Current Trends
in Science

PLATINUM JUBILEE SPECIAL

Mathematics and computing

RAVINDRAN KANNAN

Microsoft Research Labs., India
e-mail: kannan@microsoft.com

The interaction of computing and mathematics is of course a very longstanding one in that, at some elementary level the words were more or less synonymous. However, mathematics has naturally focussed on abstract concepts and ideas while many scientific fields need and perform much computation. With fast modern computers, many more problems from the realm of sciences, commerce as well as other walks of life can be solved efficiently. Much of this improvement in efficiency is owed to the field of algorithms which at the theoretical level studies the inherent complexity of computational problems, providing principles and techniques for designing more efficient algorithms as well as trying to prove that for specific problems, efficiency cannot be improved beyond certain barriers whatever the method. [The latter area called lower bounds has perhaps been less successful in that many seemingly obvious statements turn out to be hard to prove.] While many practical and detailed considerations influence efficiency of algorithms, the focus in this article is the conceptual and general methodological foundations for efficiency which tend to resemble (as a field) the abstract nature of mathematics. This resemblance is not just superficial; it manifests itself in similar standards of rigorous proofs and to some extent also the aesthetics of choice of problems (though in this aspect, the connections to practical efficiency also plays a role in the case of algorithms) and indeed has fostered an active two-way interaction between mathematics and computer science as we seek to demonstrate here.

Instead of a formal sequence of definitions and statements, we will proceed by dealing with specific examples which are nevertheless illustrative of the overall field. Our first example is *linear programming*. The solution of simultaneous linear equations is an old subject studied by Gauss and many other mathematicians. It has the very interesting feature that many results like the existence of the so-called normal forms and the consequent linear algebraic results, which, on the face of it, do not have computational content, are nevertheless proved by algorithms, Gaussian elimination being the most noteworthy. In the 20th century, many constraint optimization problems were posed as simultaneous linear inequalities. One can ask whether or not there exists a solution to a system of linear inequalities or ask for a solution (of the linear inequalities) maximizing a linear function:

$$\text{Max } c \cdot x \text{ subject to } Ax \leq b,$$

(where if A is an $m \times n$ matrix, $Ax \leq b$ stands for m linear inequalities, one for each row of A). This is called the linear programming problem and algorithms for this problem are well-studied. The now classical algorithm called the simplex algorithm for this problem uses the underlying geometry. The set of solutions to $Ax \leq b$ is a 'polyhderon' (intersection of half-spaces) defined by the m inequalities. Assuming for the moment that it is bounded, it can be described as the convex hull of its extreme points (corners). A basic theorem is that there is a corner which is the (an) optimal solution for this problem. This can be intuitively seen by perturbation arguments; one rigorous proof of this is in fact given by the simplex algorithm (another case when an algorithm proves a purely structural result). While the simplex algorithm is immensely useful, it can be shown that there are pathological

Keywords. Algorithms; polynomial time.

examples where it takes a super-polynomial in m, n number of steps. More precisely: there is no uniform bound $p(m, n)$ where $p(.,.)$ is a polynomial such that for every linear programming problem, the simplex algorithm converges in at most $p(m, n)$ steps. This theoretical issue led to a sustained effort to find a truly (always) polynomial time bounded algorithm.

The first such algorithm was developed by Khachian based on work of Iudin, Nemirovsky and Shor and is called the ellipsoid algorithm. The algorithm is best stated for the (seemingly) simpler problem of determining whether there is a solution at all to $Ax \leq b$ and if so finding one (instead of optimizing). The ellipsoid algorithm starts with a large ball in n-space which is guaranteed to contain the polyhedron $Ax \leq b$ (even though we do not yet know if the polyhedron is empty or non-empty, such a ball can be found). It checks if the centre of the ball is in the polyhedron, if it is, we have achieved our objective. If not, the polyhedron is clearly contained in a half-space through the centre (called the separating hyper plane; this uses crucially the so-called separating hyperplane theorem from convex analysis). We then find an ellipsoid which contains the ball intersected with this half-space (it is not difficult to show that we can find one with volume at most $(1-4/n)$ times the volume of the ball). The ellipsoid is guaranteed to contain $Ax \leq b$ (as was the ball earlier). We now check if the centre of the ellipsoid satisfies the inequalities; if not, there is a separating hyperplane again and we may repeat the process. After a suitable number of steps, either we find a solution to the original $Ax \leq b$ or, we end up with a very small ellipsoid. Now if the original A, b had integer entries, one can ensure that the set $Ax \leq b$, after a slight perturbation which preserves its emptiness/non-emptiness, has a volume of at least some $\epsilon > 0$ and so if our ellipsoid has shrunk to a volume of less than this ϵ, then we know there is no solution and we can stop. Clearly this must happen within $\log_{(1-4/n)} V_0/\epsilon = O((V_0 n)/\epsilon)$, where V_0 is an upper bound on the initial volume. We do not go into details of how to get a value for V_0 here, but the important points are that (i) V_0 only occurs under the logarithm and (ii) the dependence on n is linear. These features, it can be shown, ensure a 'polynomial time algorithm'. The distinction between polynomial time bounds (as a function of the length of the input data with numbers being written in say binary) and non-polynomial time bounds has turned out to be very crucial. While as in this example, a new polynomial time algorithm does not immediately yield a better practical algorithm, often, the ideas and techniques have spawned new interesting mathematics and/or have been later developed to yield practical dividends.

