Complexity of Problems

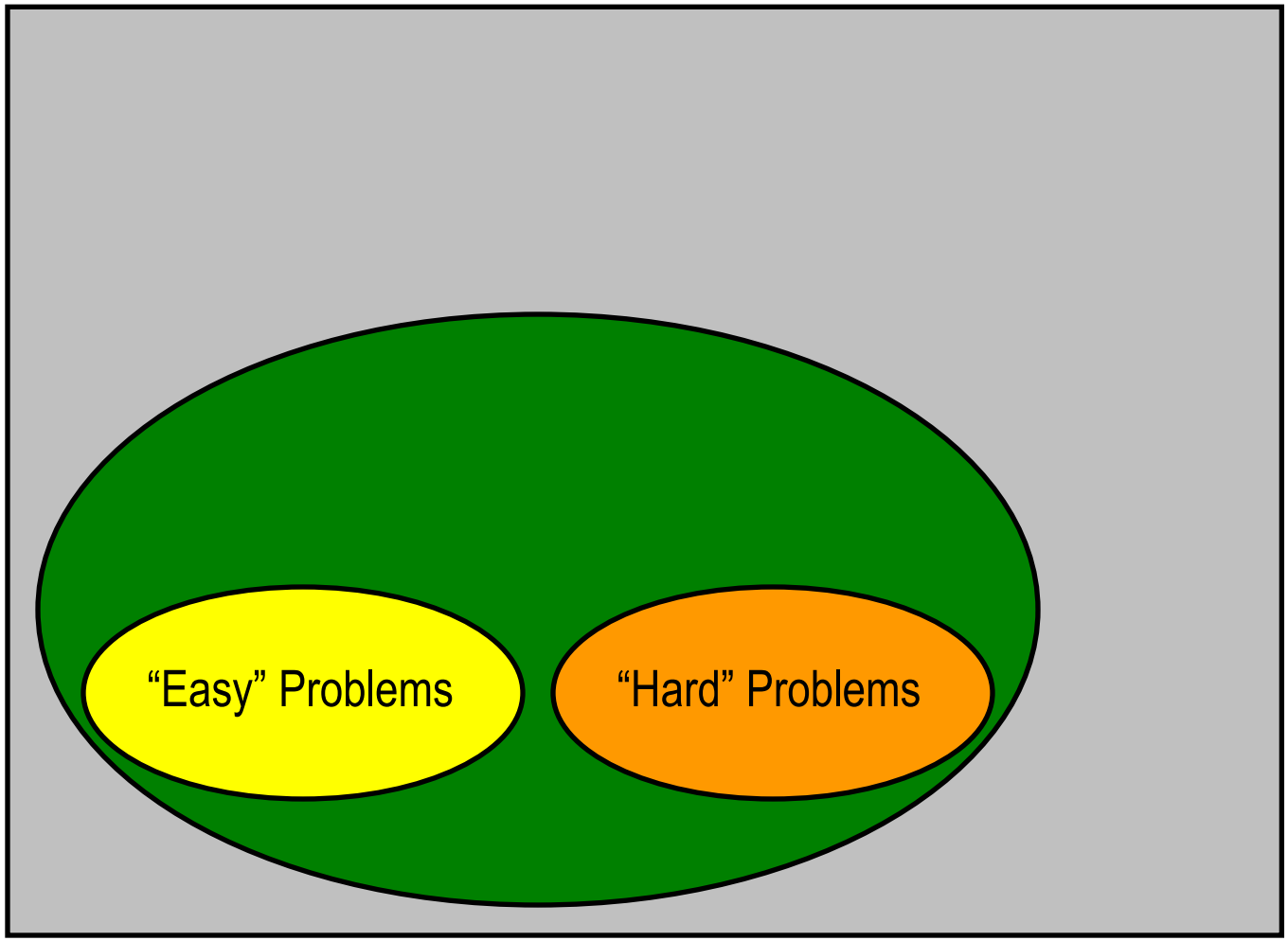
Classifying Problems

- We have seen that decision problems (and their associated languages) can be classified into decidable and undecidable.

  This result was obtained by Turing and others in the 1930' — before the invention of computers.

- After the invention of computers, it became clear that it would be useful to classify decidable problems, to distinguish harder problems from easier problems

  This led to the development of computational complexity theory in the 1960's and 1970's


"Easy" Problems
"Hard" Problems

Definitions

- Algorithms that have a polynomial complexity are algorithms whose run time is $\Theta(n^d)$ for some positive integer d.
- Algorithms that have an exponential complexity are algorithms whose run time is $\Theta(c^n)$ for some constant c greater than 1.

