8. Bertrand Russell, Mathematical logic as based on the theory of types, Amer. J. Math., 30 (1908) 222–262.
9. Th. Skolem, Einige Bemerkungen zur axiomatische Begründung der Mengenlehre. Fifth Congress of Scandinavian Mathematicians, 1922, Helsingfors, 1923, pp. 217–232.
10. Hao Wang, From Mathematics to Philosophy, London, Routledge and Kegan Paul, 1971.
11. A. N. Whitehead and Bertrand Russell, Principia Mathematica vol. 1, Cambridge University Press, Cambridge, England, 1910; 2nd edition, 1925.
12. E. Zermelo, Untersuchungen über die Grundlagen der Mengenlehre I, Math Ann., 65 (1908) 261–281.

COMPUTER SCIENCE, MATHEMATICS, AND THE UNDERGRADUATE CURRICULA IN BOTH

ANTHONY RALSTON
Department of Computer Science, SUNY at Buffalo, Amherst, NY 14226

There is a simple and basic fact about a computer which will, in the decades and centuries to come, affect not so much what is known in mathematics as what is thought important in it. This is its finiteness.

Wallace Givens [**7** (1966)]

We have thought of the calculus as the beginning, the gateway. But should it be?

H. O. Pollak [**21** (1978)]

If new disciplines may be described as emerging from old ones, then computer science may be said to have sprung mainly from mathematics, although, of course, the influence of electrical engineering was also considerable. In recent years, however, the ties between mathematics and computer science have been steadily weakening. This has been coupled with a declining belief on the part of some (most?) computer scientists in the importance of mathematical training and mathematical tools in the education of computer scientists and in research in computer science. Some of the evidence for this "declining belief" will be considered later in this paper. Here I note only my belief that, contrariwise, the importance of mathematics in computer science is and should be growing rapidly. It is this belief which motivates much of this paper.

While thus far in the three decades of computing and in the two of computer science there has been considerable discussion of the influence, or lack of it, of mathematics on computer science, there has been little discussion of what should—or should not—be the influence of the development of computer science on mathematics research and education. The other major motivation of this paper stems from my belief that the growth of computer science should be having, but has not had, a profound effect on undergraduate mathematics education.

1. The Role of Mathematics in Computer Science Education. In the 1960's, in addition to the emerging departments of computer science themselves, there were numerous examples of departments of statistics and computer science, or applied mathematics and computer science, or computer programs and options within departments of mathematics. And this was in contrast to a

Anthony Ralston received his Ph.D. in Mathematics under Eric Reissner at MIT. He has held positions at Bell Laboratories and Stevens Institute of Technology and is now Professor of Computer Science at SUNY at Buffalo. He has been a visiting faculty member at the University of Leeds, the Institute of Computer Science of the University of London, and University College, London. He is a former president of the Association for Computing Machinery and of the American Federation of Information Processing Societies. Most of Professor Ralston's early research papers were in numerical analysis. In recent years, after publishing mainly on strictly computer science topics, he has "returned" to mathematics with current interests in discrete mathematics and mathematics education. He is the author or editor of eight books, including *A First Course in Numerical Analysis* (second edition, with P. Rabinowitz, McGraw-Hill, 1978) and the *Encyclopedia of Computer Science* (Van Nostrand Reinhold, 1976).

few—but very few—joint efforts with engineering departments. In 1980, however, we find that, while some smaller colleges and universities, most of which have never had the resources to start separate departments of computer science, still house computer science within a department of mathematics, there is no example in the United States of a computer science program with a significant research reputation either housed in a department of mathematics or jointly with any mathematics-like department. But, in addition to stand-alone departments of computer science, which are the general rule, there are several significant examples, MIT and Berkeley being the best known, of joint departments of computer science and electrical engineering. (These are always called departments of electrical engineering and computer science, but that is only because computer scientists have made relatively little headway in teaching engineers the meaning of lexical order.)

Why have the ties between mathematics departments and computer science departments weakened so much over the past decade? For a variety of reasons, I think:

1. There has been a relative decrease in importance in computer science of some of its more traditional mathematical areas, such as numerical analysis.
2. The dominant role played by mathematicians in the establishment of computer science as a discipline explains the early close association of the two disciplines. But the difficulty computer science had in being recognized as a separate discipline (and not just as a branch of mathematics) at many universities, and in the academic and scholarly worlds more generally, explains the desire of many computer scientists to emphasize the distinctness of computer science and, therefore, to form separate departments.
3. Beginning with the early 1970's computer science faculties became increasingly populated by those whose degrees were in computer science. Data on this are lacking until the mid-seventies but by then [**26**] almost 40 percent of the faculty in Ph.D.-granting departments of computer science held Ph.D.'s in computer science. Moreover those with Ph.D.'s in mathematics consisted of only about 20 percent of the total faculty, and this percentage appeared to be dropping rather rapidly. Certainly by the mid-seventies there were very few computer science faculty members with *recent* mathematics Ph.D.'s. Thus the personnel ties between mathematics and computer science have been steadily attenuated since the establishment of computer science as a discipline.
4. Finally, and not least, mathematics departments have not been, and are not now, very hospitable to the ideas and techniques of computer science, surely much less so than many electrical engineering departments. This was particularly true in the sixties, as mathematics departments focused increasingly on abstract branches of mathematics. Although the seventies saw some redress of this trend, there is little doubt that computer scientists, the vast majority of whom are pragmatically and applications oriented, to a considerable degree still see little in the prevailing philosophy of departments of mathematics that implies an understanding of, or a sympathy for, the interests and aims of computer science. This is a generalization, of course, but a correct enough one to explain much of the current lack of close collaboration between mathematics departments and computer science departments.

