Why Study Computability, Complexity, and Languages?

Avi Kak

kak@purdue.edu

Purdue University

August 23, 2021
11:59pm

Updated in August 2019 with a section on why studying computational complexity is just as important in this new age of Deep Learning

1. A LAMENT

Over the last couple of decades, as research in our universities has become ever more applied, it has become less likely for our graduate students to acquire the depth and the breadth in the fundamental knowledge that defines computer science and computer engineering.

It is not uncommon to see professors who believe that taking too many theory courses is a waste of time for their grad students. They also believe that it only delays the students working on actual research problems in order to start producing results for publications.

I believe that this state of affairs does not serve the long-term interests of the students. I also believe that the research lifespan of a new graduate student depends directly on both the depth and the breadth of the ideas in which the student has mastered the fundamentals. Additionally, the confidence with which a student can articulate ideas even at a purely applied level depends ultimately on the mastery the student has acquired over the relevant foundational ideas.

It is out of this concern regarding the extent to which our educational program meets the long-term (and, also the short-term) interests of the students — at least in that branch of knowledge that underlies computer science and computer engineering — that several years ago I created my course "*Computability, Complexity, and Languages*." This course is offered in the fall semester of every odd year.

This presentation is about why such a course can enrich a student in more ways than one.

2. WHAT MAKES THIS CLASS DIFFERENT FROM OTHER SIMILAR THEORY CLASSES TAUGHT ELSEWHERE

While presenting the fundamental notions of computability, complexity, and languages, I constantly strive to connect the theoretical discussion with what's important in today's computing.

For example, when we talk about Kleene's Theorem as a theoretical foundation for regular expressions, the homework problem that goes with that discussion asks a student to create a regular expression (in any programming language of his or her choice) that could be used to trap email messages containing several commonly used variants of certain kinds of spam.

My discussion on complexity is replete with examples from computer vision, network security, and other applications that show that even when a solution to a problem is logically feasible, it may be of little practical value if the algorithmic complexity of the solution strategy is exponential.

The discussion on the algorithms for solving hard combinatorial optimization problems highlights both the importance of the greedy algorithms and the fact that often you must first precondition the data before you can invoke a performance-guaranteed greedy solution strategy.

And so on ...

3. BUT ISN'T ALL THIS THEORY EXTREMELY BORING?

In this brave new world in which we are constantly amused by all kinds of devices that permeate our living and work spaces, anything that has the potential to bore us is likely to be thought of as an exercise in mental torture.

However, rest assured, this class is not boring. How can any class be boring that brings you the following stories:

- About the stupidity of highly honored people who sit on high councils of our government and who are convinced that someday robot intelligence will exceed human intelligence; [I'll tell you this story after talking about Godel numbers and after you have understood that all computing, no matter how complex, reduces to the execution of just three different types of instructions: one increment instruction, one decrement instruction, and one jump instruction.]

- About Stephen Cook who was denied tenure by the University of California, Berkeley, despite the fact that his work laid the foundations of the Theory of NP-Completeness. [As you will see, this theory plays a foundational role in our understanding of the combinatorial complexity of algorithms.]

- About Alan Turing, one of the most brilliant logicians of the last

century, who died under most unfortunate circumstances. [Comparing Turing to most modern-day researchers, what's interesting is that he never sought high honors and awards for himself despite the fact that the enormity of his contributions was immediately obvious. It is amazing how much time and energy modern-day researchers spend politicking and posturing in order to get themselves nominated for high honors and awards and for the 15-minutes of fame that was guaranteed to each one of us by the pop-art wizard Andy Warhol.]

- About Noam Chomsky, who is credited with having laid the foundations of modern linguistics and who is still active at the age of 84. [When I talk about the Chomsky Hierarchy and the Chomsky Normal Form in the part of the course devoted to languages, I'll tell you about how Chomsky's ideas dealt a big blow to Skinner's behaviorism in psychology. The behaviorists believed that, generally speaking, our minds are shaped by the reward structures in a society. Through his investigations in languages, Chomsky came to the conclusion that we are shaped primarily by our biology and that the fundamental structures in our minds are the same regardless of where we are born.]