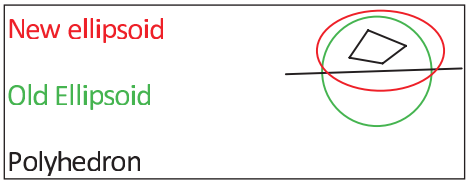

Figure 1. Ellipsoid algorithm.

The ellipsoid algorithm was a theoretical breakthrough, but did not do so well in practice. To get a better practical algorithm, it turned out one needed a new theoretical idea. This was the contribution of Karmarkar. His algorithm solves the optimization problem assuming we have a starting solution to $Ax \leq b$. We can conveniently look upon the ellipsoid algorithm as always keeping a ball containing the polyhedron; after we find an ellipsoid containing the half-space intersected with the ball, we make a linear transformation so that this ellipsoid is mapped to a ball, thus keeping the invariant that we have a ball containing the polyhedron. Karmarkar's algorithm uses projective transformations instead of linear ones at each step. It is an interior point method which at any step has a feasible point and tries to improve $c \cdot x$. The simplest way of improving $c \cdot x$ from a point x in the interior of the set would be to move along the gradient which is c. It is easy to see that this may be far from the direction in which the true optimal point lies. Karmarkar's method is a local improvement or gradient ascent algorithm too, but to ensure that one does not march off in a bad direction, it first makes a projective transformation which essentially makes the current point the center of a ball which is contained in the polyhedron. It then locally optimizes the linear function over this ball and repeats the process. Unlike the plain gradient ascent, here, one is able to argue that a certain 'potential function' does improve at each step, giving us convergence. With many more interesting details, this algorithm has turned out to be excellent on certain classes of linear programs.

As a second example, we take the computational problem of testing whether a given integer n is a prime number. The question is whether one can do this in polynomial time, i.e., time at most $O((\log n)^c)$, where c is a constant. [This is polynomial since the input n is assumed to be written in binary using $\log n$ bits.] There is a famous quote from Gauss which essentially says the dignity of mathematics demands that we find an efficient procedure to do this. In the 1970s, Miller and Rabin gave a randomized polynomial time algorithm, by which is meant that an algorithm which can use a random number generator, which when given any prime n as input, always returns 'prime' and when

given a composite as input returns 'composite' with probability at least 0.99. This algorithm used 'witnesses'. Any a such that:

$$a^{n-1} \not\equiv 1 (\mathrm{mod}\ n)$$

is clearly a witness to the compositeness of n by Fermat's little theorem. Unfortunately, there are composite numbers n for which every a satisfies $a^n = 1 (\mathrm{mod}\, n)$. Miller and Rabin developed a more sophisticated notion of witnesses and showed that for every composite number n, the set of non-witnesses forms a proper sub-group of the multiplicative group modulo n; hence by Lagrange's theorem at most $1/2$ the a's are non-witnesses implying that a random choice has probability at least $1/2$ of yielding a witness (for each composite).

For a long time the question of whether primality can be tested by a deterministic algorithm (one that does not use random numbers) in polynomial time was open. [While the actual definition of polynomial time boundedness was only made in the 60s and 70s, it can be argued that Gauss's remark foresaw this question as regards primality testing.] This was settled in a major breakthrough by Agrawal, Kayal and Saxena from the Indian Institute of Technology, Kanpur in 2002. Their algorithm is surprisingly simple and beautiful. We cannot however give a full description of it. But it relies on a version of Fermat's little theorem which asserts:

If a, p are relatively prime with $p > 1$, then the following polynomial identity holds if and only if p is prime:

$$(x - a)^p = x^p - a.$$

[The proof of this is simple.] Now, one could check primality by computing the two polynomials and checking if they are term by term equal. But since there are p terms and we want the running time to be polynomial in $\log p$, this is not a polynomial time algorithm. Instead, (and this is not made precise here), they check whether this identity holds modulo polynomials of the form $x^r - 1$ for certain r and show that, that plus a few other modifications suffice.

