From *SIAM News*, Volume 45, Number 4, May 2012

Fourier Algorithmics Enters a Growth Period

By Martin J. Strauss

We may have learned in grade school to evaluate a polynomial like $p(x) = 9x^2 + 5x + 8$ with three multiplications: $9 \cdot x \cdot x + 5 \cdot x + 8$, and to use around $n(n + 1)/2$ multiplications to evaluate a degree-n polynomial p at a *single* point. It is not hard to see how we can do this evaluation instead with just $O(n)$ multiplications. (Try to evaluate $p(x)$ with two multiplications and two additions.) Now suppose we want to evaluate p at n (special) points, the complex nth roots of unity. Remarkably, this can be done with a *total* of $O(n \log n)$ multiplications. Suppose now that only $k \ll n$ of the evaluations are "large" and the others "small," but we don't know in advance which the large ones are. Can we find the locations and approximate values of the k large evaluations, in total computation time not much greater than k? Notice that we don't have time to read all coefficients, or to evaluate a given polynomial exactly at even *one* point.

Remarkably, the answer is still yes! Beyond its appeal as a puzzle, this problem is a restatement of the discrete Fourier transform (DFT), which, along with continuous versions and their inverse transforms, is used throughout mathematics and related fields. In a paper to be published in the upcoming Proceedings of the 44th ACM Symposium on the Theory of Computing, four MIT researchers—Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price—present an algorithm that computes a Fourier transform faster than the celebrated fast Fourier transform algorithm [9]. More precisely, the HIKP algorithm gives an approximation that is good enough in practice, on most data seen in practice, in runtime proportional to $k \log(n/k) \log(n)$—better than the FFT's, as long as $k/n \to 0$, however slowly.

What was done, what are the limitations, and why are the limitations bearable in practice? What does the new algorithm give to applied mathematics, and how does it draw on classical results in applied mathematics?

■■■

The DFT takes a vector x of length n and multiplies it by the n-by-n matrix F whose (ω,t) entry is $\exp(2\pi i\omega t/n)$, returning the *spectrum* X of x. Equivalently, it regards the vector x as a sequence of coefficients of a polynomial and evaluates that polynomial on the complex points

$$e^{2\pi i/n}, e^{2\pi \cdot 2i/n}, e^{2\pi \cdot 3i/n}, \ldots e^{2\pi \cdot (n-1)i/n}.$$

In the theory of signal processing, the ωth element of the spectrum can be regarded as giving the frequency content of x at frequency ω. The Fourier transform can be used to analyze signals that come from, say, the human voice, handheld electronics, shopping patterns, or vibrations filtered by the earth.

The fast Fourier transform algorithm is generally credited to Cooley and Tukey [4]. The FFT algorithm computes an exact DFT on any input with computation time $O(n \log n)$, which is the best, or close to the best, possible. Briefly, the DFT and its inverse can be framed as finding coefficients of a degree-$(n - 1)$ polynomial that interpolates given values at the nth roots of unity. To find a degree-3 polynomial through values at $\pm 1, \pm i$, the algorithm first interpolates separately through ± 1 and $\pm i$, getting linear $p_1(z)$ and $p_i(z)$, then quickly outputs $1+z^2/2\, p_1(z) + 1-z^2/2 p_i(z)$. It is this reduction of one problem of size n to two problems of size $n/2$ that results in the claimed runtime.

In practice, many signals that we encounter and want to analyze have frequency content at only a limited number of frequencies. Which notes are being played on a piano? Generally, we care more when a skilled pianist uses at most ten fingers and less when an entire birthday party of six-year-olds use 88 fists. In the case of $k \ll n$ significant frequencies, algorithms (including HIKP) can be much faster than the general case. The HIKP algorithm cleverly draws on the signal processing literature in filter design and randomized algorithms.

The new algorithm builds on work from the last two decades on a class of algorithms that can be called sparse Fourier transform (SFT) methods. These results are of the following form: Fix n, k, and ε. (Additional techniques are available for determining an appropriate k.) Given vector $x \in \mathbb{C}^n$, compute some approximation $\tilde{X}$ such that

$$\|\tilde{X} - X\|_2 \leq (1 + \varepsilon)\|X_k - X\|_2,$$

where X_k is the best possible k-term approximation to the spectrum X of the input x. Many of these results, including HIKP, are *randomized* algorithms, meaning that they make random choices and must succeed with high probability with respect to those choices. (The input is *not* assumed to come from any distribution.)

