CS888: Introduction to Profession and Communication Skills
-- Theoretical CS

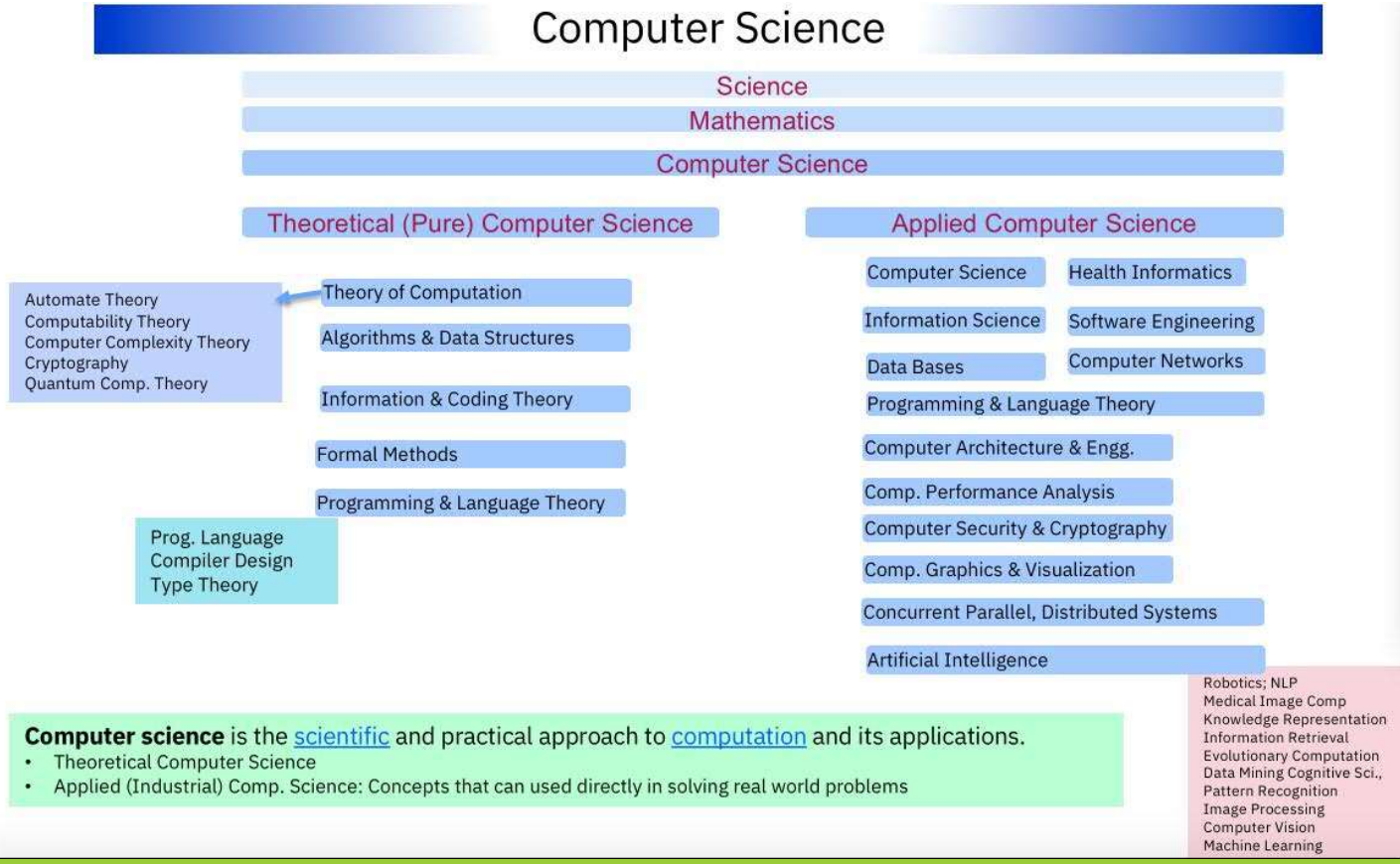
NITIN SAXENA (NITIN@CSE)

[*WITH HELP FROM INTERNET SOURCES]

AUG 21, 23, 28, 30; SEP 4, 6; 2024

Is Theory Maths?

