$\mathcal{P}$ and $\mathcal{NP}$

Ajit A. Diwan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay.

January 31, 2011.

Outline

- Computational Problems
- Algorithms for computational problems.
- Time-complexity of algorithms.
- The class $\mathcal{P}$ of computational problems.
- The class $\mathcal{NP}$ of computational problems.
- The relationship between $\mathcal{P}$ and $\mathcal{NP}$.

Composite Numbers

- A number is composite if it is the product of two numbers greater than 1.
- Is the number 10537374097354331118616940551364275782630920506 composite?
- Yes, the last digit is even!
- What about 10537374097354331118616940551364275782630920507?
- How will you find it out?
- How long will you take to find out?

Composite Numbers

- A number is composite if it is the product of two numbers greater than 1.
- Is the number 10537374097354331118616940551364275782630920506 composite?
- Yes, the last digit is even!
- What about 10537374097354331118616940551364275782630920507?
- How will you find it out?
- How long will you take to find out?

Composite Numbers

- A number is composite if it is the product of two numbers greater than 1.
- Is the number 1053737409735433118616940551364275782630920506 composite?
- Yes, the last digit is even!
- What about 1053737409735433118616940551364275782630920507?
- How will you find it out?
- How long will you take to find out?

Composite Numbers

- A number is composite if it is the product of two numbers greater than 1.
- Is the number 10537374097354331118616940551364275782630920506 composite?
- Yes, the last digit is even!
- What about 10537374097354331118616940551364275782630920507?
- How will you find it out?
- How long will you take to find out?

Composite Numbers

- A number is composite if it is the product of two numbers greater than 1.
- Is the number 10537374097354331118616940551364275782630920506 composite?
- Yes, the last digit is even!
- What about 10537374097354331118616940551364275782630920507?
- How will you find it out?
- How long will you take to find out?

Composite Numbers

- The number 10537374097354331118616940551364275782630920507 is composite.
- Why?
  10537374097354331118616940551364275782630920507
  = 20305109031301903224013097540117 9
  * 518951859904 33
- This is a proof that the number is composite.
- Is the proof correct?
- How long will you take to check it?

Composite Numbers

- The number
  10537374097354331118616940551364275782630920507
  is composite.
- Why?
  10537374097354331118616940551364275782630920507
  = 203051090313019032240130975401179
  * 51895185990433
- This is a proof that the number is composite.
- Is the proof correct?
- How long will you take to check it?

Composite Numbers

- The number 10537374097354331118616940551364275782630920507 is composite.
- Why? 10537374097354331118616940551364275782630920507 = 203051090313019032240130975401179 * 51895185990433
- This is a proof that the number is composite.
- Is the proof correct?
- How long will you take to check it?

Composite Numbers

- Is the number 203051090313019032240130975401179 composite?
- No, it is prime.
- Why?
- Proof?
- A number n is prime iff there exists a number a such that $a^{n-1} \equiv 1 \mod n$ and $a^{(n-1)/p} \not\equiv 1 \mod n$ for all primes p that divide $n-1$.

Composite Numbers

- Is the number 203051090313019032240130975401179 composite?
- No, it is prime.
- Why?
- Proof?
- A number n is prime iff there exists a number a such that $a^{n-1} \equiv 1 \mod n$ and $a^{(n-1)/p} \not\equiv 1 \mod n$ for all primes p that divide $n-1$.

Verifying Circuits

- Given a boolean circuit with n inputs and one output.
- Given the function that the circuit is supposed to compute.
- Does the circuit actually correctly compute the specified function?
- Simple approach: Evaluate the circuit for every possible input, and check that it correctly computes the required output for that input.
- 2^n possible inputs to be considered.
- Takes too long even for small n.
- Is there a better way than this?

Verifying Circuits

- Given a boolean circuit with n inputs and one output.
- Given the function that the circuit is supposed to compute.
- Does the circuit actually correctly compute the specified function?
- Simple approach: Evaluate the circuit for every possible input, and check that it correctly computes the required output for that input.
- 2^n possible inputs to be considered.
- Takes too long even for small n.
- Is there a better way than this?

Verifying Circuits

- To prove that a circuit is faulty, sufficient to find one input for which its output differs from the required output.
- To prove it is correct, we have to show that for all inputs it gives the required output.
- Is the second problem more difficult than the first?
- In what sense?

