from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Algos is the Greek word for pain. *Algor* is Latin, to be cold. Neither is the root for *algorithm*, which stems instead from al-Khwarizmi, the name of the ninth-century Arab scholar whose book *al-jabr wa'l muqabalah* devolved into today's high school algebra textbooks. Al-Khwarizmi stressed the importance of methodical procedures for solving problems. Were he around today, he'd no doubt be impressed by the advances in his eponymous approach.

Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Don-garra of the University of Tennessee and Oak Ridge National Laboratory and Fran-cis Sullivan of the Center for Comput-ing Sciences at the Institute for Defense Analyses put togeth-er a list they call the "Top Ten Algorithms of the Century."

"We tried to assemble the 10 al-gorithms with the greatest influence on the development and practice of science and engineering in the 20th century," Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here's the CiSE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.

In terms of widespread use, George Dantzig's simplex method is among the most successful algorithms of all time.

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of **Krylov subspace iteration methods**.

These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that A is a huge $n \times n$ matrix, so that the algebraic answer $x = b/A$ is not so easy to compute. (Indeed, matrix "division" is not a particularly useful concept.) Iterative methods—such as solving equations of the form $Kx_{i+1} = Kx_i + b - Ax_i$ with a simpler matrix K that's ideally "close" to A—lead to the study of Krylov subspaces. Named for the Russian mathematician Nikolai Krylov, Krylov subspaces are spanned by powers of a matrix applied to an initial "remainder" vector $r_0 = b - Ax_0$. Lanczos found a nifty way to generate an orthogonal basis for such a subspace when the matrix is symmetric. Hestenes and Stiefel proposed an even niftier method, known as the conjugate gradient method, for systems that are both symmetric and positive definite. Over the last 50 years, numerous researchers have improved and extended these algorithms. The current suite includes techniques for non-symmetric systems, with acronyms like GMRES and Bi-CGSTAB. (GMRES and Bi-CGSTAB premiered in *SIAM Journal on Scientific and Statistical Computing*, in 1986 and 1992, respectively.)

1951: Alston Householder of Oak Ridge National Laboratory formalizes the **decompositional approach to matrix computations**.

The ability to factor matrices into triangular, diagonal, orthogonal, and other special forms has turned out to be extremely useful. The decompositional approach has enabled software developers to produce flexible and efficient matrix packages. It also facilitates the analysis of rounding errors, one of the big bugbears of numerical linear algebra. (In 1961, James Wilkinson of the National Physical Laboratory in London published a seminal paper in the *Journal of the ACM*, titled "Error Analysis of Direct Methods of Matrix Inversion," based on the LU decomposition of a matrix as a product of lower and upper triangular factors.)

Alston Householder

1957: John Backus leads a team at IBM in developing the **Fortran optimizing compiler**.

The creation of Fortran may rank as the single most important event in the history of computer programming: Finally, scientists

(and others) could tell the computer what they wanted it to do, without having to descend into the netherworld of machine code. Although modest by modern compiler standards—Fortran I consisted of a mere 23,500 assembly-language instructions—the early compiler was nonetheless capable of surprisingly sophisticated computations. As Backus himself recalls in a recent history of Fortran I, II, and III, published in 1998 in the *IEEE Annals of the History of Computing*, the compiler "produced code of such efficiency that its output would startle the programmers who studied it."

1959–61: J.G.F. Francis of Ferranti Ltd., London, finds a stable method for computing eigenvalues, known as the **QR algorithm**.

Eigenvalues are arguably the most important numbers associated with matrices—and they can be the trickiest to compute. It's relatively easy to transform a square matrix into a matrix that's "almost" upper triangular, meaning one with a single extra set of nonzero entries just below the main diagonal. But chipping away those final nonzeros, without launching an avalanche of error, is nontrivial. The QR algorithm is just the ticket. Based on the QR decomposition, which writes A as the product of an orthogonal matrix Q and an upper triangular matrix R, this approach iteratively changes $A_i = QR$ into $A_{i+1} = RQ$, with a few bells and whistles for accelerating convergence to upper triangular form. By the mid-1960s, the QR algorithm had turned once-formidable eigenvalue problems into routine calculations.

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents **Quicksort**.

Putting N things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Although it's possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first item on a list that's already sorted!), Quicksort runs on average with $O(N \log N)$ efficiency. Its elegant simplicity has made Quicksort the pos-terchild of computational complexity.