- About oracles, my favorite being the Oracle of Delphi.

- And, finally, about the most taboo of the subjects in any program of education (outside of psychology and health) — the subject of sex. See page 22 of this document for what I mean.

4. TEXTBOOKS USED AND THE SCROLLS AT THE CLASS WEB SITE

Course Title: COMPUTABILITY, COMPLEXITY, AND LANGUAGES

Texts:

Computability, Complexity, and Languages
by Martin Davis, Ron Sigal, and Elaine Weyuker
Academic Press

Computers and Intractability, A Guide to the Theory of NP-Completeness
by M. R. Garey and D. S. Johnson
Publisher: W. H. Freeman

Random Graphs
by Bela Bollobas
Cambridge University Press

Class Web Site:

`https://engineering.purdue.edu/kak/ComputabilityComplexityLanguages/`

You'll find 44 lecture "scrolls" at this web site. Use the above URL or just google "ece664 purdue".

5. WHAT THIS CLASS IS ALL ABOUT

- At its core, this class is about complexity. You might ask: **What is complexity?** A humorous response — or, should I say, an imagistic response — to this question is depicted in the following picture that I saw at the web page of `https://it.inf.utfsm.cl/`:[1]

- **More seriously, this class is about the complexity of algorithmic solutions to problems.**

[1] Ordinarily, I seek permission before incorporating someone else's material in a document that bears my name as the author. But I am not able to figure out from the URL shown as to whom one should contact for permission to use this photo. It is also not possible to figure out whether this is a copyrighted photo.

- **Even more to the point, this class is about *how* to characterize the level of difficulty in solving problems amenable to algorithmic solutions.**

- The algorithmic solutions that we are interested in will usually involve some form of choice. The level of difficulty will deal with how much work is required to find the correct choices for the various arguments that go into a solution.

- Additionally, this class is about understanding computing at its most basic level. What does it mean to say that a function is computable? When is a function not computable? Are there cases when we are sure that a function is NOT computable?

6. MAIN PRACTICAL OBJECTIVE 1

- To demonstrate that even when we are sure that a problem can be solved algorithmically, we may **not** be able to actually solve the problem on any computer because of its **exponential complexity**.

- For example, it is possible to argue that all problems in computer vision should be solvable straightforwardly. Let's say we want to recognize a chair in a photograph of an indoor scene. Why can't we look for each part of the chair object in the photograph, make sure that the parts identified in the photograph are in the same relationship as the parts on a chair, and then declare whether or not we have found the chair in the photograph? Answer: **Because of combinatorial complexity.** [**Yes, I am aware of the fact that at this point the deep-learning aficionados would scream that the computational scenario I have painted for recognizing chairs in photos is not the way to go. See the next section for my counter arguments.**]

- The table on page 11 assumes that an algorithm involves n variables and that the time taken to solve the problem depends on n in the manner shown in the column with the heading *complexity*.

- For object recognition, n could be the number of primitives used to represent an object where a primitive is a feature that can be

easily recognized in a more or less view independent manner in a camera image.

- For another example, consider the problem that is known as "solving chess." Solving chess means finding the sequence of moves with which one of the players can always force a victory for every conceivable sequence of moves by the other player. Here n would be the number of pieces on a chess board times the number of positions to which the pieces could be moved at a given point in the play.

- On account of exponential complexity, guaranteed solutions to both these problems remain unimplementable in general. When n appears in the exponent in the leftmost column of the table on the next slide, we refer to that as "exponential complexity." The bottom two rows of the table are for exponentially complex problems.

