Discrete Mathematics as A Transitional Course

Chung-Chih Li[1], Kishan Mehrotra[2], and Chu Jong[1]

[1] School of Information Technology
Illinois State University, Normal, IL 61790-5150, USA
{cli2, cjong}@ilstu.edu
[2] Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244, USA
mehrotra@syr.edu

Abstract. CS educators have paid a great deal of attention to Discrete Mathematics over the past several decades. Although there have been many suggestions for improving this course, it seems to us that the real purpose of Discrete Mathematics as a transitional course has been long forgotten. We believe that the most important objective of this course is to let students be familiar with the format and structure of rigorous mathematical arguments for their future study in CS. What subjects should be taught are also important because they are tools for us to effectively achieve the objective we just mentioned. Thus, we argue that complicated subjects should not be used in this transitional course because they will distract the student's attention from the underlying structure of the arguments that we want to emphasize. In this paper we start with some prevailing misconceptions in the effort of improving this course, and then we provide our solutions.

1 Introduction

Although the curriculum of CS programs since its creation in early 1960's has kept changing to reflect the rapid development of technology, there is one idea remains unchanged: Mathematics is indispensable in our discipline. When we sometimes depart from this motto, forceful rebounds such as Kelemen et al's [7] and Henderson's [6] are always promised to follow. Ralston's interesting editorial in [11] is a sarcastic yet accurate polemic to stress the importance of mathematics in CS education. With this consensus that mathematics is essential to the success of computer science, we have paid a great attention to, in particular, Discrete Mathematics mostly in attempts to identify the right topics for the course, to improve effective teaching, to motivate students, and to locate the right place for this course in the sequence for CS majors. Our earnest endeavor was witnessed by McMaster et al's count in [10] that "since 2000, over 30 papers on D[siscrete] M[athematics] have been presented at ACM-sponsored professional meeting". In those papers, idiosyncrasies aside, there are striking discrepancies on a variety of issues ranged from "Which department should teach the course?" to "Do we need Discrete Mathematics at all?" Some opinions are rather provocative and some even totally opposite to each other. In this paper we will address these issues with our opinions along the lines.

While we agree with Leblance et al's arguments in [8] that a full-year Discrete Mathematics course is a better approach, we have to emphasize that, through out this paper, the Discrete Mathematics course should be understood as the *One-Semester* course discussed in the SIGCSE Committee on the Implementation of a Discrete Mathematics Course [2]. If a full-year course is implemented, our arguments should apply to the first semester.

2 Wrong directions

Let us start with an anecdotal example in which we can find a common situation. In order to assess students' mathematical strength, we gave the following problem to a class of 20 students at the beginning of the Theory of Computation course in Spring 2007.

Problem 1: Let A and B be any two sets. Prove that, $(A \cup B) \subseteq (A \cap B)$ if and only if $A = B$.

Let TOC stand for Theory of Computation and DM for Discrete Mathematics here after. All students in the TOC class had taken DM as a prerequisite. Also, most of them were about to graduate in May 2007. Namely, they had leaned most of the advanced topics required in the CS program such as basic automata theory from the Concepts of Programming Language course, logics from the AI course, and Trees and Graphs from the Data Structures and Algorithms courses. Thus. we should not consider our students as "mathematical illiterates" at the beginning of the class. But surprisingly, none of them gave a satisfactory proof for this almost trivial question. A typical mistake is to prove by showing some examples or Venn Diagrams; even if the examples or Venn Diagrams are consistent with the property, they fail to show the understanding of "if and only if", the ability to generate abstract concepts, and the ability to construct mathematical arguments. However, they do show that the students have the concepts of sets, $\cup$, $\cap$, and $\subseteq$, but having those concepts is not the key point of the problem. One of our best students gave the following proof:

Wrong Proof 1: Let $x \in A \cup B$. Assume that $(A \cup B) \subseteq (A \cap B)$. Then $x \in A \cap B$. Since $x \in A$ & $x \in B$, $A = B$ implies $(A \cup B) \subseteq (A \cap B)$. $\square$

Clearly, the student may have some ideas of mathematical arguments (i.e., making an assumption and trying to reach some conclusion) and he may also understand set operations and relations, but the logic is completely broken in his presentation. Another fine student majoring in Math gave:

Wrong Proof 2: If $A = B$ then $A \cup B = A$ and $A \cap B = A$, by the properties of set union and intersection. Thus, $A \cup B = A \cap B$, and $A \cup B \subseteq A \cap B$. $|A \cup B| > |A| + |B|$ when $A \neq B$, $|A \cup B| < |A| + |B|$ when $A = B$. $\square$

If we tolerate the unclear properties of union and intersection the student refers to, the proof for the if-direction is correct. But the proof for the only-if-direction under the assumption, $A \cup B \subseteq A \cap B$, shows that the student has no clue to make a move but cooks up some wrong ideas about set cardinality he should have learned in his DM class.

The assessment results show that the existence of the DM course serves no purpose at all. It is very disappointing that even our best students do not learn anything from the course. Both instructors and students are frustrated in this course. Our chronicle frustration has brought us to re-examine the fundamental purpose of the course. We find that, many approaches suggested in the literature are either irrelevant or heading some wrong directions. We describe these approaches with our disagreements in the following subsections.

2.1 Which department should teach?

It doesn't really matter. Many institutes let the Math department teach the DM course but gradually many CS departments are trying to claim the course [8, **?**]. The trend in fact has been advanced by many non-academic issues that are out of the scope of this paper. To us, there is no convincing evidence to believe that the CS department can do a better job. Whenever CS faculty members complain about the outcomes of the course, the Math department is both willing and capable to adjust the contents of the course for CS's needs in order to keep the credit hours. Nevertheless, we are more or less sided with Professor Ralston's implication in [11] that the Math department may do a better job. By any chance it doesn't, that is because, in our opinion, the Math department is too willing to comply to CS's requests and make the course too computing or programming alike. We will explain why we denounce programming materials in the DM course in a moment.

2.2 CS students - weaker and less prepared?

Some instructors may think that students' poor performance in the DM class is the result of students' attitude and lack of mathematical background. But this oversimplifies the problem. On the contrary, we believe that students' poor performance is the result of the poorly designed course with a wrong agenda. Moreover, their motivation to learn is gradually vanished in the prolonged frustration.

According to their SAT scores, CS students are still relatively strong and prepared. They in general feel comfortable in the Calculus class, but why have they suffered so much in the DM class? We do not think that CS students have real difficulties to comprehend the concepts in DM. It is safe to say that all students in the TOC class already know the concepts of sets and their basic operations before the class. But knowing those concepts does not help them to construct satisfactory mathematical proofs. We observe that the way we handle mathematical arguments in DM is very different from the way we use in other lower level mathematics courses. For example, to solve a differential equation, students only have to find or memorize a suitable operational procedure and perform the procedure for the answer. But in DM, students have to make meaningful assumptions, pick up right axioms, re-use proven theorems and corollaries, apply inference rules, and organize their arguments in a logical way. They have tremendous difficulties to "orchestrate" all these small pieces like an unexperienced conductor in front of an over-sized orchestra for Mahler. We have overlooked this difficulty and failed to use this course to smooth student's "growing-pain" in such a transitional period.

2.3 More applications?

A common "wisdom" suggests that, we should connect the topics in DM to the real-world CS problems in such a way that students may be motivated to learn the topics. First of all, we do not think that education must be always harnessed to direct utilitarian interests, especially in higher education. Also, as we pointed out earlier, motivation is doomed by frustration, not merely by the disconnection from the real world. Compared to DM, Calculus may be even less useful in a programming-centered IT career, but students are more willing to learn Calculus with an attitude that "Calculus is ... great ... and educated people should know something about it" [11]. We rather want to cultivate students to view DM from an intellectual and scientific perspective.

Secondly, given the time constraints, it is not feasible to introduce any applications in depth unless the applications are trivial. But trivial applications apparently are not compelling and students may become even more skeptical about the significance of this course. In our opinion, students should pick up serious applications in their subsequent study; some will come immediately as early as in CS1. (In subsection 3.4 we advocate that the DM course should be taught as the first course in the CS program.)

2.4 More programming?

It seems to be a formidable trend that more and more CS educators are trying to increase programming works in the DM course (e.g., [3, **?**]). We believe that this trend is based on the following two wrong (at least, unproven) assertions:

1. Students can better learn abstract concepts through coding the concepts.
2. Self-satisfaction motivates leaning, and CS students in general are good or need to be good at programming.