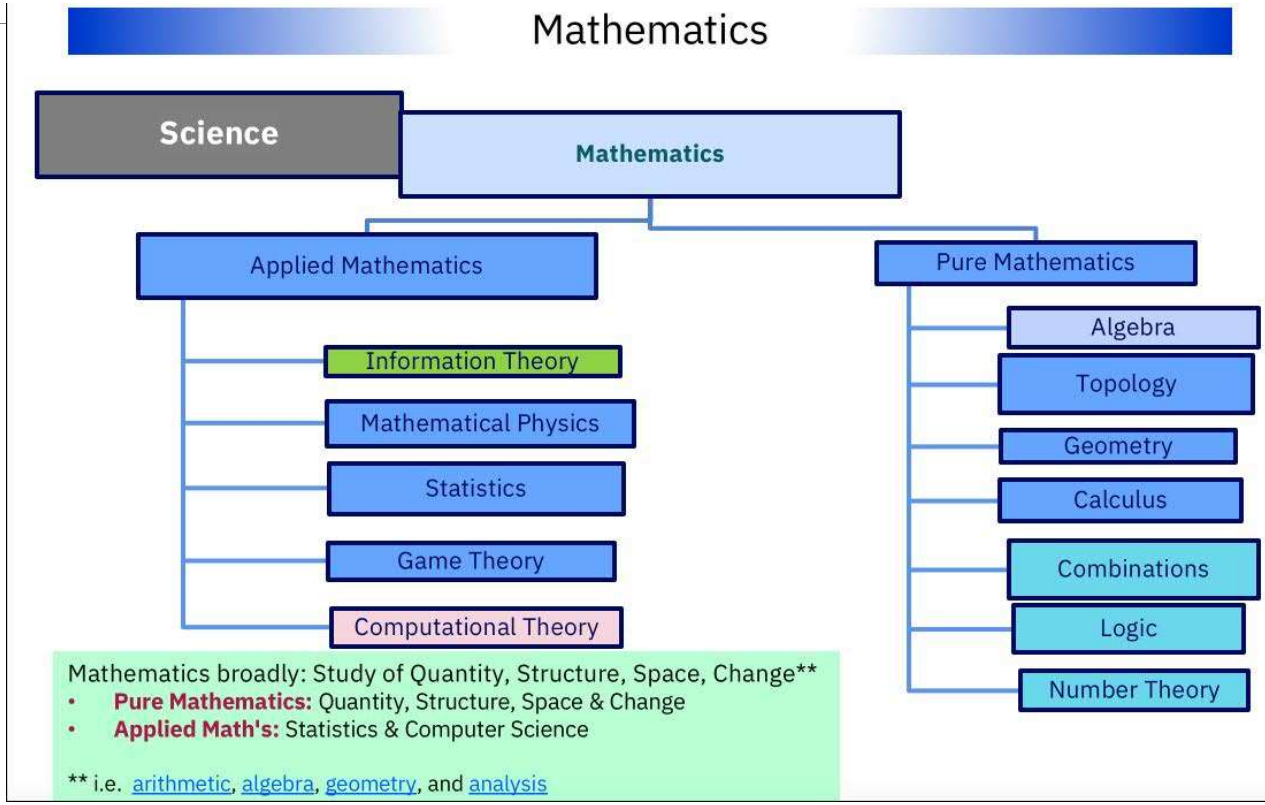
- Theory = Theoretical Computer Science
- Theory is the art of
  - designing algorithms
  - guessing their efficacy
  - proving your guess
- No theory without compute
- Maths is a larger framework
  - doesn't need compute
  - needs aesthetics!
  - different examples based on scientific areas



Is Maths Theory?