- The table on the next page also assumes that each unit of computation takes 1 microsecond. In playing chess, a unit of computation could be the evaluation of the consequences of moving one piece to one candidate position on the board. In object recognition, a unit of computation could amount to finding the best image primitive that corresponds to a given model primitive.

	problem size n					
Complexity	10	20	30	40	50	60
n	.00001 seconds	.00002 seconds	.00003 seconds	.00004 seconds	.00005 seconds	.00006 seconds
n^2	.0001 seconds	.0004 seconds	.0009 seconds	.0016 seconds	.0025 seconds	.0036 seconds
n^3	.001 seconds	.008 seconds	.027 seconds	.064 seconds	.125 seconds	.216 seconds
n^5	.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 seconds	1.0 seconds	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 seconds	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Table 1: Assumption: one step of computation takes one microsecond. This table is from the book "Computers and Intractability" by Garey and Johnson

7. BUT DOESN'T DEEP LEARNING GET AROUND SOME OF THE COMBINATORIAL ISSUES?

- In this age of deep learning, someone might say that the computational scenarios painted in the previous section for, say, solving the problem of recognizing chairs in photos is not the way to go. Such folks would say: Let's collect a large number of photos of indoor (and outdoor) scenes with chairs in them; use, say, the Amazon Mechanical Turk to get crowd-sourced volunteers to label the chairs in the images with bounding boxes; feed the training data into an appropriate CNN; and, voilá, you have a pretty reliable system for recognizing chairs.

- The deep-learning based method for recognizing chairs could be referred to as an *appearance-based approach* to solving a computer vision problem. As long as the chairs to be recognized are in the training data, the system would work well. But what if a creative furniture designer were to come up with a novel design for a chair that was very different from all the chairs in the training data. Now we cannot be so sure if the deep-learning based method would give us reliable results. On the other hand, a component based algorithm might still work (assuming that combinatorial complexity was not a factor) since even a novel chair must have certain basic surfaces and some sort of a support structure to make it function like a chair — all of those considerations could be elements of a component based algorithm for

recognizing chairs.

- I should add that a deep-learning aficionado would respond to the issue raised above by saying let's collect images with the new chair in them; get them labeled the way we got the original training data labeled; and then update the previously trained CNN with transfer learning in which you use the new training data to update the parameters in just the uppermost couple of layers of the CNN. But what if you have an urgent need to start recognizing the new chair and you are unable to round up the additional training data needed?

- As indicated above, there are good points to be made on both sides of the issue. The phenomenal success of deep learning in cracking many difficult problems has definitely raised the bar for justifying the study of combinatorial complexity.

- One thing is certain: It would be foolish to assume that deep learning would solve all complex problems that require combinatorial search. Even in computer vision, let's say you are trying to use deep learning to solve an object recognition problem in a domain that the human annotators (of the sort you may recruit through the Amazon Mechanical Turk) know nothing about. In such a case, how would you generate the large amount of ground-truth data that is needed for the training of deep networks? This is likely to be the case for industrial and military objects. Here is another question along the same lines: Suppose you fell a large tree so that it is lying on the ground. Now consider a deep-

learningbased framework that easily recognizes trees becacuse they are likely to occur frequently in a large training dataset gleaned from the internet. Will such a network recognize a felled tree as a tree? Probably not. What if the ground is sloping? Will the probability of correct detection depend on the slope of the ground? Probably yes.

- Also remember that the number of problem domains where deep learning has proved to be wildly successful is relatively small. These are typically problems where you need to categorize something or perhaps calculate a regression function. As you will see in this class, there are many, many problems that call neither for categorization nor for regression. Their solutions consist of "How to" strategies for either solving a problem exactly or with a good approximation. Consider an instance of what is known as the bin-packing problem where you have a number of boxes of varying sizes and the objects, that are again of different sizes, to be packed in the boxes in such a way that the number ofboxes used is minimized. Solving this problem optimally requires a procedure that may involve ordering the boxes and the objects by their sizes and then placing the objects in the boxes using a prescribed formula. More on this later in class.