In our opinion, the two assertions above are overstatements that may have misled us to wrong approaches to improving the DM course. For assertion 1, we would like to challenge that, "What exactly is the concept that students can't understand until they complete some programs to witness the concept?" We do not find any. For assertion 2, we must agree that self-satisfaction is an important factor in learning process, but self-satisfaction not necessarily facilitates effective learning. If a student is satisfied with some irrelevant achievement, the student may be heading a wrong direction without knowing it because he/she is rewarded that way. For example, McMaster et al [3] let programming assignments play a "substantial portion" in students' grades.

Consider our TOC class again. There is no need for students to write any programs in order to better understand the concepts of sets, union, intersection, and so on. On the other hand, there is no help for students from writing any programs when they don't know how to construct a proof for the assessment test. We do not evaluate students' mathematical maturity based on their ability to implement some programs performing a few basic set operations. We believe that most of, if not all of, the students in the TOC class have sufficient programming skills to do the job, but such skills have nothing to do with the ability to construct a simple mathematical proof.

Programming assignments in the DM course not only waste students' time mostly in the trial-and-error coding process, they also mislead students to believe that they can learn the subjects by coding some working programs. What particularly worries us is that, with very few exceptions such as the transitive-closure that is a classical example for a programming assignment, we must restrict the domain of interest to finite objects for almost all programming assignments along the lines of the topics in DM; otherwise, in general they are not computable. Here is a typical misconception we have encountered in the class of TOC. Many students consider that to test $A \subseteq B$ for any sets A and B is computable if we are given infinite memory to hold A and B because one can easily write a program to do the job; the only problem in the real world, students think, is that we don't have infinite memory. This is wrong. A Turing Machines has infinite memory but it can't test $A \subseteq B$ in general for a different reason. We believe, a programming-alike DM course may have enforced this misconception, among others.

As CS researchers and educators, we certainly do not oppose programming, but we oppose the idea of having programming works in the course of DM, because we have another agenda in this course. We rather want students use the same amount of time to exercise theorem proving and logical deduction with paper and pencil.

3 Our Pedagogical Philosophy To Discrete Mathematics

In the previous section, we have described some questions raised from the past efforts in improving the DM course. It seems to us that some efforts are irrelevant to the current problem and some approaches in fact compromise the integrity of the course. But what should we do otherwise? What do we expect from this course? In this section we try to provide our answers.

3.1 The Purpose of Discrete Mathematics

According to CC2001 [1], there are six core areas in mathematics required for CS programs to cover [2]:

DS1: Functions, Relations, and Sets

DS2: Basic Logic
DS3: Proof Techniques
DS4: Basic Counting
DS5: Graphs and Trees
DS6: Discrete Probability

All topics certainly have deep applications in computer science. However, we do not think that they should be all covered in a one-semester DM course. Time constraints are just one factor. More importantly, we do not think that any subjects that may be "interesting" to the CS faculty but new to the students should be taught in this course, because, as we believe, leaning new specific mathematical topics is not the purpose of the course. Topics in DS4, DS5, and DS6 should be covered in depth when they become necessary in other advanced courses with direct applications. Our reflection is that, any subjects that may be actually used in students' future study should be considered as the "interest" of the course, not the "principal". "Interest" is welcome, but we have to deposit enough "principal" first. Then, what is the principal of the course? Here is our proposition:

– Discrete Mathematics is an intermediate course to provide necessary mathematical *skills* for students to *transit* to more advanced studies.

We stress the words "*skills*" and "*transit*" in our statement. In other words, we want to transfer students from ones who understand mathematics with informal intuition and common senses to ones who can present mathematical and abstract objects in terms of precise definitions and express their relations in rigorous arguments. Thus, we consider the skills of managing mathematical arguments is the "principal" we want students to have after the course. Such skills are extremely important in our discipline. Even in writing an operational manual, for example, the author needs to define terminologies, domains of applications, and to organize logical sequences of operations and instructions for users.