The loosening of the ties or, if you will, the cooling of the relationship between mathematics and computer science as disciplines has been reflected in the place of mathematics in the computer science curriculum (to the detriment, I believe, of both disciplines). In the first important publication on the undergraduate curriculum in computer science [**1**], there was a clear statement of the distinction which computer scientists saw between mathematics and computer science, a distinction which reflects the trends in the two disciplines in the sixties:

> The mathematician is interested in discovering the syntactic relation between elements based on a set of axioms which may have no physical reality. The computer scientist is interested in discovering the pragmatic means by which information can be transformed to model and analyze the information transformations in the real world.

Nevertheless, because with little or no exception the authors of this report had had their academic training in mathematics, they believed strongly in the importance of mathematics in computer science education. This is reflected in their final report known as Curriculum 68 [**2**].

It is generally believed and certainly many people have testified, both formally and informally, that Curriculum 68 had far-reaching effects on the development of undergraduate computer science curricula in the United States and, indeed, elsewhere. The attitude of the Curriculum 68 committee toward mathematics is clear:

> The Committee feels that an academic program in computer science must be well based in mathematics since computer science draws so heavily upon mathematical ideas and methods. The recommendations for required mathematics courses given below should be regarded as minimal; obviously additional course work in mathematics would be essential for students specializing in numerical applications.

The "minimal" mathematics requirement in Curriculum 68 consists of six required courses—three of the four semesters of the calculus sequence, a course in probability, a course in numerical calculus and one in discrete structures—plus two of four other courses—the fourth semester of the calculus sequence, a semester of advanced calculus, a course in algebraic structures, and one on probability and statistics. Two comments are pertinent here:

1. All the courses, with the exception of discrete structures and numerical calculus, were and are standard mathematics (or statistics) department courses. Indeed, these eight courses are specified to be precisely those recommended by CUPM [**15**].
2. The discrete structures course is the only recognition in Curriculum 68 that the mathematics needs of computer science are different from those in the physical sciences. However, the description of this course in Curriculum 68 read: "Review of set algebra including mappings and relations. Algebraic structures including semigroups and groups. Elements of the theory of directed and undirected graphs. Boolean algebra and propositional logic. Applications of these structures to various areas of computer science." This illustrates that, while there are distinctions between it and the usual subject matter of abstract algebra courses (e.g., graph theory), the course as a whole was not unlike many abstract algebra courses. The attitude toward mathematics of the developers of Curriculum 68 was quite appropriate to the state of computer science in 1968. This state can be quickly summarized as follows: Mathematics education is central to the undergraduate program in computer science; the mathematics to be taught to undergraduate computer scientists should be essentially that traditionally taught to physical science majors.

During the past decade there have been two somewhat contradictory trends in the relationship between computer science and mathematics. On the one hand, the place of mathematics in the undergraduate computer science curriculum was steadily weakened, the culmination of this being Curriculum 78, which we consider below. On the other hand, there has been an increasing recognition of the relative importance of discrete mathematics to the computer scientist. For example, in his 1974 paper Knuth [**13**] notes: "Discrete mathematics, especially combinatorial theory, has been given an added boost by the rise of computer science." And he goes on to say, "I have been naturally wondering whether or not the traditional curriculum (the calculus courses, etc.) should be revised in order to include more of these discrete mathematical manipulations or whether computer science is exceptional in its frequent application of them."

The latter part of the 1970's saw the publication of two major undergraduate curriculum studies in computer science, the first (in 1977) a report of the IEEE Computer Society [**10**] and the second the long-awaited revision of Curriculum 68, called Curriculum 78—although finally published in 1979 [**3**]. Noteworthy about the former is its almost total lack of discussion of the mathematics requirements for a program in computer science and engineering. Although portions of a discrete structures course and an analysis of algorithms course are part of the core curriculum, these are part of a theory of computation sequence. Moreover, in the only sample program given in the report, it is assumed that students will take the usual calculus-linear algebra sequence in the first two years. And there is no explicit probability or statistics requirement.

Curriculum 78, while it contains considerably more discussion of the mathematics requirements for an undergraduate program than the IEEE Computer Society curriculum, is nevertheless quite disappointing relative to Curriculum 68. A comparison between Curriculum 68 and Curriculum 78 discloses that [**23**]:

1. Whereas Curriculum 68 required the student to take eight semester-length mathematics courses, Curriculum 78 requires only five mathematics courses.
2. Whereas the mathematics courses in Curriculum 68 formed an integral part of prerequisite structure, in Curriculum 78 there is no mathematical prerequisite for any undergraduate computer science course, with the exception of three advanced and clearly quite mathematical courses.
3. Curriculum 78 continues the reliance of Curriculum 68 on continuous mathematics.

We have, then, the situation that, while many computer scientists believe that more, not less, mathematics is needed in undergraduate computer science education and that the traditional calculus sequence is perhaps not the most appropriate mathematics for this curriculum, the main curriculum committees of the professional societies in computing are either ignoring this question or taking an almost opposite point of view.

A central thesis of this paper is that the role of mathematics in computer science needs to be both strengthened and redirected. The argument for redirection is made later in this paper. But the importance of redirecting the curriculum must be proportional to the importance of mathematics in the computer science curriculum itself. If mathematics is just not very important in the undergraduate computer science curriculum, then it would be hard to be very excited about changing this component of the curriculum.