Another area in which there has been a very fruitful interaction between mathematics and algorithms is that of probabilistically checkable proofs (PCP for short). This draws from coding theory, algebra as well as cryptography and algorithms. First the traditional notation of formal proofs was inherited into computer science from logic. It is best to illustrate with an example. A graph $G(V, E)$ consists of a finite set V of vertices (here we take $|V| = n$) and a subset E of $V \times V$ called edges. For $u, v \in V$, we say that u is adjacent to v if $(u, v) \in E$. A path is a sequence of vertices, each adjacent to the previous one. A Hamilton path is a path which visits each vertex precisely once. Consider the problem: given a graph, decide if it has a Hamilton path. When it does, 'G has a Hamilton cycle' is a true statement. A simple proof that is it is true would be to supply a Hamilton path; once this 'proof' is written down (this takes only $O(n \log n)$ length to specify the order of the vertices on the Hamilton cycle with $O(\log n)$ bits for each label), it is easy to 'verify' the proof by checking (in polynomial time) that indeed each vertex is visited only once and that each neighboring pair in the sequence is adjacent. However, this does not say that finding a proof (or deciding whether there is any Hamilton cycle) can be done in polynomial time. Indeed this remains a major open problem in the field. PCPs make the verifier's job in a sense much easier by making the proofs much more redundant.

The main theorem about PCP says that there is a proof system for proofs of statements of the form 'G has a Hamilton path' which:

- has length at most a polynomial in $|V|$, and
- there is a polynomial time verification procedure which chooses at random $O(1)$ bits of the proof to check and has the guarantee that:
  - if the statement is indeed true, the $O(1)$ bits pass the check with probability 1 and
  - if the statement is false (and so the purported proof is dubious), the probability that the $O(1)$ bits pass the check is less that $1/10$.

The point here is that the proof is so redundant that if it is dubious, it has in a sense a 'lot of errors' so that a random 'spot check' of $O(1)$ bits finds an error for us. PCPs were studied by Arora and Safra (based on the work of many earlier authors) and the main PCP theorem mentioned above was proved by Arora, Lund, Motwani, Sudan and Szegedy. While it is difficult to describe the proof of the theorem in detail, a basic idea from coding theory used in PCPs is worth going over. One would like to take a string of bits and make an 'error-correcting code' from it, which is a longer string so that if the longer string is transmitted over possibly faulty lines, the errors introduced in the transmission can be corrected. The most basic method of getting an error-correcting code is the following: say the n bits of the string we want to transmit are $a_1, a_2, \ldots, a_n$. Consider the polynomial

$$a(x) = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + \cdots + a_n x^{n-1}.$$

If we evaluate it at $m > n$ points and transmit the m values (as well as the corresponding points), then clearly, we can reconstruct the polynomial from any n of the values by solving linear equations. Thus, we can afford upto $m-n$ errors and still get the correct values. [There are many details which have been omitted even from this simple scheme.] Such codes serve as the starting points of the PCP theorem. While we have here described the PCP theorem as specific to the Hamilton cycle problem, it is really about all problems in a class called NP via the beautiful theory of reducibilities built up by the work of Karp and others which shows that many seemingly unrelated computational problems are in fact reducible to each other.

As our last example, we discuss the use of random walks in algorithms. Random walks have been much studied by mathematicians and physicists. Their typical use in algorithms is to solve 'counting problems'. A natural example is to find approximately (to relative error at most any specified ϵ) the volume of a convex set K in n space. It is intuitively clear that this is reducible to the problem of counting (again to relative error ϵ) the number of lattice points in K (where we choose how fine the integer lattice is). Call the lattice L.

If the diameter of K is d, the only a priori upper bound we can give on the number of lattice points is d^n; so exhaustive listing is ruled out. A classical method for this problem would be a Monte-Carlo algorithm which encloses K is a simple set – say a rectangular solid R, generates (uniform) random points in R (which is easy) and sees what proportion of them fall into K. But of course, even for the unit ball B, (for which we can find the volume in closed form by just integration), this method fails because the smallest R enclosing the ball is a cube of side 2 and it is easy to see that $\mathrm{Vol}(R)/\mathrm{Vol}(B)$ behaves like $\mathrm{c}^n/n^{n/2}$ for some constant c, which means that we have to draw $n^{n/2}/\mathrm{c}^n$ samples (which is obviously not bounded by any polynomial in n) in R before we even get one in B, let alone a good estimate.