3.2 Topics selection

As mentioned earlier, the most important objective of the DM course is to let students be familiar with the structure of rigorous mathematical arguments. Complicated concepts, however important to the discipline, are not suitable in this course because their complexity will distract the student's attention from the underlying structure of the arguments. We should use primitive concepts such as naive set theory, basic logical rules, easy discrete functions and relations, and so on as tools, not as targets, to train students for rigorous mathematical arguments and formal reasoning. We should let students re-gain the mathematical insights of those known concepts. Thus, topics in DS1$\sim$3 should be all covered in the course of DM; but we are not particularly interested in their applications, although they are everywhere in our discipline; they are just "interests". On the other hand, we consider topics in DS1$\sim$3 as perfect windows for students to deposit "principal", i.e., mathematical skills. They are selected because their basic concepts are very easy. Except Mathematical/Structural inductions that may be new to students, students already know most of the topics from their previous mathematical background. Thus, students need not spend too much time understanding the concepts before they can apply them in the arguments.

We prefer to teach the topics in the following order: (1) Sets, (2) Logic, (3) Mathematical/Structural Inductions, (4) Relations, and (5) Functions. For each topic, we start with a brief introduction that should cover all necessary concepts both in intuitive ways and precise definitions. Only one or two hours are needed to explain the basic concepts for each topic. For example, to explain the concept of bijective functions is just a matter of 10 minutes job. Then, we should use the rest of the class time to guide students how to use those terminologies, definitions, axioms, and assumptions in mathematical arguments.

Although we consider logic as the backbone of any rigorous arguments just like Alfred Tarski once put it: "It presupposes nothing but logic; that is, logic is the only preceding theory", we feel that the topic will get too involved at the end. Thus, we do not think it is appropriate to teach logic as the first topic in the course. We agree with Xing [12] that the naive set theory should come first, which also reflects the fact that the set theory was so created by George Cantor to lay a precise foundation for mathematics. Mathematical inductions are covered after logic as an application of Modus Ponens and logical inferences. The concept of relations is an extension of the concept of sets with extra properties. Likewise, the concept of functions is an extension of the concept of relations. In addition to mathematical/structural induction proofs, we have to explicitly emphasize on the patterns of mathematical proofs through out the semester, including contradiction, counterexamples, contrapositive, constructive vs. nonconstructive proofs, and so on. We do not mind if the subjects in DS4$\sim$6 are completely left out, no matter how important they are to the discipline. If students are well trained in DM, their mathematical skills should equip them to easily understand the presentations and techniques used in those advanced subjects.

The concepts involved in DS1$\sim$3 are not always trivial. From time to time, students will run into bizarre results that contradict their intuitions. For example, that the cardinalities of all countable infinite sets are all the same is a good example to convince students why we are motivated to go through those seemingly tedious formal arguments. I.e., only through such rigorousness, can we discover some fact beyond our intuition which is important to the discipline.

3.3 Mathematics as Musical Instruments

Practicing is the key to mastering. It is our believe that doing mathematics is just like playing musical instruments. In order to appreciate the music, one must at first listen to the music again and again. Consequently, if one wants to play and further master the musical instrument, one must practice again and again. However, a flawless performance of a violin maestro recorded in a CD has little help in training a beginner. Likewise, a student's mathematical skills cannot be developed by just listening to the instructor in the classroom and reading profound and well-written textbooks. To develop mathematical skills, one also needs to repeatedly practice easy arguments until one can use them like a built-in second nature. We consider this as the mathematical counterpart of the Suzuki lesson where we believe that a child's skill of playing any musical instrument is developed through repeatedly practicing the music piece by piece. Therefore, we have to ask students to do the same, even for trivial statements that can be understood intuitively. The practice problems need not be difficult. With only a few exceptions, all they need is to repeatedly use simple definitions and straightforward techniques in the arguments.