How important then is mathematics in the undergraduate computer science curriculum? Curriculum 78 will not only bolster the view of many outside the discipline that Computer Science = Programming but it will also surely strengthen the view that, while mathematics may be good for the soul (and maybe for the mind also), it has little direct relevance to undergraduate computer science. My view on this can best be expressed by the following quotation [**23**]:

> The principles and theories of any science give it structure and make it systematic. They should set the shape of the curriculum for that science, for
>
> —only in that way can they provide a framework for the mastery of facts, and
>
> —only in that way will they become the tools of the practicing scientist.
>
> This is as true for computer science as it is for mathematics, for the physical sciences, and for any engineering curriculum. Inevitably, for any science or any engineering discipline, the fundamental principles and theories can only be understood through the medium of mathematics.

But perhaps the strongest case for a strong mathematical component in undergraduate computer science can be made by noting that, even if the false view that Computer Science = Programming were true, the case for more mathematics in computer science would be overwhelming (see, for example, [**6**]). The single biggest challenge faced in the overall area of programming is how to provide it with some theoretical (i.e., mathematical) underpinnings that will enable software to be developed and maintained more efficiently and with a higher level of reliability than is normally attained nowadays. Many of the most important research areas in computer science, such as the work in program verification and structured programming, are concerned with these problems. And all of this is highly mathematical. Moreover, the need to teach computer science students how to analyze and verify algorithms is becoming increasingly clear. And this, too, requires intensive mathematical analysis. Finally, at the risk of noting the obvious, two purposes of all mathematics education are to instill in students the meaning of rigorous thinking and an appreciation of abstraction. The importance of both of these to programming—and to computer science more generally—can hardly be overstressed.

The mathematical tools and methodology required by computer science students are, with few exceptions, not the classical tools and methodology of (continuous) analysis. Rather they are the tools of discrete analysis, and it is this type of mathematics which should be the heart of the undergraduate mathematical training of computer science students. In Section 3 we shall develop this idea at length. But first, in the next section, we shall consider the influence of computing and computer science on the mathematics curriculum.

2. Computing, Computer Science, and the Mathematics Curriculum. At the risk of some modest oversimplification, the following describes the first two years of the undergraduate mathematics curriculum during the past two-thirds of a century. (This paper is a shortened version of a much longer paper by the author [**22**] which contains an extensive review supporting these conclusions. The longer report may be obtained from the author by writing to the Department of Computer Science, 4226 Ridge Lea Road, Amherst, NY 14226.)

1. The essential change over this period has been from a college algebra–trigonometry–some calculus in the first two years situation, to a two-year calculus–analytic geometry sequence, to the more common calculus–linear algebra sequence of today.
2. This change has been accompanied by changes in approach in the presentation of material from the relatively informal and concrete, to the rigorous and abstract, and now back to a more informal approach. But, whenever and however taught, calculus has remained the entrance point to all study of higher mathematics for professional mathematicians, scientists, engineers, and, more recently, social, behavioral, and management scientists.

As far as computing is concerned, two trends over the past two decades are discernible in the attitude of mathematicians toward the effect computing might have on the mathematics curriculum. One is the attempt to incorporate computing into the traditional curriculum, most notably in the calculus sequence. This trend is exemplified by many articles published over the past 10 or so years. Gordon [**8**] has a good bibliography of articles on this subject. Another and rather more recent trend concerns the attempt to combine computing and computer science with the mathematics curriculum to provide a more broadly based major centered on mathematics.

One may argue about whether or not mathematicians have embraced the use of computers in their courses as rapidly or intensively as they should have. But our purpose here is only to note that the use of computers in the calculus and other courses, however desirable and valuable, has not had any marked effect on the subject matter of undergraduate mathematics itself. Indeed, even when the mathematical community has considered programs in "computational mathematics" [**16**], the essential subject matter for the first two years has hardly changed. The only nontraditional mathematics course in the CUPM Undergraduate Program in Computational Mathematics is a course in combinatorial computing. In fact, the mathematics requirements of this program are quite similar to those of Curriculum 68.

A number of mathematicians have recognized that the computer allows a number of areas of mathematics to be taught experimentally (e.g., McKenna [**19**]). But there has been little realization that the advent of computers *and* computer science might suggest some fundamental changes in the undergraduate curriculum itself.

It is natural and proper that mathematicians have been worried and frustrated by declining enrollments, particularly at the upper division undergraduate and graduate levels. The frustration and worry were succinctly juxtaposed in a 1979 report on the role of applications in the undergraduate curriculum [**20**]:

> Mathematics is playing an increasing role in the physical and management sciences and in engineering, in both academic and nonacademic spheres. Yet enrollments are declining drastically in the mathematics major and in many of the traditional postcalculus courses.

An even more pessimistic outlook is contained in a view toward 1984 in [**18**, p. 42] which predicts that by then the only service courses left to mathematics departments would be "remedial algebra... and two semesters of calculus."

A natural response to these unwelcome trends is to try to make the mathematics curriculum more attractive to students. One general approach, already in progress before the advent of declining enrollments, was to emphasize less abstract approaches and subject matter. Related to this are attempts to introduce more applied mathematics and applications of mathematics into the curriculum. But by far the most popular and recommended response—not, by the way, unrelated to the other two—is to introduce more computing and computer science into the mathematics curriculum, sometimes into programs called "mathematical sciences." My own belief is that a mathematical sciences curriculum not only makes disciplinary sense but is a proper and reasonable response to enrollment difficulties and societal needs. There are, however, two pertinent comments about such curricula:

1. They imply a belief on the part of mathematicians that computer science is a "mathematical science" (whatever that is) in much the same sense as statistics is. But those portions of computer science which are growing most rapidly (i.e., computer systems and software) are, despite their great needs for mathematical underpinnings, not mathematics in any usual sense of the word. Indeed, one of the reasons for the aforementioned greater affinity of computer science departments with departments of electrical engineering than with mathematics departments is because the core areas of computer science are closer in outlook and approach to engineering than to mathematics.
2. The curricula which have been proposed for the mathematical sciences are, with the typical exception of one discrete mathematics or combinatorial computing course, quite traditional in the sense that the courses they propose are typically amalgams of traditional mathematics, computer science, and statistics courses. It would be wrong to be too critical of this, because, in times of limited resources, there are obvious attractions to new programs which require the offering of few new courses. Nevertheless it is my belief that, through resource inhibitions or lack of imagination, these programs have missed an opportunity to be far more useful and innovative than they are.