8. MAIN PRACTICAL OBJECTIVE 2

- So if exponential complexity is the bete noire of algorithm designers, how do we talk about it? How do we figure out if a given solution strategy possesses exponential complexity? Is all hope lost when a solution strategy turns out to possess exponential complexity? This is where the theory of NP-Completeness comes to our rescue. So one of the main objectives of this course is to introduce the student to the theory of NP-Completeness.

- In the Theory of NP-Completeness, the problems whose solutions possess exponential complexity are generally referred to as being NP-Complete.

9. MAIN PRACTICAL OBJECTIVE 3

- To show that there exist many important real-life problems that are even harder than the exponentially complex problems that come under the label NP-Complete. Many search and optimization problems fall in this category. [For a more precise phrasing of the first sentence here, replace "are even harder than" by "are at least just as hard as".]

- You will learn to differentiate between the characterizations NP-Complete and NP-Hard. [You will be surprised by how many folks out there do NOT have a precise sense of the difference between the two. While every NP-Complete problem is NP-Hard, the opposite is not true.]

- You will learn that oracles are important to understanding how to best compare the complexity of two different problems. And if you would allow me to amuse you a bit, you will find me appealing to the Oracle of Delphi to help you understand the notions of P^{NP}, NP^{NP}, etc., for characterizing the relative complexity of two different problems.

- While you will find me appealing to oracles, you have my ironclad guarantee that I shall NOT be speaking to shamans. **SO PLEASE DO NOT BE AFRAID! Rest assured that we at Purdue do NOT believe in shamanism.**

10. MAIN PRACTICAL OBJECTIVE 4

- To show that there exist many important **number problems** that appear to be exponentially complex at first sight but that nevertheless are amenable to fast (polynomial time) solutions for all practical purposes. The fast solutions employ algorithmic tools such as dynamic programming and in some cases exploit certain relationships that exist between the variables intrinsic to the problem and the variables that can be defined to help solve the problems.

- You will thus learn to differentiate between Strong-Sense NP-Completeness and Weak-Sense NP-Completeness.

11. MAIN PRACTICAL OBJECTIVE 5

- Regarding combinatorial optimization, you will learn to differentiate between those optimization problems that lend themselves to performance-guaranteed approximate solutions by greedy algorithms and those that do not.

- You will learn that for greedy algorithms to do their magic effectively, it is almost always necessary that you first *precondition* the input data.

- If you would allow me to express ideas a bit less seriously, at the least you will learn how to pack more efficiently for your next major airplane trip. [You will learn this through the greedy solution strategies for the Bin Packing (BINPACK) problem. The idea here is to pack your objects of different sizes in the smallest number of bags. By the way, BINPACK is important to VLSI folks.]

- **Now that the airlines are charging for your luggage by the bag, why would you deny yourself a chance to learn how to pack more efficiently in the future?**

12. MAIN PRACTICAL OBJECTIVE 6

- Even before we talk about the level of difficulty of solving a problem, we must talk about whether or not a problem is even solvable. The course will therefore start with the notions of computability and solvability.

- As you will discover, for some problems even to assume that a problem has a solution can lead to fundamental logical contradictions.

13. A NOTION CENTRAL TO THE STUDY OF COMPLEXITY

- At a conceptual level, all problems that require algorithmic solutions can be turned into *computation on languages*.

- Even problems that are ostensibly numerical in nature can be transformed into computation on languages.

- This allows models of computations on languages to serve as vehicles for developing the notions of complexity theory.

- It is for this reason that the first third of the course deals with models of computation, especially models of computations on languages.

14. SOME VERY EXCITING AND GENUINELY USEFUL THINGS YOU WILL LEARN AS WE STUDY COMPUTATIONS ON LANGUAGES

- You will, for example, learn the theory of regular expressions. **I simply cannot think of a more useful tool than regular expressions when it comes to processing textual data.**

- To help you make a connection between regular expressions as they are developed in theory and as they are actually used in programming languages, the course includes a programming homework involving regular expressions. On the surface, the famous Kleene's Theorem that leads to regular expressions seems not to have much in common with how the regular expressions are actually used in programming languages. The goal of this homework is to see how this theorem translates into the syntax used for regular expressions in practice.

