f(x)2$. This is the "topological sort" of the partial order $B = \{ 0 < 2, 1 < 2 \}$ that is *stable*, meaning that subsequences of incomparable elements are preserved. The problem is, can we design circuits C_n, each computing $f(x)$ on strings x of length n, that have size $O(n)$?

The circuits C_n have *input gates* labeled $x_1, \ldots, x_n$ which receive the corresponding "trits" (0, 1, or 2) of the input string x, and *output gates* $y_1, \ldots, y_n$ giving $y = f(x)$. The first question is, what interior computational gates can C_n have? A *comparator gate* g for a partial order $(P, <)$ has two input and two output wires, maps (a, b) either to (a, b) or (b, a), and never maps to (d, c) when $c < d$. The unique *stable comparator* g_P maps (a, b) to (a, b) unless $b < a$. The following slightly extends the famous 0-1 law for comparator networks:

Theorem 1. If a circuit C_n of comparator gates computes $f(x)$ correctly for all $x \in \{ 0, 2 \}^n$ (not even including any 1s), then for every partial order $(P, <)$, the circuit C_P with each comparator replaced by g_P computes the stable topological sort of P.

Proof. First suppose C_P errs for a total order $(P, <)$. Then there are $x, y \in P^n$ such that $C_P(x) = y$, but for some j, $y_j + 1 < y_j$. Take the permutation p such that $x_i = y_{p(i)}$ for all indices i. Define a binary string $y' \in \{ 0, 2 \}^*$ by $y'_i = 0$ if $y_i < y_j$, $y'_i = 2$ otherwise, and x' by $x'_i = y'_{p(i)}$ for all i. Then $C_n(x') = y'$ (exercise: prove this by induction taking gates one at a time), contradicting that the original C_n was correct on $\{ 0, 2 \}^*$.

For $(P, <)$ not a total order, an error $C_P(x) = y$ (which might violate only stability) is also an error in the total order $(P_x, <')$ with $P_x = \{ (a, i) : x_i = a \}$ and $(a, i) <' (b, j)$ if $a < b$ or a is not comparable to b and $i < j$. [ ]

Corollary 2. Circuits C_n of comparator gates computing f require size $n \log_2(n) - O(n)$. [ ]

This follows by applying the standard sorting lower bound to C_P. It's interesting that we did not need 1s in x to argue stability, and the lower bound allows gates g in C_n to be arbitrary when either input is 1.

For general circuits, however, the argument doesn't hold, and all bets are off! To see why, consider sorting the total order $\{ 0 < 1 < 2 \}$. Clever $O(n)$-size circuits can *count* the numbers a, b, c of 0s, 1s, and 2s in the input string x, respectively, and then assemble the correct output $y = 0^a 1^b 2^c$. For the basic idea see [Muller-Preparata, JACM 1975], and various sources on the "Dutch National Flag Problem." Applying this counting idea to our poset B reduces our task to "nice" strings z of length $N = 2^k$ with exactly $N/2$ 2s.

Theorem 3. If $s(N)$-size circuits D_N can compute $f(z)$ for "nice" z, then f has circuits of size at most $s(4n) + O(n)$.

Proof. We can build $O(n)$-size circuits E_n that on inputs x of length n count b, c as above and find k such that $m = 2^{k-1}$ is the least power of 2 above n. Make $E_n(x)$ output $z = x1^{m+c-n}2^{m-c}$, which gives $|z| = N < 4n$. Then compute $y' = D_N(z)$ and re-use the computed b, c, m to pluck off the n bits of $f(x)$. [ ]

This reduction to nice z enhances the "flow" metaphor. The m-many 2s in z can be advance-routed to the last m places of y', so the whole issue is how the m-many 0s and 1s in z flow together into the first m places of y'. *Must* this flow progress (without loss of circuit-size generality) by "squeezing out 2s" in an intuitively plane-filling fashion, allowing "mileposts" whose forced spacing might mandate having $n \log_2(n) - O(n)$ gates? Or can linear-size networks rise above the planar view? No one I've asked has known, and lack of them frustrates a desired general linear-size circuit simulation of my "Block Move" model. Issues of "self-routing superconcentrators" (see N. Pippenger, STOC'93) may be involved. Nor do I know nicer descriptions of $O(nlogn)$-sized circuits than "use ancillas to tag bits of x and work in P_x as in the proof of Theorem 1, employing ideas of Theorem 3 and/or mapping into the $O(nlogn)$-sized Ajtai-Komlos-Szemeredi networks" (STOC'83). Those seeking an $o(nlogn)$ upper bound may be my guest, but those believing a super-linear circuit lower bound must reflect that no such bounds are known for string functions whose graphs belong to NP or to E.

The above inductive definition of f yields a linear-time algorithm on any model that simulates each operation of a double-ended queue in $O(1)$ time. But is booting a 2 to the rear in $f(2x) = f(x)2$ really in constant time, even amortized? True, our technical issues shrink away on passing from linear to polynomial time, so all this may seem to have nothing to do with P versus NP. But *au-contraire* the Baker-Gill-Solovay "oracle" obstacle may mean nothing more than that standard "diag-sim" and timing techniques are insensitive to internal information flow. The "Natural Proofs" obstacle *may* ultimately say only that network-preparation/"nonuniformity" is a subtly powerful consideration. Honing tools for information-flow analysis on incrementally more-general cases that yield super-linear lower bounds may be the walk to walk before trying to run.