The conclusion I would emphasize here is that there is no evidence of any significant impact of computing or computer science on the essential subject matter of the undergraduate mathematics curriculum, particularly the first two years. I believe that there should be such an impact and in Section 4 I return to this subject.

3. Mathematics for Computer Science. In order to approach the question of how the undergraduate mathematics curriculum itself might be restructured, it is instructive to consider first the mathematics requirements of an undergraduate computer science curriculum.

The computer science courses taken by a student during the first two undergraduate years generally consist roughly of the following:

1. A one-semester or, increasingly commonly, a one-year sequence introduction to computer science which emphasizes
   —algorithmic processes
   —learning to program in one or more higher level languages
   —programming methodology and style
2. A one-semester course on assembly language programming and machine organization.
3. A one-semester data structures course.

What mathematics is desirable and/or necessary to support such courses? To begin, we note the total absence of any *need* for calculus in any of these courses. This is not to say that, for purposes of examples, it would not be useful to draw upon the calculus. But it is certainly not necessary, and the potential for examples in all the courses listed above is so great that an inability to use calculus is no great loss.

We have argued earlier that mathematics is more important than ever for computer science students. But if not calculus, then what? The answer follows most easily from the observation that one definition of computer science favored by some is the study of algorithms [**13**]. Whether or not you find this an acceptable definition, it is undeniable that the design, analysis, and verification of algorithms should play a crucial role in the first year of computer science and an important role in many subsequent courses. While the importance of algorithms can also be stressed in calculus courses (see, for example, Rosser [**24**]), not only is discrete mathematics rather than continuous mathematics the main mathematical component of most computer algorithms but by far the fastest growing area of algorithmics is that which needs discrete mathematics.

Since discrete mathematics is one of those amorphous terms with various different meanings to different people, we should first indicate as clearly as possible what we shall take its meaning to be in the remainder of this paper. As used here, discrete mathematics covers those branches of mathematics which focus mainly or entirely on discrete objects such as combinatorics, graph theory, abstract algebra, linear algebra, number theory, set theory, and discrete probability. The term also naturally implies an emphasis on discrete analysis in contradistinction to the traditional emphasis on the analysis of continuous functions.

To get some idea of the kinds of mathematics needed to support the computer science courses listed above, let us list some topics which require mathematics and which are always, or often, covered in such courses:

—analysis of algorithms, including the use of induction, discrete probability, often the summation or manipulation of series, and various aspects of basic combinatorial theory;
—verification of algorithms, including the generation of logical assertions about algorithms and informal proofs of these;
—numbers and number systems, particularly the discrete number system which is the basis of all computer arithmetic;
—simple queueing theory used to discuss basic aspects of scheduling;
—random number generation, since random number generators are often used in problems given in basic computer science courses.

In addition, various other topics in discrete mathematics may also be covered.

Most mathematicians would agree, I think, that, for students intending to major in disciplines other than mathematics, the mathematics they study during their first two undergraduate years should support and, where possible, be coordinated with corresponding courses in their intended major discipline. Therefore, in the absence of mathematics courses which support the first two years of computer science, instructors in computer science are usually forced to cover the necessary mathematics themselves. Inevitably this has two bad results:

—the time spent on the mathematics is a distraction from the essential computer science subject matter in addition to lessening the computer science component of the courses;
—in order to minimize the distraction and the loss of computer science subject matter, the mathematics tends to be covered cursorily, incompletely, or both.

It is hardly news that the need to shore up a student's understanding of subject B in a course on subject A is usually unsuccessful in imparting real understanding of B and detracts from the coverage of A.

The sequence of courses to be outlined briefly now is only one possible solution to this problem. This sequence is discussed in extensive detail in [22]. Here, then, is an outline of a possible two-year sequence in discrete mathematics.

First Year

A one-year course in discrete mathematics covering:

Algorithms and Their Analysis
Introductory Mathematical Logic
Limits and Summation
Mathematical Induction
The Discrete Number System
Basic Combinatorial Analysis
Difference Equations and Generating Functions
Discrete Probability
Graphs and Trees
Basic Recursion and Automata Theory

Remarks.

1. The "glue" which should be used (and which the author has used in such a course) to hold together an (otherwise) disparate set of topics is the notion of an algorithm and the analysis of algorithms.
2. The above should be judged by its general flavor and the philosophy it implies and not by its details. With Berztiss [4], I believe that the syllabus itself "is of no great importance, as long as the basic intents are being satisfied." This syllabus should only be considered as an example of one of many possible syllabuses and should be judged as to
   —whether the topics chosen are important (rather than most important)
   —whether they are accessible to first-year students
   —whether they will tend to develop mathematical maturity and sophistication as well as the calculus
   —whether they are appropriately supportive of the computer science curriculum.

Second Year

For this year we recommend a semester each of linear algebra and abstract algebra. The former needs little more said about it. It should be closely modeled on the current one-semester course so prevalent in the second-year mathematics curriculum, partly because this would involve minimal curriculum upset and partly because this course is quite relevant to the mathematical needs of computer science students.