- You will learn the importance of finite automata as a programming paradigm for very fast recognition of certain languages.

- You will learn why some basic problems in string processing that appear to be so solvable can be theoretically demonstrated to

have no solutions at all.

- You will also learn that while to a lay person string processing appears to be different from number processing, the two are really the same from the standpoint of fundamental reckoning.

- Finally, for those of you who will eventually specialize in the systems side of programming, the language portion of this class will give you a solid theoretical foundation for your future courses on compilers. **All modern compilers are based on two abstractions: regular expressions for lexical analysis of the input and context-free grammars for parsing the tokens produced by lexical analysis.** The class includes a programming homework on specifying a context-free grammar using the EBNF (Extended Backus Naur Form) notation and then parsing a simple text file using the grammar. We will use the open-source JavaCC to generate the parser from the grammar.

15. WE WILL START WITH NUMBERS

- In order to develop these ideas incrementally and in a way that you would enjoy, we will start with models of computation on numbers. *It seems like we have some sort of a subconscious association between computation and working with numbers. So any time you want to talk about models of computation, it is good to start with numbers.*

- We will answer the following question: What is the most basic manner in which practically all computations on numbers could be represented?

- You will be introduced to a programming language whose instruction set has only three statements in it. You will learn that practically every problem in number processing can be programmed in this ridiculously simple language. Talk about a reduced instruction set computer! (This is the very famous Church's Thesis.)

- That will set us up to talk about models of computation on languages.

16. INCLUDING RANDOM GRAPHS

- My long-term goal in this course is to also include some discussion on the complexity related to the probabilistic representation of knowledge for solving combinatorial problems and the related probabilistic algorithms.

- As a step in that direction, the last part of this course will include some material from the book "Random Graphs" by Bollobas.

- In this part of the course, I will examine issues such as: Whereas graph isomorphism is, in general, exponentially complex, random graph isomorphism can be solved in low-order polynomial time. What implications does that have for solving real-life graph isomorphism and subgraph isomorphism problems?

17. FINALLY, HERE ARE SOME OTHER REASONS FOR WHY YOU WILL ENJOY THIS CLASS

As I have pointed out in my talk "*Why Robots Will Never Have Sex*," there is no shortage of highly influential people who are convinced that computers will soon exhibit intelligence that will surpass that of humans and that the human race had better watch out for the day when computers will take over. Several of these people have received the highest awards that the country can bestow on them and they sit in the high councils of the governments and industry. **This course will show you how stupid these so-called famous people really are.**

You will learn two theoretical concepts in this course that have a direct bearing on the issue of machine intelligence verses human intelligence: these are the concepts of pairing functions and Gödel numbers. A pairing function shows us how you can create a 1-1 mapping between a two-dimensional discretized space and a one-dimensional line. Similarly, the Gödel numbers show us how you can create a 1-1 mapping between any n-dimensional discretized space and a one-dimensional line.

What that means is that dimensionality higher than one is not a fundamental property of ostensibly multidimensional spaces if they are discretized.

Digital computers, by their very nature, can only do their calculations in discretized spaces. So, from a strictly theoretical viewpoint, even when a computer is doing its thing in an n-dimensional space, the same results could in principle be obtained through calculations that only need to work in a one-dimensional space. (Considering that Gödel numbers tend to be large since they involve products of powers of primes, it may not be practical to carry out such dimensionality reduction in a given context. But whether something is practical today is not the issue in this discussion.)

On the other hand, human imagination works in a continuum. We have no trouble imagining objects with shapes and motions that are continuous in time and space. The human mind has no trouble imagining the infinitesimal and the infinite. After all, that is how calculus was born.