Verifying Circuits

- To prove that a circuit is faulty, sufficient to find one input for which its output differs from the required output.
- To prove it is correct, we have to show that for all inputs it gives the required output.
- Is the second problem more difficult than the first?
- In what sense?

Verifying Circuits

- To prove that a circuit is faulty, sufficient to find one input for which its output differs from the required output.
- To prove it is correct, we have to show that for all inputs it gives the required output.
- Is the second problem more difficult than the first?
- In what sense?

Definition of Computational Problems

- A problem is defined by an infinite set of instances, and a function that maps each instance to an "answer" for that instance.
- Instance: A positive number n.
  Answer: 1 if n is composite and 0 otherwise.
- Instance: A Boolean formula f in n variables and a logic circuit C with n inputs and one output.
  Answer: 1 if for all inputs the circuit correctly evaluates the formula and 0 otherwise.

Definition of Computational Problems

- A problem is defined by an infinite set of instances, and a function that maps each instance to an "answer" for that instance.
- Instance: A positive number n.
  Answer: 1 if n is composite and 0 otherwise.
- Instance: A Boolean formula f in n variables and a logic circuit C with n inputs and one output.
  Answer: 1 if for all inputs the circuit correctly evaluates the formula and 0 otherwise.

Definition of Computational Problems

- A problem is defined by an infinite set of instances, and a function that maps each instance to an "answer" for that instance.
- Instance: A positive number n.
  Answer: 1 if n is composite and 0 otherwise.
- Instance: A Boolean formula f in n variables and a logic circuit C with n inputs and one output.
  Answer: 1 if for all inputs the circuit correctly evaluates the formula and 0 otherwise.

Decision Problems

- Decision Problems are those for which the answer is 0 or 1 for all instances.
- Instances are usually encoded as strings of symbols, usually bit strings.
- A decision problem is identified by a “language”, the subset of all strings that encode instances of the problem for which the answer is 1.
- The “language” of composites $\{4, 6, 8, 9, 10, 12, 14, 15, 16, \ldots\}$

Algorithm for a Computational Problem

- An algorithm is a sequence of “basic” instructions that manipulates input data in a specified way and computes some output.
- Can consider it to be a C program, but usually “instructions” are assumed to be much simpler.
- An algorithm solves a computational problem if for every instance of the problem, it takes the instance as input and computes the answer for that instance as output, by executing a finite number of “basic” instructions.

Time Complexity of an Algorithm

- The time required by an algorithm, for solving a particular instance of a problem, is measured by the number of "basic" instructions executed to compute the output.
- The time can vary dramatically for different instances.
- Usually, the time increases with the "size" of input, which is a measure of the amount of data contained in the input.
- Measure time as a function of the "size" of input.
- When inputs are strings of symbols, "size" is essentially the number of symbols.
- Worst-case time is the maximum time required for inputs of size n.

Polynomial-time Algorithms

- The time complexity of an algorithm is at most $T(n)$ if for any input of "size" n, the algorithm computes the answer in at most $T(n)$ steps.
- Here $T(n)$ is a function of n.
- The algorithm is said to be a polynomial-time algorithm if $T(n)$ is bounded by some polynomial in n, that is $T(n) \leq cn^k$ for some constants c and k.
- $\mathcal{P}$ is the set of all decision problems that can be solved by polynomial-time algorithms.

Example of a Polynomial-Time Algorithm

Instance: Two positive integers n and m
Answer: Greatest Common Divisor of n and m.

```
while (m > 0)
{
          r = n % m;
          n = m;
          m = r;
}
return n;
```

If m is a k-bit number, the number of bit operations is at most ck^3.

Example of a Non-Polynomial-Time Algorithm

Instance: A positive integer *n*.
Answer: 1 if *n* is composite and 0 otherwise.

```
i = 2;
while ( i*i <= n)
{
          if  (n % i == 0) return 1;
          i++;
}
return 0;
```

If *n* is a *k*-bit number, the number of bit operations is at least $2^{k/2}$ in the worst case.
However, this problem is in $\mathcal{P}$, a famous result due to Agrawal, Kayal, Saxena (I.I.T. Kanpur).