Since abstract algebra is usually a third-year, two-semester sequence, it is not quite so easy to specify what to do here. Obviously a single semester of the usual sequence is a possible solution motivated both by the desire to avoid unnecessary curriculum upset and because, as we shall argue in the next section, the curriculum outlined here is also intended to be reasonable for mathematics majors. Nevertheless, because computer science students do have specific needs in the area of abstract algebra, it is worthwhile to outline the set of topics which such a course should include:

Functions, relations, and equivalence
Fields, rings, and ideals
Groups, including at least permutation and cyclic groups, semigroups and cosets
Lattices
Algebras generally, including Boolean algebras.

In recent years there have been several books published, some quite good, with titles like "Discrete Mathematics for Computer Science" (e.g., Stanat and McAlister [25], Tremblay and Manohar [27], and Levy [14]), which focus, in the main, on topics like those above and which are aimed at second-year computer science students.

Precisely how the two-year curriculum discussed would support the first two years of computer science is beyond my scope here but is discussed in [22].

What mathematics should follow the first two years for computer science students? A third year of undergraduate mathematics is almost a necessity for such students; the almost obvious choices for this third year are:

1. *Calculus.* We argued previously only that calculus is hardly needed in the first two years of a computer science curriculum. But we have not argued, and would not argue, that calculus plays no role in computer science more generally. Indeed, there are numerous areas of computer science where calculus plays an important role, of which we mention only a few for illustration:

(a) One can go to considerable depth in the *analysis of algorithms* without a need for the tools of calculus, but these tools are eventually invaluable, as a perusal of Knuth's books [12] will exemplify.
(b) The subject matter of continuous *probability and statistics* are necessary to the practicing computer scientist in a variety of contexts (e.g., in the analysis of hardware and software performance and in the design and analysis of simulation models and experiments).

(c) Most computer scientists still view *numerical analysis* as a branch of their subject; of course the foundations of much of numerical analysis as well as its tools are dependent upon calculus.
(d) Some topics, such as *queueing theory*, that can be successfully introduced without calculus can be considered in any depth only with the help of calculus.

Finally, we note that the tools of discrete and continuous mathematics have always been interrelated in much of mathematics education. To propose the study of one to the exclusion of the other, even if one is much more relevant to a discipline than the other, would be pedantic.

Thus we argue for a semester of calculus at the beginning of the third year for computer science students. Why only a semester? For two reasons:

(a) Because a great deal more can be covered in one semester at the junior level than at the freshman level due to the presumed mathematical maturity of students who have had a two-year curriculum as outlined above.
(b) Because some topics (e.g., limits, summation of series) covered in the normal calculus sequence and related notation will have been introduced in the first two-year sequence, less time need be spent on them in calculus itself.

Thus, it seems plausible to me that at the end of a one-semester calculus course in the junior year, the student would have covered as much material as at the end of a one-year freshman sequence.

As a final word on this subject and to note that the idea of preceding calculus with discrete mathematics is not new, here is a quote from a section in [17] entitled "Thoughts on a 'Postponed' Calculus Course with Emphasis on Numerical Methods":

> In recent years many questions have been raised about the special role played by the basic calculus sequence as the first set of courses in traditional college mathematics curricula. There are many arguments for beginning with the calculus, *but with the growth of computer science and the need for more mathematics in the behavioral and social sciences there are more and more arguments for postponing the calculus courses.* [Italics added.]

2. *Statistics.* We argued above the importance of statistical reasoning to computer scientists as indeed it is important to all scientists. The second half of the junior year, following the calculus course, is the time for such a course. Except that the student would already be familiar with some discrete probability and its applications from the first year course, this one-semester course could be the one often taught by mathematics or statistics departments at the junior level.

4. Undergraduate Mathematics for Mathematics Students.

Now—and most important—we consider the implications of the foregoing for undergraduate education for mathematics majors. Specifically we ask: Can—and should—the two-year discrete mathematics curriculum outlined in Section 3 be considered as an alternative to, or even as a replacement for, the traditional two-year calculus (-linear algebra) sequence?

Two motivations for considering this question suggest themselves:

1. The year of discrete mathematics outlined in Section 3 for computer science students is also a sensible one—more sensible than a year of calculus—for social, behavioral, and management science undergraduates [22]. Therefore, if the two-year sequence proposed for computer science students is also appropriate for mathematics majors, this would ease the teaching problem considerably at smaller colleges where teaching resources are limited. (Such colleges seldom have engineering programs which need service from the mathematics department. If discrete mathematics becomes the standard for the first two years of mathematics, that leaves the problem of how to handle physical science students for whom—for physics majors, at least—calculus is still a necessity in the freshman year. There is, I think, no simple short-term solution to this problem other than to teach a service calculus course for such students.)
2. More compelling, however, is the possibility that introducing potential mathematics majors to college mathematics via discrete mathematics is to be preferred on its intrinsic merits to the calculus sequence.

What reasoning can be adduced to support the second motivation?