It is clear that as n grows large, the exponential complexity algorithms may become impractical. (i.e. It may take many years of continuous computation in order to find the correct answer).

Complexity Classes

This will be an informal discussion of the basics of this terminology.

Much of the terminology refers to decision problems. A *decision problem* can be described as a problem that requires a "yes" or "no" answer. The class P refers to the set of all decision problems for which polynomial-time algorithms exist. (P stands for polynomial)

- P Class
- NP Class
- NP hard
- NP complete

Complexity Measures

Every decidable problem has a set of algorithms (= TMs) that solve it.

What property of this set of algorithms could we measure to classify the problem?

- The difficulty of constructing such an algorithm?
- The length of the shortest possible algorithm?

  (Giving a static complexity measure[2].)
- The efficiency of the most efficient possible algorithm?

  (Giving a dynamic complexity measure.)

Dynamic Complexity Measures

A dynamic complexity measure is a numerical function that measures the maximum resources used by an algorithm to compute the answer to a given instance

To define a dynamic complexity measure we have to define for each possible algorithm M a numerical function φ_M on the same inputs

Time Complexity

The most critical computational resource is often time, so the most useful complexity measure is often time complexity

If we take Turing Machine as our model of computation, then we can give a precise measure of the time resources used by a computation

Definition The time complexity of a Turing Machine T is the function Time_T such that $\text{Time}_T(x)$ is the number of steps taken by the computation $T(x)$

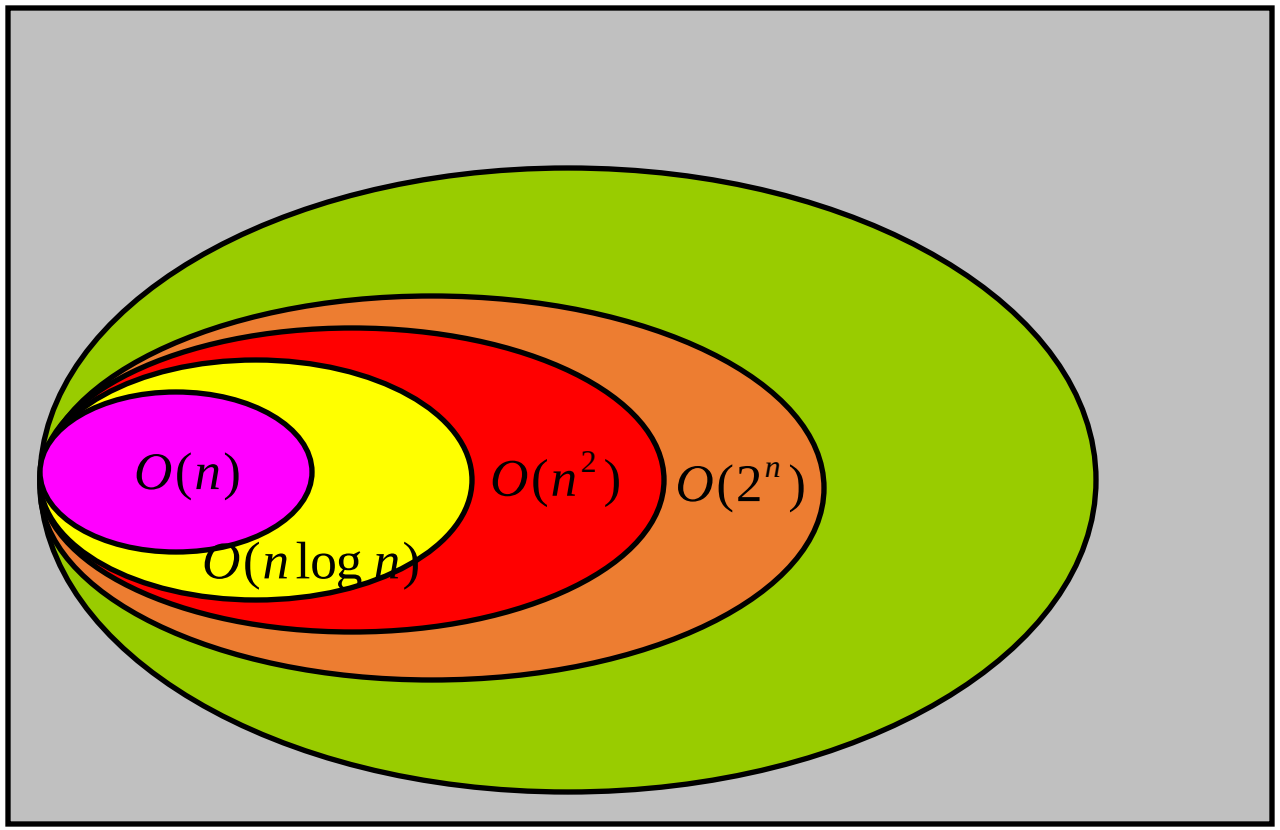
Space Complexity

Another important computational resource is amount of "memory" used by an algorithm, that is space. The corresponding complexity measure is space complexity

As with time, if we take Turing Machine as our model of computation, then we can easily give a measure of the space resources used by a computation

Definition The space complexity of a Turing Machine T is the function Space_T such that $\text{Space}_T(x)$ is the number of distinct tape cells visited during the computation $T(x)$

Examples



P Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a “yes” or “no” answer) that can be solved by a deterministic machine in polynomial time.