James Cooley

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the **fast Fourier transform**.

Easily the most far-reaching algo-rithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quick- sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

John Tukey

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an **integer relation detection algorithm**.

The problem is an old one: Given a bunch of real numbers, say $x_1, x_2, \ldots, x_n$, are there integers $a_1, a_2, \ldots, a_n$ (not all 0) for which $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$? For $n = 2$, the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of x_1/x_2. If x_1/x_2 is rational, the expansion terminates and, with proper unraveling, gives the "smallest" integers a_1 and a_2. If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, $B_3 = 3.544090$ and $B_4 = 3.564407$, of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 257^{30}.) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the **fast multipole algorithm**.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations—one for each pair of particles. The fast multipole algorithm gets by with $O(N)$ computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

What new insights and algorithms will the 21st century bring? The complete answer obviously won't be known for another hundred years. One thing seems certain, however. As Sullivan writes in the introduction to the top-10 list, "The new century is not going to be very restful for us, but it is not going to be dull either!"

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

"An intuitive algebraic approach for solving Linear Programming problems"

Source: Zionts [1974] (or many others).

$$\begin{array}{lrcl} [\max] z = & 0{,}56x_1 + 0{,}42x_2 & & \\ \text{s.to} & x_1 + 2x_2 & \le & 240 \\ & 1{,}5x_1 + x_2 & \le & 180 \\ & x_1 & \le & 110 \end{array} \qquad \{1\}$$

A[1]

$$\begin{array}{lrcl} [\max] z = & 0{,}56x_1 + 0{,}42x_2 & & \\ & x_1 + 2x_2 + \{x_3\} & = & 240 \\ & 1{,}5x_1 + x_2 + \{x_4\} & = & 180 \\ & x_1 + \{x_5\} & = & 110 \end{array} \qquad \{2\}$$

This has (always) an obvious, sure solution. Let

$$x_1, x_2 = 0 \qquad \{3\}$$

Then

$$\begin{bmatrix} x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 240 \\ 180 \\ 110 \end{bmatrix} \qquad \{4\}$$

$$z = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 240 \\ 180 \\ 110 \end{bmatrix} = 0 \qquad \{5\}$$

Is this optimal ? How to improve ?

There does not appear (Dantzig) to be a systematic way of setting *all* the nonbasic variables simultaneously to optimal values —hence, an *iterative*[2] method.

Choose the variable that increases the objective function *most* per unit (this choice is arbitrary), in the example, x_1, because its coefficient (0,56) is the largest.

According to the constraints, x_1 can be increased till:

B

$$\begin{array}{rcl} x_1 = 240 & & x_1 = 240 \\ 1{,}5x_1 = 180 & \rightarrow & x_1 = 120 \\ x_1 = 110 & & x_1 = 110 \end{array} \qquad \{6\}$$

The *third* equation (why ?) in {2} leads to $x_1 = 110$ and $x_5 = 0$. The variable x_1 will be the *entering* variable and x_5 the *leaving* variable:

[1] A, B, C identify the iteration, as summarized below.

[2] *Iterative:* involving repetition; relating to *iteration*. *Iterate* (from Latin *iterare*), to say or do again (and again). Not to be confused with *interactive*.

C
$$x_1 = 110 - x_5 \qquad \{7\}$$

Substituting for x_1 everywhere (except in its own constraint), we have

$$\begin{array}{rrrrrrrr} [\max]z = & 0{,}56(110 - x_5) & + 0{,}42x_2 & & & & & \\ & (110 - x_5) & + 2x_2 & + x_3 & & & = & 240 \\ & 1{,}5(110 - x_5) & + x_2 & & + x_4 & & = & 180 \\ & x_1 & & & & + x_5 & = & 110 \end{array} \qquad \{8\}$$

A
$$\begin{array}{rrrrrrr} [\max]z = & 0{,}42x_2 & & & - 0{,}56x_5 & + 61{,}6 & \\ & + 2x_2 & + \{x_3\} & & - x_5 & = & 130 \\ & x_2 & & + \{x_4\} & - 1{,}5x_5 & = & 15 \\ \{x_1\} & & & & + x_5 & = & 110 \end{array} \qquad \{9\}$$

which is of course equivalent to Eq. {2}.