1. We note first that teaching discrete mathematics instead of calculus to potential mathematics majors in the first year is not such a drastic change as it may at first appear. Given that mathematics majors would take a two-year discrete mathematics sequence like that outlined in Section 3, the obvious junior-level mathematics for such students would be a year of calculus (of course!). In such a year, it should be possible to cover essentially as much as in three semesters of freshman and sophomore mathematics. Thus, at the end of the junior year, mathematics majors could be as well prepared for standard senior mathematics electives as they are now. Indeed, we should emphasize that a change to discrete mathematics as the normal freshman course would, in fact, not result in students receiving a bachelor's degree in mathematics being familiar with a very different set of topics in mathematics than they are now. What it would result in is students with a quite different orientation toward mathematics (and, perhaps, computer science also) than is customary now. The implications of this for graduate study in mathematics, and for the kinds of jobs bachelor's degree recipients might obtain, are considerable.
2. At least insofar as mathematics majors themselves are concerned, the foregoing is more negative about why no harm would be done than positive about why starting college mathematics with discrete mathematics might be directly advantageous. More specifically, we ask: Does the current thrust of mathematics itself support the changes proposed? This question needs consideration from a variety of points of view:
   (a) First, we may ask whether the current thrust of mathematics research supports a greater emphasis on discrete mathematics. This is not a question which can be answered definitively given the vast and variegated fabric of mathematics research today. What can be said—and probably all that can be said—is that the level of research on discrete mathematics, in particular, and in those areas of mathematics strongly affected by computers more generally (e.g., numerical analysis, number theory) has grown much faster than in more classical areas of analysis over the past two decades [**22**]. Therefore, it is, perhaps, fair to conclude that the proposals in this paper are not in obvious conflict with the current fabric of mathematics research.
   (b) Second, we consider the professional needs of bachelor's degree recipients in mathematics who seek jobs in business or industry (but not teaching—see (c) below). Until the 1950's such jobs as there were in business and industry for those with bachelor's degrees in mathematics were mainly in scientific industry (e.g., research laboratories) or in positions where the most prevalent task was statistical data analysis. In both areas the tools of classical analysis were of paramount importance.

   But the computer revolution has profoundly changed this. A considerable number of the kinds of jobs alluded to above still exist. However, added to these have been a large number of computer-related jobs where the preponderant mathematics tools are those of numerical analysis, a mixture of classical and discrete analysis. More recently, as the effective design and analysis of computer algorithms has been seen to be increasingly important in making effective use of computers, jobs with an essentially discrete mathematics flavor have become increasingly common.

   I know of no data which separate the industrial positions open to mathematics majors by the type of mathematics most needed in them. But I think it is safe to say that the kind of program proposed herein, which is a mixture of the discrete and continuous, would prepare the mathematics major for a wider variety of jobs than the classical curriculum in which discrete mathematics typically plays such a minor role. (Of course this problem could be remedied by increasing the discrete component in the current curriculum without materially changing the first two years of that curriculum. My contention in this context is only that, in terms of job preparation, such a curriculum is not to be preferred to the one proposed herein.)

   The above ignores the fairly large number of mathematics majors who are still hired into essentially programming jobs with a rather small mathematics component. I believe the discrete mathematics curriculum discussed here is clearly preferable to the classical curriculum for such people. But this argument should not be pressed too far because the phenomenon of mathematics students being hired into programming jobs is a transient one pending the time when computer science programs produce enough graduates to satisfy the demand for them.

(c) Third, and last, we consider the needs of those mathematics majors who embark on careers in teaching at the grammar and high school level. The important issue here, I think, is to prepare prospective teachers as well as possible for a teaching career spread over many years during which there may be considerable need for adaptation to new course material or approaches to teaching. Lack of adaptability has perhaps been one of the reasons that new approaches to teaching school mathematics have been less successful than was hoped. At least it seems to be the case that the general failure and weakness of "computer math" courses in the high schools has been due not only to a lack of familiarity with computers but also to a lack of knowledge of the mathematics—that is, discrete mathematics—most relevant to such a course at the high school level. Anyhow it seems fair to conclude that an undergraduate mathematics program better balanced between discrete and classical analysis than is now common would certainly be advantageous to prospective teachers of mathematics.

And we note that the growing importance of discrete mathematics is no transient phenomenon. It has been fueled by the growth of computers and computer science, a growth that will surely not end until computers are ubiquitous in all branches of science and technology and, more generally, in society broadly. One certain result of this is that computer science will become the largest, perhaps by far the largest, source of problems for mathematics and mathematicians.

It is also incumbent upon me to consider the argument against preceding calculus with discrete mathematics. At the heart of this argument is the belief that calculus is the natural transition course from high school to college mathematics. This is a difficult argument to deal with because of its totally qualitative nature. It is true that the experiments with "finite mathematics" for freshmen in the sixties, following the publication of the book by Kemeny et al. **[11]**, seem to have pretty well died out. But this is more likely to have been because these courses were not part of an integrated curriculum than because it was inherently unreasonable to teach such mathematics to freshmen. (It should also be noted that these finite mathematics courses were considerably different from what we have proposed in Section 3.) Too often, unfortunately, such courses came to be viewed as an easier way for non-mathematics or non-science majors to satisfy a mathematics requirement than by taking calculus.

My view on this argument is expressed in part by Hammer **[9]**, who advocates that "the finite calculus should be presented before the calculus" on the grounds that the latter is more easily assimilated by the student who understands the former. This point of view is similar to that of Gordon **[8]**.

Even so, however, there is the question of whether discrete mathematics serves to reinforce high school mathematics as well as calculus does. This is a matter of importance, because we have all experienced the phenomenon that "students always have to be taught what they should have learned in the preceding course" **[5]**. In terms of manipulative skills, particularly those of high school algebra, discrete mathematics requires more consistent use of such skills than calculus does. As to trigonometry I see little to choose between the two. Both require some recourse to trigonometric concepts and identities. Calculus courses do, of course, spend considerable amounts of time on differentiation and integration of trigonometric functions, more time than discrete mathematics courses would spend on summation or differencing of trigonometric functions, but little of this material involves much of *trigonometric* significance. On balance, I believe it is hard to make a strong case favoring either calculus or discrete mathematics as a reinforcer of high school mathematics.