In fact, many complicated proofs can be decomposed into small pieces and each of them are simply the applications of easy techniques. Here is an astonishing example from the class of TOC. I.e., the *use* of Pumping Lemma. (We restrict our discussion to regular languages for simplicity.) We emphasize the word "use" but not the lemma itself, since we observe that students in TOC have little or no problem at all to understand the lemma itself, although it is a new and nontrivial concept to them. But only a very small portion of students can fully understand the logic behind in using the lemma to dispute a given language being regular. We believe that if students were well trained in the DM course, they should be able to formulate the lemma and work on its negation by themselves as follows. $P(L)$: L is regular. $Q(L)$: L satisfies the pumping property, where

$$\begin{aligned}&Q(L) : \exists m \forall \omega [|\omega| \geq m \rightarrow \exists x \exists y \exists x(\\ &\quad (\omega = xyz) \wedge (|xy| \leq m) \wedge (|y| > 0) \wedge \forall i (xy^i z \in L))].\end{aligned}$$

Thus, the pumping lemma states: $P(L) \rightarrow Q(L)$. Or equivalently, $\neg Q(L) \rightarrow \neg P(L)$. To prove $\neg P(L)$, we need to argue that $\neg Q(L)$ is true. With the applications of the following basic equivalences: $\neg \forall x p \equiv \exists x \neg p$, $\neg \exists x p \equiv \forall x \neg p$, $p \rightarrow q \equiv \neg q \rightarrow \neg p$, $\neg(p \rightarrow q) \equiv p \wedge \neg q$, $\neg(p \wedge q) \equiv \neg p \vee \neg q$, we can formulate $\neg Q(L)$ as follows:

$$\begin{aligned}&\neg Q(L) : \forall m \exists \omega [(|\omega| \geq m) \wedge \forall x \forall y \forall x(\\ &\quad \neg(\omega = xyz) \vee \neg(|xy| \leq m) \vee \neg(|y| > 0) \vee \exists i (xy^i z \notin L))].\end{aligned}$$

The remaining task is to make $\neg Q(L)$ true by finding appropriate values for ω and i quantified by $\exists$. We believe that the lack of practicing in the DM course is the reason for students' later struggle in this example.

3.4 Discrete Mathematics as the first course

Most CS programs make DM as a prerequisite to Algorithms and TOC. A common approach is to schedule the DM course in the second year of the CS program. By then, students should have taken CS1 and CS2, and probably are taking Data Structures and DM at the same time. Namely, DM is not considered as a prerequisite to CS1, CS2, and Data Structures. This is justifiable only if the course emphasizes on the topics listed in DS4$\sim$6. In this case, the topics listed in DS1$\sim$3 must be skipped or briefly skimmed. As a result, the course will then emphasize on the topics that are new and rather complicated to the students who not yet have enough mathematical skills to handle.

As we advocate in subsection 3.2 that the concepts used in the DM course should be easy enough for students to focus on the structure of arguments, we also think that reviewing those basic concepts and making them precise for students can benefit other courses including CS1 and CS2. We observe that illogical thinking will only lead to frustration, poor programming, and numerous revisits to fix a certain code before correct code is produced; and even after the correct code is achieved, the student will not be able to reason why the code is correct. Thus, we also advocate that the DM course should be taken in the first semester right after students get into the CS program. It is reasonable to ask students taking CS1 and DM in the same semester.

3.5 Rewrite textbooks in concise styles

There have been quite a lot of good textbooks for DM and we have to admit that we haven't really seen a bad one for *teachers*; but, unfortunately, not for most students. No matter how excellent we might think our choice is, we receive constant complaints from students about the textbook. When students struggle with the subjects, they like to request extra reading in addition to their already thick textbooks. (Nevertheless, this is an encouraging evidence that students do study hard in this course.) But all textbooks in our list are all the same in the sense that all of them are very wordy. Ironically, the more the authors try to explain, the further the students get lost in words. It seems to us that the reason students wrestling with the subjects in DM to certain extent is the language problem, not the subjects themselves. English is needed to explain concepts in intuition but the concepts of the subjects in DM at this level are almost trivial. Moreover, English as a natural language is rather clumsy in formal proofs and prone to mistakes and misunderstanding. We therefore believe that we should minimize the role of English in the process of training students' mathematical skills and let students try to comprehend the subjects and theorems in terms of mathematical languages, i.e., definitions, axioms, assumptions, inference rules, deductions, logical consequences, and so on. In fact, this opinion is not new to the community of DM education. For example, Epp [4] and Gries et al [5] advocate increasing the role of logic in teaching the subjects.

In following we highlight some key features we are looking for in an ideal textbook for the DM course.