Given this fundamental dichotomy between the abilities of the human mind and what the machines are capable of, for anyone to claim that they are convinced that the machines will soon take over the world is more an indication of the bankruptcy of their own minds than of the power of computers.

18. SYLLABUS

Each of the lectures listed below is hyperlinked to a "scroll" at the web site for this class

1. Why Study Computability, Complexity, and Languages

2. Sets, Tuples, Cartesian Products, Partial and Total Functions, Predicates, Quantifiers, Proofs

PART 1: COMPUTABILITY

3. The Minimalist (but all powerful) Programming Language S

4. Partially Computable Functions

5. Primitive Recursive Functions

6. Sixteen Building-Block PRC Functions

7. Minimalization, Pairing Functions, and Godel Numbers

8. Church's Thesis and Unsolvability of the Halting Problem

9. A Universal Program that can Execute Any Program and the Concept of Recursive Enumerability

10. Extending Computability Theory to Calculations on Strings, and The Minimalist (but all powerful) Language S_n for String Processing

11. The Post-Turing Language T for String Processing and the Equivalence of the Languages S, S_n, and T

12. Turing Machines

13. Languages Accepted by a Turing Machine, Recursive Sets vs. Recursively Enumerable Sets

14. String Computations with Productions and Processes

15. Post's Correspondence Problem

16. Phrase Structure Grammars and Context Sensitive Grammars

PART 2: GRAMMARS AND AUTOMATA

17. Finite Automata and Regular Languages

18. Properties of Regular Languages

19. Regular Expressions and the Pumping Lemma

20. Context-Free Languages

21. Regular Languages, the Chomsky Normal Form, and Bar-Hillel's Pumping Lemma

22. Properties of Context-Free Languages

23. Bracket Languages and the Pushdown Automata

PART 3: THE THEORY OF NP-COMPLETENESS

24. Computational Complexity: Some Basic Definitions

25. Defining P and NP

26. A More Precise Definition of Class NP

27. Polynomial Transformability of Problems in Class NP

28. Definition of NP-Complete and the SATISFIABILITY Problem

29. The six basic NP-Complete Problems and NP-completeness proof for 3SAT

30. NP-Completeness Proofs for 3DM, VERTEX COVER, CLIQUE, and PARTITION

31. NP-Completeness by Restriction: Subgraph Isomorphism (SGI), KNAPSACK, Multiprocessor Scheduling (MS), and Bounded-Degree Spanning Tree (BDST)

32. NP-Completeness Proofs by Local Replacement: Squencing within Intervals (SQUINT), and Partition into Triangles (PIT)

33. NP-Completeness Proofs by Component Design: Minimum Tardiness Sequencing (MINTARD)

34. Reasoning about the Complexity of the Subproblems of a given NP-Complete Problem (Ex.: Graph 3-Colorability)

35. Strong-sense and Weak-sense NP-Completeness for Number Problems

36. Extending the Complexity Theory to Search Problems

37. Generalizing Polynomial Transformability to Turing Reducibility and the Definition of NP-Hard

38. Complements and Optimization Versions of the NP-Complete Decision Problems

39. Approximation Algorithms for Combinatorial Optimization: Bin Packing Optimization (BINPACK), Traveling Salesman (Restricted) Optimization (TSRO), etc.

40. Nonapproximable and Easily Approximable NP-Hard Optimization Problems: Graph Coloring Optimization Problem (GRAPHCOLO), Knapsack Optimization Problem (KNAPSACKO), etc.

41. The Classes NPI, Co-NP, P^NP and NP^NP of Problems and the Class #P of Enumeration Problems

42. Storage Considerations: The Classes PSPACE, PSPACE-Complete, and PSPACE-hard

PART 4: RANDOM GRAPHS

43. Random Graphs --- An Introduction

44. The Degree Sequence and the Polynomial-Time Graph Isomorphism for Random Graphs

---