From *SIAM News*, Volume 37, Number 3, April 2004

Sublinear Computing: When Ignorance Is Bliss

By Barry A. Cipra

He had long ago concluded that he possessed only one small and finite brain, and he had fixed a habit of determining most carefully with what he would fill it.

—Annie Dillard, *The Living*

One of the well-known rules for being an organized, efficient officeworker is to handle a piece of paper once and only once: read, act, toss. Letting paperwork pile up while you ponder and procrastinate is a timeworn recipe for disarray and further delay. The computer revolution's transformation of physical into virtual clutter only compounds the problems of the information pack rat, making it easy to amass millions of disorganized files instead of mere thousands.

Even the capacious hard drives of computers are soon swamped by the relentless onslaught of information if they try to keep records of everything. Take network management, for example. Routers on the Internet—the computers that figure out how that Web page you requested makes its way from a server in Serbia to your laptop in Los Angeles—have access to all the data they could possibly want, to determine how the network is being used (and, presumably, how to improve it—or bill for the usage): IP origin/destination numbers, packet sizes, timestamps, and so forth. But routers can't afford to maintain terabytes of data for off-line processing—and by the time we have routers that can, levels of traffic on the Internet will most likely make terabytes seem tiny.

Remote digital sensing is another case in which the ability to gather information soon outstrips the resources available to store it. Just think of all the correlations potentially lurking in the air-quality data continuously collected—and mostly overwritten—by pollution-control agencies. Or picture the discoveries implicit in the photons pouring into telescopes (and surveillance cameras). In a growingly bit-rich digital environment, the ability to deal decisively with streaming data is imperative. As Abraham Lincoln might have put it, you can store all the data some of the time and some of the data all the time, but you can't store all the data all the time.

Dealing with the data glut has been the subject of invited addresses at the last two Symposia on Discrete Algorithms, sponsored by SIAM and the Association for Computing Machinery. At SODA 2003, held in Baltimore, "Muthu" Muthukrishnan of Rutgers University and AT&T Labs–Research described some of the approaches researchers are developing for coping with modern—and future—data streams. Among the key ideas: making numerical "sketches" by projecting data onto randomly oriented subspaces, thereby ensuring, with high probability, the reconstruction of good approximations. At SODA 2004, held January 11–13 in New Orleans, Bernard Chazelle of Princeton University stressed the need for a new paradigm of "sublinear" computing, in which the computer never looks at more than a tiny fraction of the input.

Summary Execution

According to the Pythagorean system, Gargantua would, with his tutor, recapitulate briefly all that he had read, seen, learned, done and assimilated in the course of the day.

—Rabelais, *Gargantua and Pantagruel*

What distinguishes current data sets from earlier ones is automatic data feeds. "We don't just have people who are entering information into a computer," Muthukrishnan says. Instead, we have computers entering data into each other. And although they periodically crash, computers don't get tired, they don't take coffee breaks, and, like the Terminator, they absolutely will not stop.

The main limitation of computers is buffer size: the amount of data they can juggle at any one time. The trick is to use the buffer shrewdly, picking from the stream of data only what you need and letting everything else pass by.

Consider a simple, mnemosynic example. Suppose that someone removes a card from a deck of 52 and proceeds to show you the rest, one at a time, and then asks you to name the missing card. In principle, you could mentally put check marks in a 4×13 array and then scan the array for the unchecked entry. But very few people can do that, even with lots of practice. It's a lot easier to use some simple arithmetic: Convert each card into a three-digit number, where the hundreds digit specifies the suit—say 1 for clubs, 2 for diamonds, 3 for hearts, and 4 for spades—and the other two digits specify the value, from 1 (ace) to 13 (king); then simply keep a running sum, subtracting 13 whenever the two-digit part exceeds 13 and dropping any thousands digit (e.g., adding the jack of hearts—311—to the running sum 807 gives 1118, which reduces to 105). The missing card is simply what would have to be added to the final sum to get 1013 (so that for a final sum of 807, the missing card would be the 6 of diamonds).

More mathematically, Muthukrishnan explains, a number missing from a permuted presentation of the integers 1 to N can be spotted from their sum; the storage requirement is $O(\log N)$—essentially just the number of digits in N. If *two* numbers are missing, both can be identified by tracking the sum of the squares of the presented numbers along with the running sum, and then doing a little algebra with the two results. (Turning this into a trick with cards might stretch most people's arithmetic abilities.)

You can find k missing numbers by keeping running sums (modulo a prime bigger than N, if you like) of the first k powers; alternatively, you can track the first k symmetric polynomials, with updates $S_k = S_k + nS_{k-1}, \ldots, S_2 = S_2 + nS_1$, $S_1 = S_1 + n$ when the number n appears. Unfortunately, the computational complexity of the reconstruction algorithm—for

turning the final list of running sums into a list of missing numbers—grows exponentially with k. More precisely, the *known deterministic* algorithms are exponential in k; if you're willing to use a randomized algorithm, the computational complexity drops to $O(k^2 \log N)$.

i	1	2	3	4	5	6	7	8	9	10
y_i	117	82	314	15	197	216	73	155	19	101
$s(i)$	8	10	–	9	6	3	2	5	7	1
$p(i)$	10	7	6	–	8	5	9	1	4	2

Table 1. *A doubly linked list of numbers y_i. The successor and predecessor functions s(i) and p(i) point to the locations in the array of the next greater and next smaller numbers.*

Muthukrishnan distinguishes three types of data stream models. In each, input items $a_1, a_2, \ldots$ arrive sequentially and determine a signal A_i. The "time series" model is the simplest: A_i simply is a_i. In the "cash register" model, the signal A_i is a vector, and each a_i increments one of its components by a non-negative amount. The "turnstile" model allows the updates to be negative as well as positive; the name alludes to subway turnstiles, which can count the numbers of people entering or exiting at various stations. At issue are the processing time for each item a_i, storage space for the signal A, and processing time for analysis of the signal. Ideally, the processing time and storage space requirements should be sublinear—preferably poly-logarithmic in the size of the input—although signal analysis can in some cases be allowed to take longer (e.g., if it's done off-line by a bigger computer).