Moreover, if discrete mathematics should come to be accepted as a (or *the*) standard freshman mathematics course, this would undoubtedly suggest some changes in the high school curriculum.

Here I would note only that this "top-down" approach to changes in the secondary and primary school curricula has much to recommend it in contrast to the "bottom-up" approach which has been prevalent in recent years.

Still, when all is said and done, the pro and con arguments rest on very slender foundations with little hard evidence to support either side. At the least, it seems clear to me that the *a priori* arguments against the teaching of discrete mathematics to freshmen can hardly be considered compelling enough to suggest that experiments in this area are not warranted.

5. Halfway Houses. The foregoing has purposely contrasted two models of undergraduate mathematics: (1) the prevalent, current one in which the first two years are concerned (mainly) with the calculus, and (2) a proposed model in which the first two years would be devoted to discrete mathematics. By so doing I have been able to contrast clearly the two approaches without muddying the waters with possible compromises. But not only are such compromises possible, they need consideration, which we briefly give them in this section.

Earlier sections of this paper have stressed the arguments in favor of discrete mathematics as the subject matter for the first two years of college mathematics. By considering some of the arguments against this approach, we shall be able to define most easily some possible halfway houses between the purely discrete and the purely continuous approaches.

1. Perhaps the most serious argument against the discrete approach has been alluded to earlier. This is that the physical sciences and engineering need calculus early to support their courses and curricula. With the discrete approach, therefore, you would limit the opportunity of students to postpone the choice of a major until the end of the sophomore year, because a student who embarked on the discrete sequence in the freshman year would find it difficult to major in a physical science or engineering. (The other side of this, of course, is that students who start with the calculus sequence may, in the future, find it hard to major in computer science or even management.)
2. Even for students not majoring in the physical sciences or engineering (e.g., computer science students), a lack of calculus in the first two years may cut them off from desirable courses. Consider, for example, the student who wishes to take a course in integrated circuit design, which must be preceded by an electronics and linear circuits course, which in turn needs physics, which needs calculus. (A response to this argument is that it's true but that it must be accepted that no undergraduate program which is educational as well as professional can possibly allow the student all options in the major.)

As the parenthetical comments indicate, I think there are rebuttals to the arguments above and probably also to other arguments that could be made against the discrete mathematics idea. But certainly there is also enough merit in these arguments to make some people who are sympathetic to my general thesis nevertheless look for less radical ways to implement it. Among the various possibilities, the most obvious and straightforward would be to divide the first two years between discrete mathematics and calculus. For example:

(a) Prospective computer science majors could take a year of discrete mathematics followed by a year of calculus. For some further thoughts on this subject, see [**17**].

(b) Prospective physical science and engineering majors could begin with calculus the first year and then take a year of discrete mathematics. In this context it should be noted that (1) the usual two years of calculus–linear algebra for students pursuing these areas is not required because *all* (or even perhaps a major fraction) of the material covered is of immediate professional need but, rather, in part at least, to develop mathematical sophistication; in this latter context one year of calculus and one year of discrete mathematics should serve just as well; and (2) because physical scientists and engineers should become sophisticated users of computers, the discrete mathematics would be of substantial use to them and would support a computer science sequence in the sophomore year.

(c) For mathematics majors, either of the above or a program of one semester each of calculus and discrete mathematics in each of the first two years could be considered. (It is interesting to note, as Professor A. W. Tucker has informed me, that a quarter-century ago the Committee on the Undergraduate Program (the forerunner of CUPM) proposed a freshman year of "universal mathematics" consisting of one semester of discrete mathematics and one semester of polynomial calculus and analytic geometry.)

(d) For those students unsure of their eventual major, one semester each of calculus and discrete mathematics during the first year could be satisfactory and would avoid the foreclosing of options.

Although, ideally, courses in both calculus and discrete mathematics would be taught differently depending upon whether or not students were having their first exposure to college mathematics; at a small college with limited teaching resources not too much need be lost if freshmen and sophomores were mixed in the same course.

Readers of this paper will surely imagine other possibilities than those described above.

6. Conclusion. Questions of education arouse strong feelings. For this reason, and because seldom, if ever, can propositions about education be proved or even strongly supported with evidence, they provoke strong statements. We have tried to avoid unequivocal statements unsupported by evidence in this paper and we hope our critics will also. Our essential proposition is simple and not immodest: *It is time to consider (i.e., try) an alternative to the standard undergraduate mathematics curriculum which would give discrete analysis an equivalent role to that now played by calculus in the first two years of the undergraduate curriculum.*

Whatever opinion anyone may have about the essential merit of this idea, it would be difficult to argue that it could do any harm to students on which it was tried. True, there are always initial difficulties with new courses and curricula. Texts are untried or unavailable. Experience is lacking on what is easy, what is hard, what seems to work, what doesn't, etc. But counterbalancing this is the enthusiasm typically present in people trying something in which they believe or, at least, for which they have high hopes, an enthusiasm which inevitably communicates itself to students and stimulates and involves them. Whether or not you judge the proposals in this paper to be a radical departure from the present curriculum, it seems unlikely that anyone, on balance, could believe that they would do educational or professional harm to the students.

Acknowledgement. I have had comments, criticism, and help from many mathematicians and computer scientists in the preparation of the paper [**22**] from which this paper has been drawn. Of these I am particularly grateful to Steven Garland, Richard Hamming, Dennis Martin, David Moursund, Gerald Myerson, Elliot Organick, Gerald Rising, Jacob Schwartz, Alan Tucker, and Albert Tucker.

References

1. ACM Curriculum Committee on Computer Science, An undergraduate program in computer science—preliminary recommendations, Comm. ACM, 8 (1965) 543-548.

