CS120: Intro. to Algorithms and their Limitations Anshu & Hesterberg

Lecture 25: Conclusions

Harvard SEAS - Fall 2023 *2023-12-05*

1 Announcements

- Fill out Q evaluation
- Review sessions
- Problem set 9 duedate
- Problem sets 8/9 revision videos
- Extra OH

Recommended Reading:

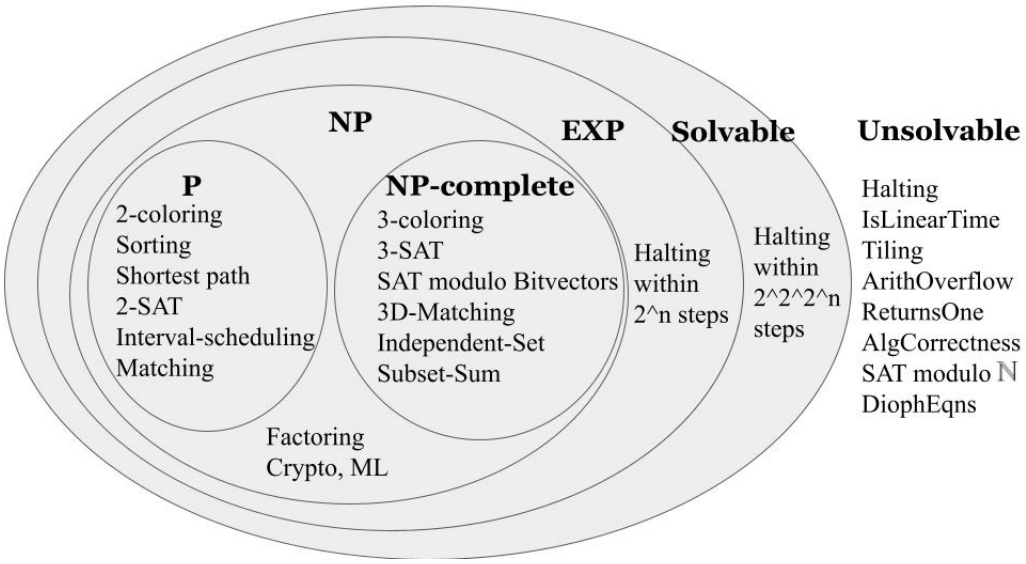
- Roughgarden IV, Epilogue
- MacCormick, Chapter 18

2 An Algorithmicist's Workflow

When confronted with a real-world algorithmic problem (like Web Search, Interval Scheduling, Deduplication, Census Data Releases, Google Maps, Kidney Exchange, Lyber, "Magic Maze", Register Allocation/Map Coloring, Programming Team, ArithmeticOverflow, ArtemisParty, ...), you can tackle it using the skills from cs120 (and future classes) by looping through the following steps:

1. Mathematically model
    - What kinds of **input data**? Key-value pairs (CS165), graphs and combinatorial structures (AM107, Math 152), logical formulas (Math 141, Phil 144), programs (CS 152, CS 153)
    - What kind of **output/queries/objective function**?
    - Is it a **one-shot** computational problem or **data structure** problem? How many queries will be made? Does it need dynamic updating or can it be static? (CS 124)
    - How much **details**/structure to incorporate? We have seen many cases where incorporating a restriction makes the problem easier. For instance, SAT vs 2-SAT, Matching vs Bip. Matching, directed vs undirected graphs, unlimited universe vs bounded universe for sorting, edge weights vs unweighted graphs...)
    - Social context and **ethiCS**? (CS108, CS126, CS136, CS208, CS226, CS238)
    - What is the right **computational model** and relevant **resources** to consider? (CS61, CS121, CS152, CS165)

2. Look for related problems (in class, in the literature, on the web) and try to obtain an algorithm by **reduction to** another problem:



This diagram shows a separation between P and NP-complete problems, but there is also the possibility that P = NP = NP-complete, in which case the entire NP circle collapses into one class.

3. Try to obtain an algorithm by **reduction to** other problems

   - IntervalScheduling, AreaOfConvexPolygon reducing to Sorting
   - Rotating Paths reducing to Shortest Paths
   - Bipartite Matching reducing to Rotating/Alternating Paths
   - 2-SAT reducing to Shortest Paths

4. Try to apply **algorithmic techniques**

   - BFS
   - Greedy
   - Exhaustive Search
   - Divide and Combine (MergeSort, Randomized Selection)
   - Data Structures (Sorted Arrays, BSTs, Hash Tables)
   - Many more in CS124, CS182, AM121, CS165: dynamic programming, DFS, priority queues, randomized algorithms, approximation algorithms, local search, linear/integer/-convex programming

     ⋮

5. Try to show hardness/unsolvability by **reduction from** other problems

   - NP-hardness for combinatorial search problems

- Unsolvability for program verification and logic problems
- More in CS121, CS127, CS221, Phil144

6. And/or settle for weaker guarantees
   - Find more structure to incorporate in your modelling, or
   - Use heuristics that only run efficiently or are only correct on some inputs
   - Settle for approximation algorithms
   - Randomized algorithms like Monte Carlo algorithms
   - More in CS124, CS152, CS182, CS223, AM121

3 Other Takeaways

- Universality
  - (Extended) Church-Turing Thesis
  - Turing-equivalent models
  - (Word-)RAM, Compilers
  - Universal programs
  - Technology-independence
  - More in CS121, CS61, CS152, CS153, Phil 144
- Rigorous mathematical theory
  - Computation and computational problems can be precisely modelled
  - Algorithms can be rigorously analyzed for correctness and efficiency
  - Some problems, including important ones we want to solve, can be *proven* to be impossible or intractable to solve
- There is much we don't know!
  - Can Sorting be done in $O(n)$ time? Matching in $O(n+m)$?
  - P vs. NP
  - Are polynomial-time randomized algorithms more powerful than deterministic polynomial-time algorithms? What about polynomial-time quantum algorithms?
  - Is secure cryptography possible?

4 CS120 Learning Outcomes

From the Syllabus: "By the end of the course, we hope that you will all have the following skills:

- To mathematically abstract computational problems and models of computation
- To design and implement algorithms using a toolkit of algorithmic techniques

- To recognize and formalize inherent limitations of computation
- To rigorously analyze algorithms and their limitations via mathematical proof
- To appreciate the technology-independent mathematical theory of computation as an intellectual endeavor as well as its relationship with the practice of computing."

5 Where to Learn More

- Theory of Computation seminar: http://toc.seas.harvard.edu/
- Many other CS courses, especially x2x. Look at grad (2xx) courses too. (CS120 may serve as a sufficient substitute for CS121/CS124 in some of them.)
- Read more of our textbooks (Roughgarden, MacCormick, CLRS, and the references therein)
- Come talk to us in office hours!