Non-Deterministic Polynomial-Time Algorithms

- There are many problems for which no polynomial-time algorithms are known.
- However, it is not proved that no such algorithm can exist.
- Many of these problems seem to have a common property.
- This property is captured by the existence of non-deterministic polynomial-time algorithms for solving them.

Non-Deterministic Polynomial-Time Algorithm

A decision problem is said to be solvable by a non-deterministic polynomial-time algorithm if there is an algorithm such that

1. The algorithm takes an instance of the problem and some arbitrary unspecified additional data as input.
2. If the answer to the instance of the problem is 1, the algorithm outputs 1 for some choice of the additional unspecified data.
3. If the answer to the instance is 0, the algorithm always outputs 0, no matter what additional data is supplied to it.
4. The time complexity of the algorithm is polynomial in the size of the instance.

Example

- Instance: A positive number n.
- Answer: 1 if n is composite and 0 otherwise.
- Additional data is another positive number m
- return 1 if (1 < m < n) and m divides n
  otherwise return 0
- If n is composite, there is some choice of m for which the algorithm will output 1.
- If n is not composite, the algorithm will never output 1, no matter what m is.

Example

- Instance: A positive number n.
- Answer: 1 if n is composite and 0 otherwise.
- Additional data is another positive number m
- return 1 if (1 < m < n) and m divides n
  otherwise return 0
- If n is composite, there is some choice of m for which the algorithm will output 1.
- If n is not composite, the algorithm will never output 1, no matter what m is.

Satisfiability Problem

- Instance: A Boolean formula in product of sum form.
- Answer: 1 if the formula is satisfiable, that is, it evaluates to 1 for some assignment of values to the variables.
- Additional data is an assignment of values to the variables
- Evaluate the formula for the assignment given and return its value.
- If the formula is satisfiable, for some choice of the assignment it must evaluate to 1.
- If it is not satisfiable, it will evaluate to 0 for all assignments.

Satisfiability Problem

- Instance: A Boolean formula in product of sum form.
- Answer: 1 if the formula is satisfiable, that is, it evaluates to 1 for some assignment of values to the variables.
- Additional data is an assignment of values to the variables
- Evaluate the formula for the assignment given and return its value.
- If the formula is satisfiable, for some choice of the assignment it must evaluate to 1.
- If it is not satisfiable, it will evaluate to 0 for all assignments.

The class $\mathcal{NP}$

- $\mathcal{NP}$ is the set of all decision problems that can be "solved" by a non-deterministic polynomial-time algorithm.
- Associated with each instance of a problem in $\mathcal{NP}$ is a set of possible "solutions" for that instance.
- The "solutions" correspond to the additional data.
- The answer to an instance is 1 if there exists at least one solution satisfying some specified property.
- Whether a proposed solution satisfies the required property can be checked in time polynomial in the size of the instance.
- The number of possible solutions is exponential in the size of the instance.

$\mathcal{P}$ and $\mathcal{NP}$

- The million-dollar question: is $\mathcal{P}$ a proper subset of $\mathcal{NP}$?
- Are there problems in $\mathcal{NP}$ that cannot be solved in polynomial-time?
- Widely believed to be true.
- No proof found for some time, not clear whether any progress has been made at all.
- A large number of practically useful problems are in $\mathcal{NP}$ but not known to be (and believed not to be) in $\mathcal{P}$.

co-$\mathcal{NP}$

- Not all problems have non-deterministic polynomial-time algorithms.
- Is a given circuit faulty?
  A ”solution” is an input for which the circuit is faulty.
  Whether a circuit is faulty for a given input can be checked in polynomial-time.
- Is a given circuit correct?
  What are possible “solutions”?
  What property must be satisfied by the “solution” that can be checked in polynomial-time?
  No such solutions or properties known (and believed not to exist).
- Complements of problems in $\mathcal{NP}$ need not be in $\mathcal{NP}$.
- The set of these problems is denoted co-$\mathcal{NP}$.
- Is co-$\mathcal{NP} = \mathcal{NP}$?