The first result in this area was described by Mansour [13] (following related work by Goldreich, Levin, and Kushilevitz [7,12]), in which the problem was solved in runtime polynomial in $k \log n$. In [8], Gilbert, Muthukrishnan, and the author improved the runtime to k times an (unspecified and unoptimized) polynomial in $\log n$. There are additional important results, some in different models [1–3,5,10,11]. These al-gorithms, however, are faster than the FFT only for $k \leq n/\log n$ (or smaller); this may have been the chief obstacle inhibiting their wide adoption.

The HIKP runtime of $(k/\varepsilon) \log(n/k) \log(n)$, or $k \log(n)$ in certain (noiseless) special cases, beats the FFT up to constant factors for all values of k, improves on previous SFT methods in essentially all cases, and is optimal under certain assumptions. Simulations show that HIKP's runtime beats the FFT's with a break-even point of $k = 2^{17}$ and $n = 2^{22}$, a 3%-sparse signal, in the noiseless special case.

To see how sublinear-time Fourier algorithms work, consider a 1-sparse signal, i.e., $Fx = X$ is zero except at one unknown coordinate, ω. Suppose the length n is a power of 2. Examining $x_t \pm x_{t+n/2}$ gives the parity (least significant) bit of ω; other bits are learned similarly. With more than one frequency, we need a randomized filterbank that likely (over the algorithm's random choices) produces k new signals, in many of which exactly one frequency is present. A large fraction of the k terms present are found and removed, after which additional terms are sought.

The HIKP algorithm reflects innovations in several areas. It uses somewhat sophisticated filters that limit the number of samples from x while having good (though still approximate) filtering properties. Second, recovering $\log \log(n)$ bits of the $\log n$ bits in ω's identity all at once turns out to be faster overall than recovering just one bit at a time. Finally, once a frequency component is identified, its contribution is subtracted in later rounds from the *filtered* version of the signal, rather than from the original signal.

It would appear that none of these techniques alone would suffice—in particular, innovations from several disciplines combined to produce the improved properties of the HIKP algorithm. Together, the innovations make the new algorithm much faster and (at last!) a viable competitor to the FFT in practice.

Exploiting the new algorithm remains as a challenge for algorithm designers. Just as matrix multiplication algorithms and surrounding computational infrastructure exploit sparse structure when present in large-scale computations, Fourier transform algorithms and infrastructure will need to exploit sparsity for large problems.

References

[1] A. Akavia, *Deterministic sparse Fourier approximation via fooling arithmetic progressions*, Proceedings of the 2010 Conference on Learning Theory, A.T. Kalai and M. Mohri, eds., Omnipress, 2010, 381–393.

[2] A. Akavia, S. Goldwasser, and S. Safra, *Proving hard-core predicates using list decoding*, Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 2003, 146–157.

[3] E.J. Candès, J.K. Romberg, and T. Tao, *Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information*, IEEE Trans. Inform.Theory, 52:2 (2006), 489–509.

[4] J.W. Cooley and J.W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), 297–301.

[5] D.L. Donoho, *Compressed sensing*, IEEE Trans. Inform. Theory, 52:4 (2006), 1289–1306.

[6] A.C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, *Near-optimal sparse Fourier representations via sampling*, Proceedings of the 34th Annual ACM Sym-posium on Theory of Computing, J.H. Reif, ed., ACM, 2002, 152–161.

[7] O. Goldreich and L.A. Levin, *A hard-core predicate for all one-way functions*, Proceedings of the 21st ACM Symposium on Theory of Computing, D.S. Johnson, ed., ACM, 1989, 25–32.

[8] A. Gilbert, S. Muthukrishnan, and M. Strauss, *Improved time bounds for near-optimal space Fourier representations*, Proceedings of SPIE Conference on Wavelets, 2005.

[9] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, *Nearly optimal sparse Fourier transform*, Proceedings of the 44th ACM Symposium on Theory of Computing, 2012, to appear.

[10] H. Hassanieh, P. Indyk, D. Katabi, and E. Price, *Simple and practical algorithm for sparse Fourier transform*, Proceedings of the ACM–SIAM Symposium on Discrete Algorithms, SIAM, 2012, 1183–1194.

[11] M.A. Iwen, *Combinatorial sublinear-time Fourier algorithms*, Found. Comput. Math., 10:3 (2010), 303–338.

[12] E. Kushilevitz and Y. Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput., 22:6 (1993), 1331–1348.

[13] Y. Mansour, *Randomized interpolation and approximation of sparse polynomials*, Pro-ceedings of the International Colloquium on Automata, Languages, and Programming, W. Kuich, ed., 623, Lecture Notes in Computer Science, Springer, New York, 1992, 261–272.

Martin Strauss is a professor in the Departments of Mathematics and Electrical Engineering and Computer Science at the University of Michigan.