# Berkeley Math Circle – Advanced

Tuesday, November $8^{th}$

## Introduction to NP-Completeness

Matt Anderson
Simons Institute / Union College

## 1 Warm-up: Compositeness

Recall the definitions of *prime* and *composite*.

**Definition 1.** *Let $x$ be a natural number.*

- *$x$ is* prime *if $x > 1$ and the only natural numbers that evenly divide $x$ are 1 and $x$.*
- *$x$ is* composite *if $x$ is not prime.*

For example, the numbers $\{2, 7, 23\}$ are prime and the numbers $\{1, 4, 42, 99\}$ are composite.

**Question 2.** *What method (algorithm) do you use to determine whether a number is prime or composite?*

**Question 3.** *Suppose someone told you that a particular number $x$ was composite. How might they quickly demonstrate that to you?*

**Question 4.** *Suppose someone told you that a particular number $x$ was prime. How might they quickly demonstrate that to you?*

We can describe primes and composites as *languages* over the natural numbers.

**Definition 5.** *Let $L \subseteq U$ be sets. We say that $L$ is a* language *over $U$. The elements of $L$ are called* YES instances *of the language and the elements of $U - L$ are called* NO instances *of the language.*

For example, Primes $= \{x \mid x \text{ is prime}\}$ is a language over the universe of natural numbers. The language Composites is the complement in the natural numbers of the language Primes.

## 2 Algorithms and Efficiency

An algorithm is a method for completing a task.

**Question 6.** *Have you used any algorithms today before coming to Berkeley Math Circle?*

We are interested in thinking about the following kind of algorithms.

**Definition 7.** *Let $L$ be a language over $U$. A* decision algorithm *$\mathcal{A}$ for $L$ is a method that takes an element $x$ of $U$ as input and outputs whether $x$ is a YES instance or NO instance of $L$. We say $\mathcal{A}$ decides $L$ in this case.*

Ideally we want algorithms to complete their task while using the fewest resources possible.

**Question 8.** *What are some resources that an algorithm might use?*

One standard resource to consider is time. That is, how much time did it take to complete the task? We assume that basic operations like addition, multiplication, comparison of number take one "unit" of time to perform a single operation. In order to make running time comparable across different instances we compare running time as a function of the input instance size. For an instance $x \in U$ we denote its size as $n = |x|$.

**Question 9.** *For* Composites *what is a reasonable definition of instance size?*

We use big-oh notation to describe the running time of algorithms. Big-oh notation captures the dominant, long-term limit running time of an algorithm as a function of its input size.

**Definition 10.** *An algorithm $\mathcal{A}$ for the language $L$* runs in time $O(f(n))$ *if there is a constant $c$ such that for all $x \in U$, $\mathcal{A}$ uses at most $c \cdot f(|x|)$ time before producing an output.*

**Question 11.** *What are the strongest running time bounds you can place on our algorithm for* Composites*? How does this depend on the definition of input size for this problem.*

We can define the notion of an "efficient" algorithm.

**Definition 12.** *An algorithm $\mathcal{A}$ is* efficient *if there is a polynomial $n^k$ such that $\mathcal{A}$ runs in time $O(n^k)$.*

**Question 13.** *Is our algorithm for* Composites *efficient?*

## 3 P and NP

We can classify languages based on how difficult they are to solve.

**Definition 14 (P).** P *denotes the set of all languages that can be decided efficiently.*

**Definition 15 (NP).** NP *denotes the set of all languages that whose YES instances can be verified efficiently. This means there is a verification algorithm that takes as input an instance and a "proof" that the instance is a YES instance and checks whether or not the proof is correct.*

Thus Composites $\in$ NP. (It also turns out that Composites $\in$ P, however, it is much much more difficult to argue this).