We now have a **new** (equivalent) LP problem, **to be treated as the original was.** The process can continue *iteratively.*

$$\begin{bmatrix} x_1 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 110 \\ 130 \\ 15 \end{bmatrix} \qquad \{10\}$$

From Eq. {2} or Eq. {9}, respectively,

$$z = \begin{bmatrix} 0{,}56 & 0 & 0 \end{bmatrix} \begin{bmatrix} 110 \\ 130 \\ 15 \end{bmatrix} = 61{,}6 \qquad \{11\}$$

$$z = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 110 \\ 130 \\ 15 \end{bmatrix} + 61{,}6 = 61{,}6 \qquad \{12\}$$

Now, x_2 is the new entering variable. According to the constraints, it can be increased till:

B
$$\begin{array}{rcccl} 2x_2 & = 130 & & x_2 & = 65 \\ x_2 & = 15 & \rightarrow & x_2 & = 15 \\ 0x_2 & = 110 & & x_2 & = \infty \end{array} \qquad \{13\}$$

C
$$x_2 = 15 - x_4 + 1{,}5x_5 \qquad \{14\}$$

Substituting for x_2 everywhere (except its own constraint), we have

$$\begin{array}{rrrrrrrr} [\max]z = & 0{,}42(15 - x_4 + 1{,}5x_5) & & & - 0{,}56x_5 & & + 61{,}6 & \\ & + 2(15 - x_4 + 1{,}5x_5) & + x_3 & & - x_5 & = & 130 & \\ & x_2 & & + x_4 & - 1{,}5x_5 & = & 15 & \\ x_1 & & & & + x_5 & = & 110 & \end{array} \qquad \{15\}$$

$$\begin{array}{rrrrrrrr} [\max]z = & & & - 0{,}42x_4 & + 0{,}07x_5 & & + 67{,}9 & \\ & & \{x_3\} & - 2x_4 & + 2x_5 & = & 100 & \\ & \{x_2\} & & + x_4 & - 1{,}5x_5 & = & 15 & \\ \{x_1\} & & & & + x_5 & = & 110 & \end{array} \qquad \{16\}$$

A

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 110 \\ 15 \\ 100 \end{bmatrix} \qquad \{17\}$$

Now, x_5 is the new entering variable. According to the constraints, it can be increased till:

$$\begin{array}{rcrrcl} 2x_5 & = 100 & & x_5 & = 50 & \\ -1{,}5x_5 & = 15 & \rightarrow & x_5 & = \ldots & \\ x_5 & = 110 & & x_5 & = 110 & \end{array} \qquad \{18\}$$

B

$$x_5 = 50 - \frac{1}{2}x_3 + x_4 \qquad \{19\}$$

C

Substituting for x_5 everywhere (except its own constraint), we have

$$\begin{array}{rrrrrrrr} [\max]z = & & - 0{,}42x_4 & + 0{,}07\left(50 - \frac{1}{2}x_3 + x_4\right) & & + 67{,}9 & \\ & x_3 & - x_4 & + x_5 & = & 50 & \\ x_2 & & + x_4 & - 1{,}5\left(50 - \frac{1}{2}x_3 + x_4\right) & = & 15 & \\ x_1 & & & + \left(50 - \frac{1}{2}x_3 + x_4\right) & = & 110 & \end{array} \qquad \{20\}$$

$$\begin{array}{rrrrrrrr} [\max]z = & & - 0{,}035x_3 & - 0{,}35x_4 & & + 71{,}4 & \\ & & x_3 & - x_4 & + \{x_5\} = & 50 & \\ & \{x_2\} & + 0{,}75x_3 & - 0{,}5x_4 & = & 90 & \\ \{x_1\} & & - 0{,}5x_3 & + x_4 & = & 60 & \end{array}$$

A

$$\begin{bmatrix} x_1 \\ x_2 \\ x_5 \end{bmatrix} = \begin{bmatrix} 60 \\ 50 \\ 50 \end{bmatrix} \qquad \{21\}$$

Now, no variable produces an increase. So, this is a *maximum.*

In sum:

A In the system of equations, find the identity matrix (immediate solution).
B search for an *entering* variable (or finish)
C consequently, find a *leaving* variable (if wrongly chosen, negative values will appear).

References:

– ZIONTS, Stanley, 1974, "Linear and integer programming", Prentice-Hall, Englewood Cliffs, NJ (USA), p 5. (IST Library.) ISBN 0-13-536763-8.

– See others on the course webpage *(http://web.ist.utl.pt/mcasquilho)*.

❖