co-$\mathcal{NP}$

- Not all problems have non-deterministic polynomial-time algorithms.
- Is a given circuit faulty?
  A ”solution” is an input for which the circuit is faulty.
  Whether a circuit is faulty for a given input can be checked in polynomial-time.
- Is a given circuit correct?
  What are possible “solutions”?
  What property must be satisfied by the “solution” that can be checked in polynomial-time?
  No such solutions or properties known (and believed not to exist).
- Complements of problems in $\mathcal{NP}$ need not be in $\mathcal{NP}$.
- The set of these problems is denoted co-$\mathcal{NP}$.
- Is co-$\mathcal{NP} = \mathcal{NP}$?

co-$\mathcal{NP}$

- Not all problems have non-deterministic polynomial-time algorithms.
- Is a given circuit faulty?
  A ”solution” is an input for which the circuit is faulty.
  Whether a circuit is faulty for a given input can be checked in polynomial-time.
- Is a given circuit correct?
  What are possible “solutions”?
  What property must be satisfied by the “solution” that can be checked in polynomial-time?
  No such solutions or properties known (and believed not to exist).
- Complements of problems in $\mathcal{NP}$ need not be in $\mathcal{NP}$.
- The set of these problems is denoted co-$\mathcal{NP}$.
- Is co-$\mathcal{NP} = \mathcal{NP}$?

$\mathcal{NP}$-Completeness

- Many problems in $\mathcal{NP}$ not known to be in $\mathcal{P}$.
- However, we can show that if any one of them is in $\mathcal{P}$ then all of them are.
- A problem in $\mathcal{NP}$ is said to be $\mathcal{NP}$-Complete if a polynomial-time algorithm for the problem implies a polynomial-time algorithm for every problem in $\mathcal{NP}$.
- Not clear why such a problem should exist.
- Cook's theorem : Satisfiability problem is $\mathcal{NP}$-Complete.
- $\mathcal{P} = \mathcal{NP}$ if and only if Satisfiability can be solved in polynomial-time.

Proving $\mathcal{NP}$-Completeness

To prove a decision problem is $\mathcal{NP}$-Complete

1. Show that it is in $\mathcal{NP}$.
2. Show that a polynomial-time algorithm for this problem would imply a polynomial-time algorithm for some known $\mathcal{NP}$-Complete problem.

This would imply a polynomial-time algorithm for every problem in $\mathcal{NP}$.

A large number of problems known to be $\mathcal{NP}$-Complete.

Proving $\mathcal{NP}$-Completeness

To prove a decision problem is $\mathcal{NP}$-Complete

1. Show that it is in $\mathcal{NP}$.
2. Show that a polynomial-time algorithm for this problem would imply a polynomial-time algorithm for some known $\mathcal{NP}$-Complete problem.

This would imply a polynomial-time algorithm for every problem in $\mathcal{NP}$.
A large number of problems known to be $\mathcal{NP}$-Complete.

Timetabling Problem

- Instance: A set C of courses and a set of E of unordered pairs of courses, representing conflicts (common students) between courses.
- Answer: An assignment of courses to slots such that conflicting courses are in different slots, and the number of slots used is minimized.
- A simpler decision problem: Answer 1 if 3 slots are sufficient and 0 otherwise.
- The decision problem is in $\mathcal{NP}$.
- It is also $\mathcal{NP}$-Complete.

Timetabling Problem

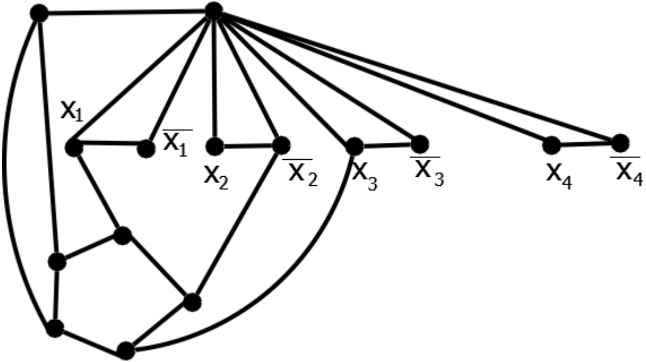
- A polynomial-time algorithm for the Timetabling decision problem gives a polynomial-time algorithm for the Satisfiability problem.
- Convert an instance of Satisfiability to an instance of Timetabling.
- "Solutions" for instance of Satisfiability correspond to "solutions" for Timetabling.
- Valid solutions for Satisfiability correspond to valid solutions for Timetabling and vice versa.
- Answer to instance of Satisfiability is 1 if and only the answer to instance of Timetabling is 1.
- If we have a polynomial-time algorithm for Timetabling we get a polynomial-time algorithm for Satisfiability.