Another natural algorithm is to do a random walk: at time t, the walk is at some point $x \in K \cap L$. We pick one of its $2n$ coordinate neighbours, each with probability $1/(2n)$ and go to it if it is in K, otherwise we stay at x. This is one step of the walk. It is not difficult to show that under mild conditions, this walk converges in the steady state to the uniform distribution on $K \cap L$ (which assigns probability $1/|K \cap L|$ to each point). But we would like to argue that after a *polynomially bounded* (in n) number of steps, the probability distribution is close to the steady state. The rate of convergence has been related in the basic theory of random walks to the gap between the first and second eigenvalue of the transition probability matrix of the walk. In important papers, Jerrum and Sinclair and Alon related this gap to a purely combinatorial quantity called conductance. [Jerrum and Sinclair used this to supply the first proofs that some versions of random walks Physicists have used in statistical mechanics do in fact converge in time polynomial in the size of the problems; thus this is an example where the scientists have computed with a simple algorithm and the rigorous theorems about convergence came from the computer scientists.] In the case of the random walks on convex sets, conductance relates to something called 'relative isoperimetry'. The central question here can be phrased as:

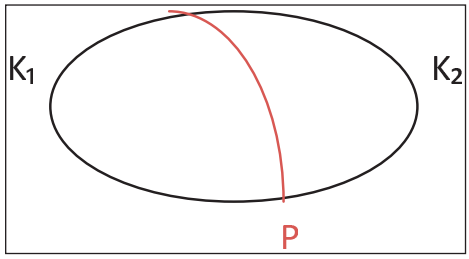

Figure 2. Relative isoperimetry.

If a convex set K in Euclidean n space of diameter d is cut into two parts K_1, K_2 by P (interior to K), prove

$$\mathrm{Vol}_{n-1}P \geq \frac{c\mathrm{Vol}_n(K_1)\mathrm{Vol}_n(K_2)}{d\ \mathrm{Vol}_n(K)}.$$

[Here all sets are assumed to be nice (measurable); Vol_n denotes usual (Les-begue) volume and Vol_{n-1} denotes 'surface area' in n space.]

This question arose in the work of Dyer, Freize and Kannan who gave the first polynomial randomized algorithm to estimate the volume of n dimensional convex sets. They made do with a more complex version of this relative isoperimetry result from differential geometry. The question above was answered in the affirmative by Lovász and Simonovits. It has been extended to other commonly occurring multi-variate probability measures (like the Gaussian). The area has also raised some questions which have led to interesting developments in functional analysis as well as probability.

Of course, what has so far been covered is just a small sample of the contributions of mathematics and theoretical computer science to each other and is by no means anything like an exhaustive list. But it provides a brief glimpse of the interaction of many areas of mathematics – convex geometry, analysis, number theory etc. – and algorithms. Many challenges and questions continue to be probed.

In closing, we mention very briefly some more topics. One of the crucial applications of number theory (especially primality testing, factoring an integer into its prime factors and more sophisticated developments like theory of elliptic curves) has been to cryptography where there has been a constant interaction. Numerical analysis which in a sense predates modern computer science is a very important area and not only has many connections to practical computing, but also to analysis. But unfortunately, many computer science curricula of the last few decades have divorced themselves from numerical analysis, but, the remarkable realization in the last few years that numerical analysis, especially combined with randomized algorithms scales up very well has led to a renewed interest. Topology has had a long interaction with graph theory. The difficult area of lower bounds draws really from a variety of mathematical areas.

References

[1] Arora S, Lund C, Motwani R, Sudan M and Szegedy M 1992 Proof verification and hardness of approximation problems; in *Proc. 33rd Annual IEEE Sym. Foundations of Computer Science*, 14–23.

[2] Agrawal M, Kayal N and Saxena N 2004 Primes is in P; *Ann. Math.* **160** 781–793.

[3] Dantzig G B 1951 Maximization of linear function of variables subject to linear inequalities; in: *Activity analysis of production and allocation* (ed) Koopmans T C (New York: Wiley) 359–373.

[4] Dyer M, Frieze A and Kannan R 1991 A random polynomial time algorithms for computing the volume of convex sets; *J. ACM*, 1–17.

[5] Iudin D B and Nemirovsky A S 1976 Information complexity and effective methods for solution of convex extremal problems; *Ekonomika i Matematichesky Metody* **12** 357–369.

[6] Jerrum M and Sinclair A 1989 Approximating the permanent; *SIAM J. Computing* **18(6)** 1149–1178.

[7] Khachiyan L G 1979 A polynomial time in linear programming; *Doklady Akademii Nauk SSSR* **244** 191–194.

[8] Karmarkar N 1984 A new polynomial time algorithms for linear programming; *Combinatorica* **4** 373–395.

[9] Lovász L and Simonovits M 1993 Random walks in a convex body and an improved volume algorithm; *Random structures and algorithm* **4**.

[10] Miller G L 1976 Riemann's hypothesis and tests for primality; *J. Computer and Systems Sci.* **13** 300–317.

[11] Rabin M O 1980 Probabilistic algorithm for testing primality; *J. Num. Theo.* **12** 128–138.

[12] Shor N Z 1977 Cut-off method with space extension in convex programming problems; *Cybernatics* **13** 94–96.