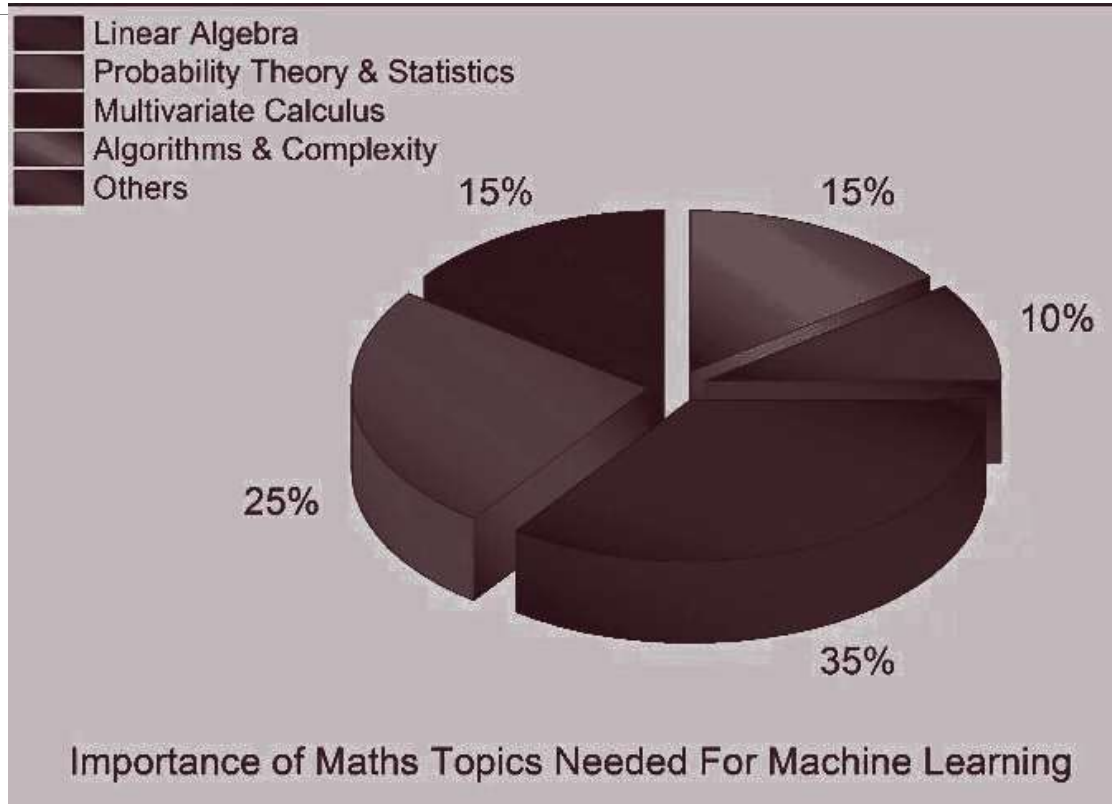
- Maths *may* interact with computation
  - but it largely exists on it own
  - = abstract or pure
- Mathematicians don't need computers
- Our motivation is *all* from computing
- Maths textbooks have a very precise and rigorous notation
- Theory manuscripts are written in a notation that has a computational feel!

- Theory enriches Maths and vice versa!



Is Maths necessary in CS?

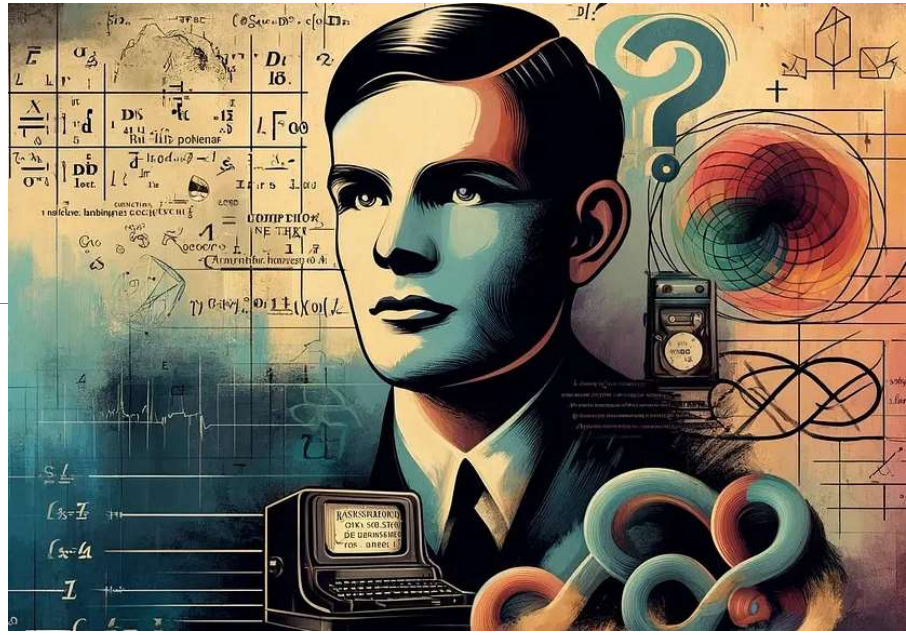
- Linear Algebra → ML, Graphics
- Probability → Machine Learning
- Calculus → Deep learning, Complexity
- Number theory → Cryptography, Error-correcting code
- Geometry → Vision, Motion planning
- Algebra + Combinatorics → Communication, Storage
- Game theory → Matching, Auction, Trading, Recommendation system