**Question 16.** *It is the case that* P $\subseteq$ NP. *Why?*

P and NP are two examples of *complexity classes.*

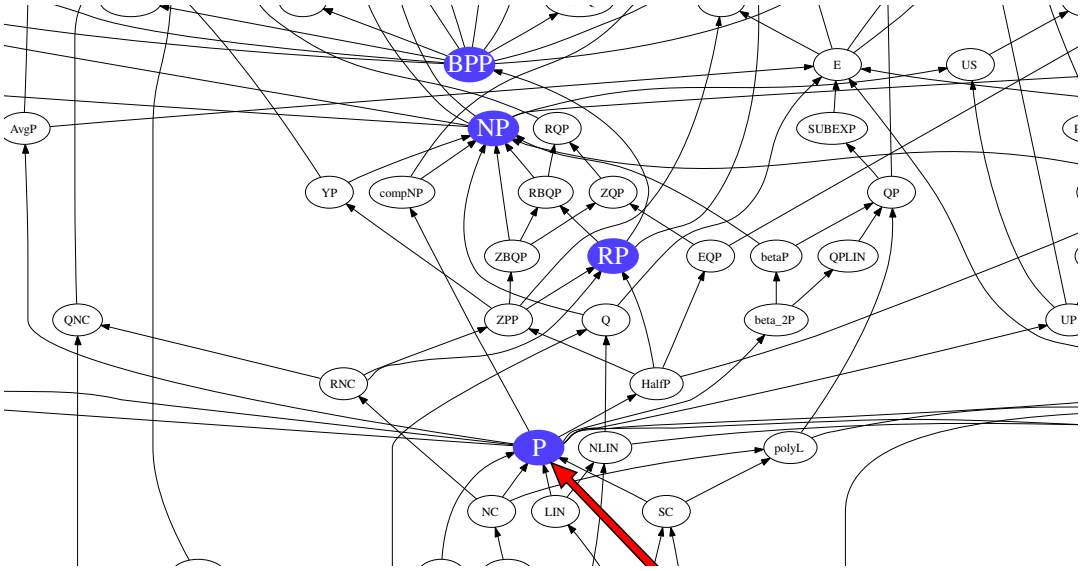


Figure 1: There are many others complexity classes, a collection of them is maintained by the complexity zoo – `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`.

## 4 Examples of NP Problems

There are many interesting problems in NP. Some of the most intuitive ones are graph problems.

A graph $G$ is called *Hamiltonian* if $G$ has a cycle which visits all vertices exactly once. The language HamCyc is the set of graphs that have Hamiltonian cycles.

**Question 17.** HamCyc $\in$ NP. *Why? What demonstrates that a graph is a YES instance of* HamCyc*?*

A related problem is the traveling salesperson problem. Let $G$ be a graph where each edge is labeled with a positive integer cost. An instance of the traveling salesperson problem (TSP) is a weighted graph $G$ and an integer $k$, and such an instance is a YES instance if there is a Hamiltonian cycle of $G$ where the sum of the weights on the cycle is at most $k$.

**Question 18.** *What is the relative difficulty of deciding* HamCyc *and* TSP*?*

**Question 19.** *Argue that deciding* TSP *is at least as hard as deciding* HamCyc*.*

## 5 Reductions

The answer to the previous question of something called a *reduction*. Informally a reduction is an algorithm that takes an instance of one problem and transforms it into an instance of another problem where membership is preserved.

**Definition 20.** *A language $L \subseteq U$* reduces to *a language $L' \subseteq U'$ if there is a function $\Gamma : U \to U'$ where for all $x \in U$, $x \in L$ iff $\Gamma(x) \in L'$. We denote this by $L \leq L'$.*

Reductions say that one problem can be decided by first transforming into another problem and then solving the second problem. For example, we have HamCyc reduces to TSP, that is, HamCyc is at least as hard to decide as TSP. Reductions are only interesting if they are efficiently computable, otherwise the reduction could decide the original problem without need to decide the second problem. All the reductions we will discuss have this property.

There is a subset of problems in NP, called NP-*complete problems*. These are problems that every other problem in NP can be efficiently reduced to.

**Definition 21.** *A language $L$ is* NP-complete *if*

1. $L \in$ NP *and*
2. *for all* $L' \in$ NP, $L' \leq L$.

It turns out that both HamCyc and TSP are NP-complete.

**Question 22.** *Does* TSP $\leq$ HamCyc*?*

**Question 23.** *Suppose that* HamCyc $\in$ P *what does it say about whether* TSP $\in$ P*?*

## 6 P $\neq$ NP?

The study of NP-completeness was initiated to try to understand the structure of languages in NP and their relationship with P. This resulted in the following widely accepted conjecture.

**Conjecture 24.** P $\neq$ NP

**Question 25.** *Resolve this conjecture.*

## 7 Other NP-complete Problems

Clique Given a graph $G$ and a natural number $k$ determine whether there exists a subset $S$ of the vertices of $G$ where $|S| = k$ and **all** pairs of vertices in $S$ are adjacent.

Independent Set Given a graph $G$ and a natural number $k$ determine whether there exists a subset $S$ of the vertices of $G$ where $|S| = k$ and **no** pairs of vertices in $S$ are adjacent.

Vertex Cover Given a graph $G$ and a natural number $k$ determine whether there exists a subset $S$ of the vertices of $G$ where $|S| = k$ and every edge of $G$ is incident to at least one vertex in $S$.