Sampling is one obvious way to deal with an overabundance of data. But sampling has its limitations. Some analyses require an unwieldy number of samples. And the turnstile model is particularly ill-suited for sampling, since some—and, in some cases, *all*—samples could be deleted. Imagine, for example, a billion lights, all turned off to begin with; someone randomly flicks a million of them on and then turns off all but four. If the problem is to identify the lights left on, simple sampling will almost certainly fail to find them.

Muthukrishnan and colleagues, including Anna Gilbert, Yannis Kotidis, and Martin Strauss of AT&T Labs–Research, have developed techniques for such problems. One method is to project incoming data onto a small set of random vectors in the signal space—in effect, sampling is done not on the input but on the space it comes from. The resulting "sketch" can be used to answer questions about the signal with preassigned precision and probability of success. Instead of absolute precision with vanishing probability, the method provides acceptable amounts of each, in somewhat the same way quantum mechanics requires uncertainties in position and momentum. (There is, however, no known uncertainty principle, à la Heisenberg, for data streams.)

Myopic Computing

Dans les champs de l'observation, le hasard ne favorise que les esprits préparés. (In fields of observation, chance favors only the prepared mind.)
—Louis Pasteur

Even when all the data is available, it can be undesirable to look at very much of it, Chazelle says. Consider the "successor searching" problem: From a (doubly) linked list of N numbers, $y_1, y_2, \ldots, y_N$, find the smallest of those (if any) that are greater than or equal to a given number x. (The links are "successor" and "predecessor" functions that point from y_i to $y_{s(i)}$ and $y_{p(i)}$, respectively—see Table 1 for an example.) There is an easy algorithm that runs in $O(N^{1/2})$ expected time. The first step is to pick $N^{1/2}$ elements at random and see where x would fit in this sublist. The true successor of x in the complete list can be found by working backward from its successor or forward from its predecessor in the sublist (it must have one or the other, and often has both). In any event, the algorithm looks, on average, at another $O(N^{1/2})$ numbers in the list.

Chazelle and colleagues Ding Liu of Princeton and Avner Magen of the University of Toronto have developed similar sublinear algorithms for a variety of geometric problems. They have found, for example, an $O(N^{1/2})$ time algorithm for checking whether two convex polygons, each with N vertices, intersect. The algorithm produces a point of intersection if there is one, and a separating line or plane if there isn't. (See Figure 1.) Roughly speaking, the algorithm finds a separating line touching the convex hulls formed by $N^{1/2}$ random vertices from each polygon (or it shows that the hulls intersect). It then looks at the portion of each original polygon that lies on the other poly-gon's side of this line; if there is an intersection, it must involve one (or both) of these portions, each of which has $O(N^{1/2})$ vertices. A similar algorithm works for convex polyhedra, although the details are necessarily more complicated. In both cases the complexity estimates are delicate. (For non-convex objects, all bets are off.)

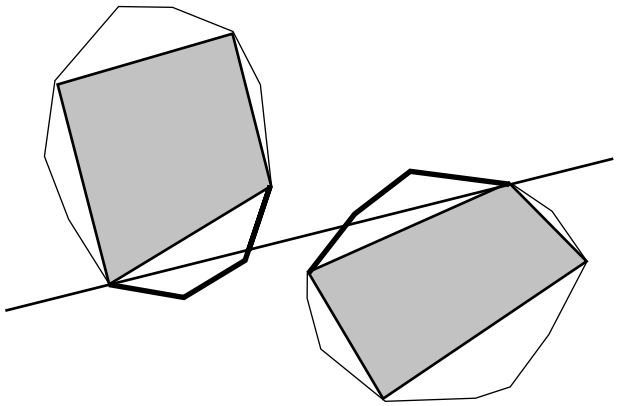

Figure 1. *First steps of an algorithm for checking whether two polygons intersect. A line separates two randomly generated convex hulls (shaded), each with a relatively small number of vertices. The line does not separate the original two polygons, although only a small piece of each, involving relatively few edges (heavy lines), lies on the "wrong" side of the line. If the two polygons intersect, the intersection must involve one of these pieces.*

The sublinear geometers have also found an $O(N^{1/2})$ time algorithm for approximating the length of the shortest path between any two points on the surface of a convex polyhedron. More precisely, the algorithm runs in $O(N^{1/2}/\varepsilon^{5/4})$ time, where ε is the relative error. The basic idea is to "wrap" the polyhedron with another polygon, one that has $O(1/\varepsilon^{5/4})$ vertices (notice that there's no N here!) and that stays within ε of the original polyhedron. The trick is to show that the length of the shortest path along the "ε-wrapper"

is within ε of its counterpart on the polyhedron (more precisely, the ratio of the two is within ε of 1). This actually isn't true for extremely short paths, because the wrapper gains length at every bend (compare the inner and outer lengths of an L-shaped ruler), but that's a minor technical point.

Geometric problems like these arise in computer graphics and in such applications as terrain maps for geographic information systems, but the need for sublinear algorithms runs the gamut of computer science. Chazelle is broadly interested in the issue of reformulating computational questions in ways that respect the data glut. "This is not out of frustration at being unable to resolve the big questions of complexity theory," he says. "On the contrary, it reflects the dazzling progress in data acquisition that has been witnessed in the applied sciences over the last decades: So much data, so little time."

Barry A. Cipra is a mathematician and writer based in Northfield, Minnesota.