2. ______, Curriculum 68—Recommendations for academic programs in computer science, Comm. ACM, 11, (1968) 151-197.

3. ______, Curriculum 78—Recommendations for the undergraduate program in computer science, Comm. ACM, 22 (1979) 147-166.

4. A. T. Berztiss, The why and how of discrete structures, SIGCSE Bulletin, 8 (1976) 22-25.

5. R. P. Boas, Jr., "If this be treason ...," this MONTHLY, 64 (1957) 247-249.

6. E. W. Dijkstra, Programming as a discipline of mathematical nature, this MONTHLY, 81 (1974) 608-612.

7. W. Givens, Implications of the digital computer for education in the mathematical sciences, Comm. ACM, 9 (1966) 664-666.

8. S. P. Gordon, A discrete approach to computer-oriented calculus, this MONTHLY, 86 (1979) 386-391.

9. P. C. Hammer, Computer science and mathematics, in Proceedings of the IFIP World Conference on Computer Education, vol. 1, IFIP, Amsterdam, 1970, pp. 53-55.

10. IEEE Computer Society Education Committee, A Curriculum in Computer Science and Engineering, IEEE Computer Society, 1977.

11. J. Kemeny, J. Snell, and G. Thompson, Introduction to Finite Mathematics, 1st ed., Prentice-Hall, 1956; 3rd ed., 1974.

12. D. E. Knuth, The Art of Computer Programming, vols. 1-3, Addison-Wesley, 1968, 1969, 1973.
13. ______, Computer science and its relation to mathematics, this MONTHLY, 81 (1974) 323–343.
14. L. S. Levy, Discrete Structures of Computer Science, John Wiley, 1980.
15. MAA Committee on the Undergraduate Program in Mathematics, A General Curriculum in Mathematics for Colleges, CUPM, 1965.
16. ______, Recommendations for an Undergraduate Program in Computational Mathematics, CUPM, 1971.
17. ______, Recommendations on Undergraduate Mathematics Programs in Computing, CUPM, 1972.
18. MAA, PRIME 80, Proceedings of a Conference on Prospects in Mathematics Education in the 1980s, 1978.
19. J. E. McKenna, Computers and experimentation in mathematics, this MONTHLY, 79 (1972) 294–295.
20. National Research Council, The Role of Applications in the Undergraduate Mathematics Curriculum, 1979.
21. H. O. Pollak, quoted in The Chronicle of Higher Education, 15 May 1978, p. 3.
22. A. Ralston, Computer Science, Mathematics and the Undergraduate Curricula in Both, Technical Report 161, Department of Computer Science, SUNY at Buffalo, 1980.
23. A. Ralston and M. Shaw, Curriculum 78—Is computer science really that unmathematical?, Comm. ACM, 23 (1980) 67–70.
24. J. B. Rosser, Mathematics courses in 1984, this MONTHLY, 79 (1972) 635–648.
25. D. F. Stanat and D. F. McAlister, Discrete Mathematics in Computer Science, Prentice-Hall, 1977.
26. O. E. Taulbee and S. D. Conte, Production and employment of Ph.D.'s in computer science, Comm. ACM, 20 (1977) 370–372.
27. J. P. Tremblay and R. Manohar, Discrete Mathematical Structures with Applications to Computer Science, McGraw-Hill, 1975.

MACHIAVELLI AND THE GALE-SHAPLEY ALGORITHM

L. E. DUBINS AND D. A. FREEDMAN
Department of Statistics, University of California, Berkeley, CA 94720

Summary. Gale and Shapley have an algorithm for assigning students to universities which gives each student the best university available in a stable system of assignments. The object here is to prove that students cannot improve their fate by lying about their preferences. Indeed, no coalition of students can simultaneously improve the lot of all its members if those outside the coalition state their true preferences.

1. Introduction. The object of this paper is to generalize the following result of Gale and Shapley [1]. For simplicity, suppose first that there are equal numbers of students, denoted

This paper describes joint research, but Freedman is mainly responsible for the exposition, including this footnote.

Before he completed his studies for his doctorate in mathematics at the University of Chicago in 1955, Lester Dubins had solved the problem of minimizing the length of a planar curve subject to boundary conditions and a curvature constraint and had settled a conjecture concerning an infinite game with incomplete information. His thesis, written under the guidance of Irving Segal, was of a more abstract nature and concerned a Radon-Nikodým derivative for Banach-space-valued random variables. After spending two years at the Carnegie Institute of Technology, he enjoyed three NSF postdoctoral fellowship years, two of which were spent at the Institute for Advanced Study at Princeton. Since 1960, he has been on the faculty of the University of California, Berkeley. His research, influenced by Leonard J. Savage and Bruno de Finetti, is largely concerned with the mathematics of (subjective and finitely additive) probability with excursions into other areas, particularly geometry, is oriented toward problems which are intuitive and concrete, and is frequently collaborative. He coauthored a book with L. J. Savage which was published by McGraw-Hill in 1965 under the title *How to Gamble if You Must* and reprinted by Dover in 1976 as *Inequalities for Stochastic Processes*.

David Freedman got his Ph.D. at Princeton in 1960; his thesis supervisor was William Feller. He spent a postdoctoral year at Imperial College, London. In 1961, he joined the faculty of the University of California, Berkeley. He has published many articles, some monographs, and an elementary textbook. His theoretical research is mainly in probability and statistics, on representation theorems like de Finetti's, on the sampling behavior of histograms, and on bootstrap methods for confidence intervals or significance levels. His applied work is in the legal area and in the validation of energy data and energy models.—*Editors*