Coloring Given a graph $G$ and a natural number $k$ determine whether there exists a coloring of $G$ where each vertex is assigned one of $k$ colors and no adjacent vertices are assigned the same color.

Partition Given a set of $n$ numbers $N = \{a_1, a_2, \ldots, a_n\}$ determine whether there is a subset $S \subset N$ where the sum of elements in $S$ is exactly half the sum of the elements in $N$.

Subset Sum Given a set of $n$ numbers $N = \{a_1, a_2, \ldots, a_n\}$ and a number $k$ determine whether there is a subset $S \subset N$ where the sum of elements in $S$ is exactly $k$.

Knapsack Given a set of $n$ items, $N = (v_1, w_1), \ldots, (v_n, w_n)$, where $v_i$ and $w_i$ are positive numbers giving the value and weight of the $i^{th}$ item, and target value $v$ and weight $w$ determine whether there is a subset of items whose weights sum to at most $w$ and whose values sum to exactly $v$.

## 8 Reduction Exercises

**Question 26.** *Show* Independent Set $\leq$ Clique.

**Question 27.** *Show* Partition $\leq$ Subset Sum.

**Question 28.** *Show* Subset Sum $\leq$ Partition.

**Question 29.** *Show* Partition $\leq$ Knapsack.

**Question 30.** *Show* Independent Set $\leq$ Vertex Cover.

**Question 31.** *Show that when* $k = 2$*, the* Coloring *is in* P.

## 9 Satisfiability

There is one problem that is central to connecting many NP-complete problems together, and it captures the essence of efficient verification.

A Boolean formula is like an arithmetic formula but instead of using the operations addition (+), subtraction (-), multiplication ($\cdot$), and division (/) on variable whose values are numbers it uses the operations AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) on variables whose values are true and false. Recall that $a \wedge b$ is true only when both $a$ and $b$ are true, and is false otherwise, and that $a \vee b$ is true when at least one of $a$ and $b$ are true and is false otherwise. Below is an example of a Boolean formula.

$$f(a, b, c) = a \wedge (b \vee \neg c)$$

This formula is true under the assignment $a = T, b = F$, and $c = F$, and false when $a = F, b = T$, and $c = F$. A Boolean formula is called *satisfiable* when there is some assignment to its variables that makes it true. When there are no assignments to the variables that make a formula true it is otherwise called *unsatisfiable*. The language SAT is the set of all satisfiable Boolean formula.

**Question 32.** SAT $\in$ NP. *Why?*

**Theorem 33 (Cook-Levin).** SAT *is* NP*-complete.*

One simple kind of Boolean formula are 3-Conjunctive Normal Form formula (3CNF). A 3CNF formula is an AND (conjunction) of a number of ORs of three variables or their negations. More formally, every 3CNF formula $f$ can be written as

$$f = \wedge_{i=1}^{m} \vee_{j=1}^{3} l_{ij}$$

where each $l_{ij}$ is either a variable or its negation. The language of satisfiability restricted to 3CNF formula is called 3SAT, that is, it consists of all 3CNF formulas that are satisfiable.

**Question 34.** *Show* 3SAT $\leq$ SAT*, and* SAT $\leq$ 3SAT*. Hint: One direction is easy and one direction is not.*

**Question 35.** *Show* 3SAT $\leq$ Independent Set.

**Question 36.** *Show* 3SAT $\leq$ Coloring *for* $k = 3$.

**Question 37.** *Show* 3SAT $\leq$ HamCyc.

**Question 38.** *Show* 3SAT $\leq$ Partition.

# 10 Further Study

Languages often become easier to decide we relax their requirements in some way.

**Question 39.** *For the problems discussed today we wanted algorithms that were* exact*, that is they always quickly give the correct answer for every input. What if we did not require this exact form of correctness. What are some alternative definitions of "correctness" that we could use?*

**Question 40.** *Some of the problems we discussed today took secondary parameters. What if we assume these of the parameters where constant, would they then have efficient algorithms then? (Hint: Think about* Vertex Cover *and* Coloring*.)*

**Question 41.** *Are there any other relaxations of problems that might make them easier to solve?*

## 10.1 Further Reading

Here are a few references were you could learn more about this topic.

Popular Science

- Lance Fortnow. *The Golden Ticket: P, NP, and the Search for the Impossible*, Princeton University Press, 2013.
- Timothy Lanzone, director. *Travelling Salesman.* Fretboard Pictures, 2012.

Undergraduate Textbooks

- Michael Sisper. *Introduction to the Theory of Computation.* Cengage Learning, 1996, 2006, 2013.
- Michael Garey and David Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co. New York, NY, USA, 1979.