Features:

- The solution to P problems is easy to find.
- P is often a class of computational problems that are solvable and tractable.
- Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

This class contains many natural problems like:

- **Calculating the greatest common divisor.**
- **Finding a maximum matching.**
- **Decision versions of linear programming.**

NP Class

- The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Features:

- The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.
- Problems of NP can be verified by a Turing machine in polynomial time

Let us consider an example to better understand the NP class. Suppose there is a company having a total of 1000 employees having unique employee IDs. Assume that there are 200 rooms available for them. A selection of 200 employees must be paired together, but the CEO of the company has the data of some employees who can't work in the same room due to some personal reasons.

This is an example of an NP problem. Since it is easy to check if the given choice of 200 employees proposed by a coworker is satisfactory or not i.e. no pair taken from the coworker list appears on the list given by the CEO. But generating such a list from scratch seems to be so hard as to be completely impractical.

It indicates that if someone can provide us with the solution to the problem, we can find the correct and incorrect pair in polynomial time. Thus for the NP class problem, the answer is possible, which can be calculated in polynomial time.

This class contains many problems that one would like to be able to solve effectively:

- Boolean Satisfiability Problem (SAT).
- Hamiltonian Path Problem.
- Graph coloring.

NP-hard class

- An NP-hard problem is at least as hard as the hardest problem in NP and it is the class of the problems such that every problem in NP reduces to NP-hard.

Features:

- All NP-hard problems are not in NP.
- It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.
- A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Some of the examples of problems in Np-hard are:

- No Hamiltonian cycle.

NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP.

Features:

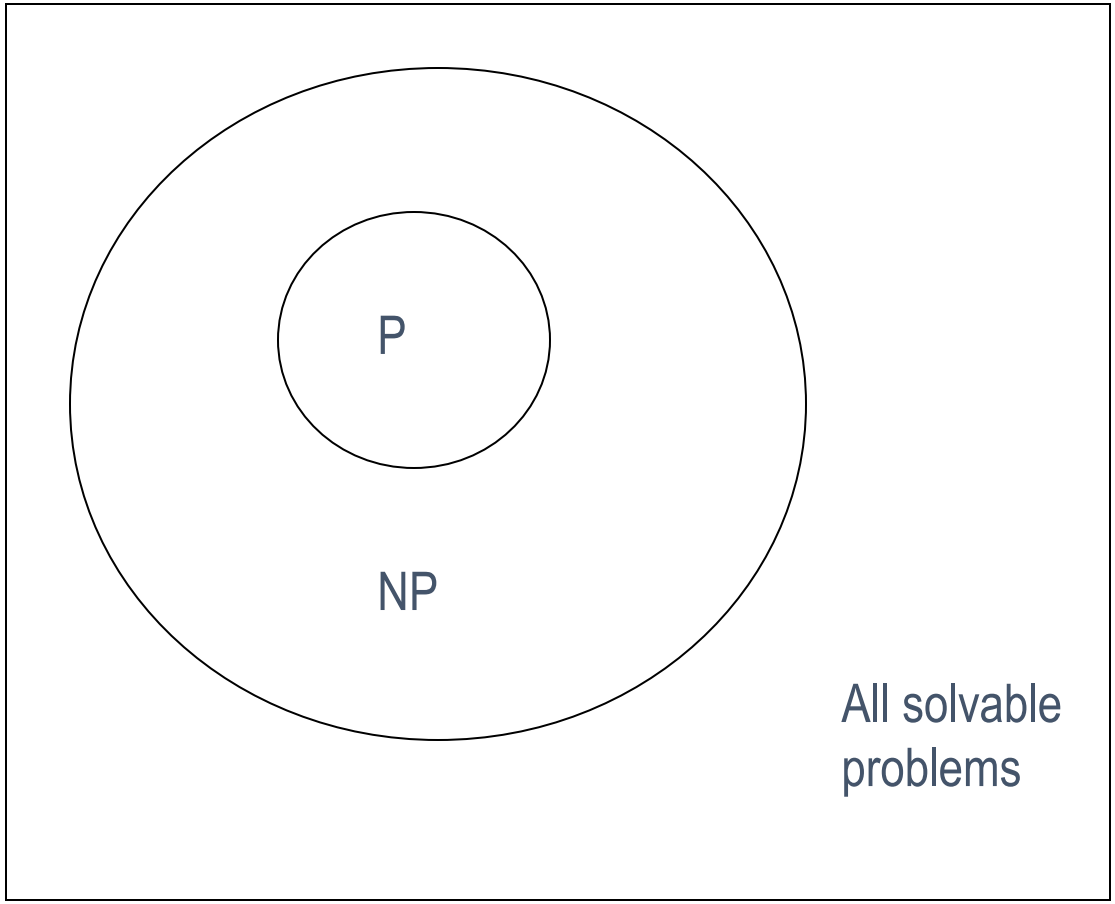
- NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.
- If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem in polynomial time.