1. Basic and intuitive concepts should not be unnecessarily explained in lengthy discussion. Instead, students should re-learn those concepts in terms of precise mathematical definitions and use them to express those known properties and operations. For example, the entire chapter of Sets should be based on a few definitions given at the beginning of the chapter.
2. Proofs of any theorems even if they are intuitively understandable should not be omitted, e.g., $A \subseteq A \cup B$. The purpose is to let students be familiar with the pattern of mathematical arguments and understand that many facts

that we have taken for granted are indeed mathematically provable based on definitions and logic.
3. In proofs, every single step should be explained by explicitly pointing out which definitions, previous proven theorems, or logical rules are used to arrive the concerned step. Phrases like "it is clear that ..." in most cases will just leave students wondering in the dark, and should be avoided.
4. Student should see enough amount of problems and their solutions for practicing.

To our knowledge, there is no DM textbook that meet the key features mentioned above. Thus, we decided to write our own experimental lecture notes to respond our own echo (see [9]). Our experimental notes had been used at Syracuse University for four semesters. In those semesters, we witnessed that students' performance and responses were more positive, and we received fewer complaints and requests for other supplementary reading material compared to other semesters when we used more conventional textbooks.

We believe that it will be worthwhile for the community to invest our effort in rewriting DM textbooks in a much more concise style for the educational purpose.

4 Conclusions

Like many educators, we have suffered from chronicle frustration in teaching DM and the awful gap between our expectation and actual outcomes of the course. We decided to re-examine the fundamental purpose of the course and how it is implemented in CS undergraduate programs. Many researches have been done over the decades, but we find that some suggestions are too ambitious by adding more topics into the course, while some are heading the wrong direction by turning this course into a programming course. We thus take a different way in hoping to fix the problem from the bottom. We give up complicated subjects and make DM a truly transitional course. We believe that students should build up their mathematical skills first in this course and learn new subjects with necessary mathematical maturity afterwards in their future study.

It is for sure that we do not have the capacity in this short paper to settle all issues. Nevertheless, we would like to urge educators in the community of CS to carefully re-examine the fundamental purpose of this course. We hope that our speculation can help us reach some consensus and lead to a more consistent philosophy in the syllabi of the DM course.

References

1. The ACM/IEEE Task Force Report on Computing Curriculum 2001- Computer Science Volume. `http://www.sigcse.org/cc2001/`.
2. The SIGCSE Committee on the Implementation of a Discrete Mathematics Course. `http://www.cs.grinnell.edu/~sigcse/sigcse-committees/discrete-math-charge.html`.

3. A. Decker and P. Ventura. We claim this class for computer science: A non-mathematician's discrete structures course. In *SIGCSE*, pages 100–104, Norfolk, Virginia, USA, 2007.
4. S. S. Epp. The role of logic in teaching proof. *American Mathematical Monthly*, 110(10):886–899, 2004.
5. D. Gries and F. B. Scheider. A new approach to teaching discrete mathematics. *Problems, Resources, and Issues in Mathematics Undergraduate Studies, PRIMU*, V(2), 1995.
6. P. B. Henderson. Mathematical reasoning in software engineering education. *CACM*, 46(9):45–50, 2003.
7. C. Kelemen, A. Tucker, P. Henderson, K. Bruce, and O. Astrachan. Has our curriculum become math-phobic (an american perspective). In *ITiCSE and SIGCSE/SIGCUE*, pages 132–135, Helsinki, Finland, 2000.
8. M. D. Leblanc and R. Leibowitz. Discrete partnership – a case for a full year of discrete math. In *SIGCSE*, pages 313–317, Houston, Texas, USA, 2006.
9. C.-C. Li and K. Mehrotra. *Problems on Discrete Mathematics, V 1 & 2*. `http://www.itk.ilstu.edu/ faculty/chungli/DIS300/ dis300v1.pdf (dis300v2.pdf)`.
10. K. McMaster, M. Anderson, and B. Rague. Discrete math with programming: Better together. In *SIGCSE*, pages 442–446, Covington, Kentucky, USA, 2007.
11. A. Ralston. Do we need any mathematics in computer science curricula? *SIGCSE Bulletin*, 37(2):6–9, 2005.
12. C.-C. Xing. Proof diagrams: A graphical tool for assisting set proofs. *CCSC*, 22(5):70–77, 2007.