Importance of Maths Topics Needed For Machine Learning

Theory statements

- The Divine Dilemma: Can God Solve the Halting Problem?
- The halting problem is undecidable.
- Theorem: The halting problem is undecidable.
- Theorem [Turing, 1936]: The halting problem is undecidable.
- Define the concepts:
  - *Halting problem* is a decision problem that asks if a computer program will terminate, or run forever, given a description of the program and an input.
  - *Undecidable problem* is a decision problem that cannot be solved by an algorithm.



- Algorithm is a set of rules that must be followed when solving a particular problem.
- Define a *Turing Machine*
  - as a symbolic description "of rules that must be followed".
- Definition: An *algorithm* is a Turing machine.

Theory statements in Systems

- Mathematically *model* the real-life problem.
- Allocate T tasks to P processors.
- Define the concepts:
  - *Feasible*
  - *Optimal*
  - *Hard, Easy*
  - *NP-hard*

The Journal of Real-Time Systems, 4, 145–165 (1992)

Allocating Hard Real-Time Tasks: An NP-Hard Problem Made Easy*

K.W. TINDELL, A. BURNS AND A.J. WELLINGS
Real Time Systems Research Group, Department of Computer Science, University of York, England

Abstract. A distributed hard real time system can be composed from a number of communicating tasks. One of the difficulties with building such systems is the problem of where to place the tasks. In general there are P^T ways of allocating T tasks to P processors, and the problem of finding an optimal feasible allocation (where all tasks meet physical and timing constraints) is known to be NP-Hard. This paper describes an approach to solving the task allocation problem using a technique known as *simulated annealing*. It also defines a distributed hard real-time architecture and presents new analysis which enables timing requirements to be guaranteed.

1. Introduction

Building real-time systems on distributed architectures presents engineers with a number of challenging problems. One issue is that of scheduling the communication media, another concerns the allocation of software components to the available processing resources. Distributed systems typically consist of a mixture of periodic and sporadic tasks, each with an associated deadline and possibly precedence constraints. Failure to meet the deadlines of critical tasks may lead to a catastrophic failure of the system, and consequently off-line analysis of allocation and processor scheduling is required to guarantee task deadlines.

In general, the three activities of task allocation, processor scheduling and network scheduling are all NP-hard problems (Burns 1991). This has led to a view that they should be considered separately. Unfortunately, it is often not possible to obtain optimal solutions (or even feasible solutions) if the three activities are treated in isolation. For example, allocating a task T to a processor P will increase the computational load on P, but may reduce the load on the communications media (if T communicates with tasks on P), hence the response time of the communication media is reduced, allowing communications deadlines elsewhere in the system to be met. The tradeoffs can become very complex as the hard real-time architecture becomes more expressive; a simple and scalable approach is required.

Previous approaches to solving the task allocation problem have mostly concentrated on graph theoretic algorithms [for example (Chen and Yur 1990; Chu and Lan 1987)] or heuristics [for example (Bannister and Trevedi 1983; Houstis 1990)]. Most have tried to maximize system throughput (i.e., minimize the computational and communication resource requirements for tasks in the system), often by reducing *bottlenecks*, resulting in allocations which may or may not be schedulable. However, these approaches do not take a global view; they rely on the observation that fast systems (i.e., ones which maximize system

*This work was supported in part by British Aerospace (Commercial Aircraft) Ltd, and the UK Department of Trade and Industry.

Assignment 6

https://hello.iitk.ac.in/

deadline 12pm (end of class)