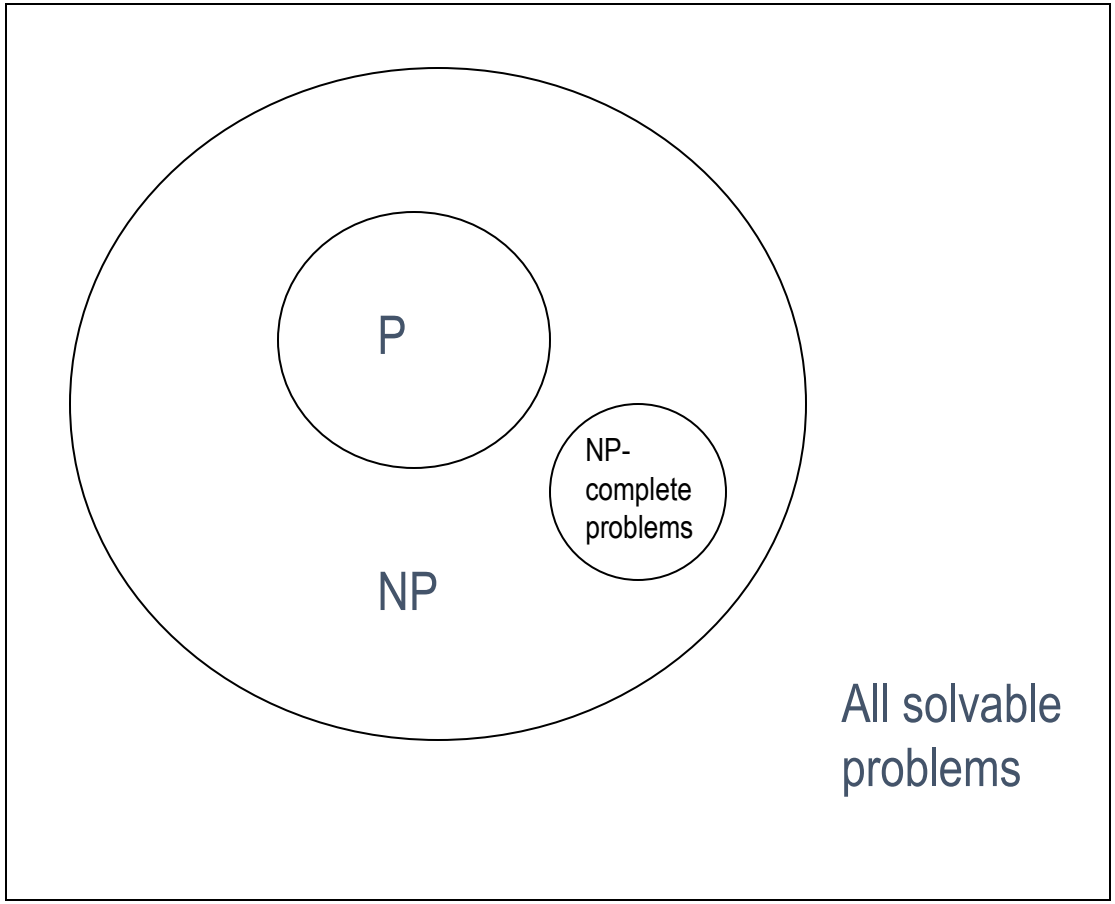
Some example problems include:

Decision version of 0/1 Knapsack.

Vertex cover.

Complexity Class	Characteristic feature
P	Easily solvable in polynomial time.
NP	Yes, answers can be checked in polynomial time.
NP-hard	All NP-hard problems are not in NP and it takes a long time to check them.
NP-complete	A problem that is NP and NP-hard is NP-complete.


P
NP
All solvable problems


P
NP-
complete
problems
NP
All solvable
problems