$\mathcal{NP}$-Completeness of Timetabling

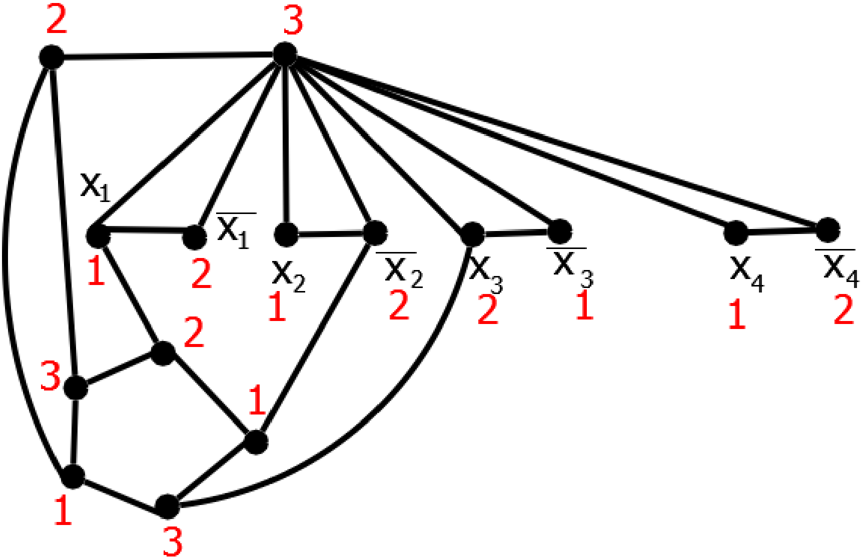
Instance of Satisfiability:

$(x_1 + \overline{x_2} + x_3).(\overline{x_1} + x_4 + \overline{x_5})\ldots$

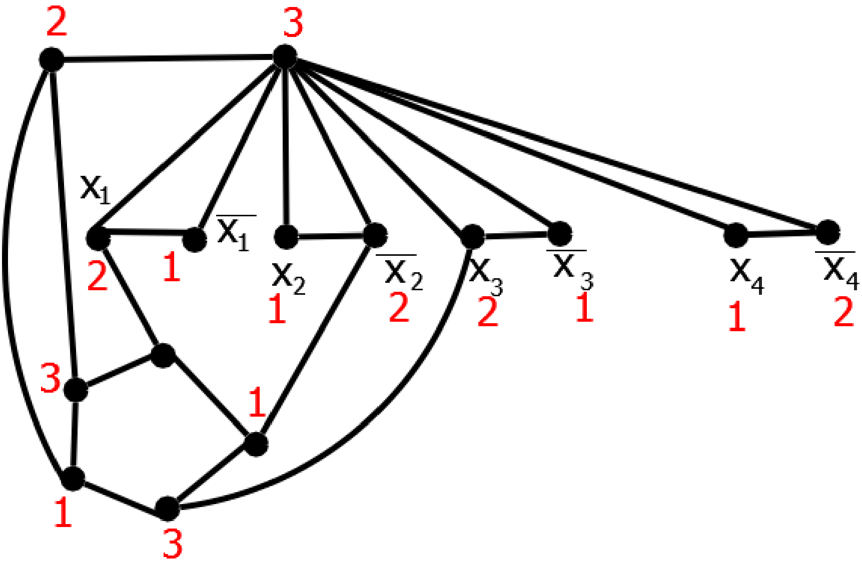
Corresponding instance of Timetabling:



$\mathcal{NP}$-Completeness of Timetabling



$\mathcal{NP}$-Completeness of Timetabling



Conclusion

- $\mathcal{P}$ is the class of problems that can be solved by polynomial-time algorithms.
- $\mathcal{NP}$ is the class of problems for which a proposed solution can be verified in polynomial-time.
- Is $\mathcal{P} == \mathcal{NP}$?
- Can solutions that can be verified easily also be found easily?
- Intuition suggests they are different.
- Can it be proved formally?

Thank You