Tractable and Intractable problems

CS483
Jana Kosecka

Tractable problems
- Algorithms for sorting, searching, graph traversal
- Can run in time polynomial, logarithmic in the size of input

For many cases feasible (polynomial time) algorithms are not Known => intractable problems - take exponential time

Reasonable vs Unreasonable time

Good function - polynomial time (bounded by polynomial n^k)

Tractable problems

Bad functions - super-polynomial (all the other ones)
- exponential, factorial and combination of those

Intractable problems
>> it is not just matter of having faster computer - see Table 1.1 (book)

Class P - class of problems - polynomial time bounded solutions

Class NP - class of problems - nondeterministic polynomial time bounded (to be defined in more detail shortly)

Types of Problems

- Combinatorial optimization problems
- Decision Problems: given the instance of the problem the output is yes or no
- Each combinatorial optimization problem can be reformulated as a decision problem but putting the bound on cost
Examples

Optimization problem: Given a graph $G(V,E)$ determine optimal coloring $C(G)$ such that no two neighboring vertices are colored using the same color.

Decision problem: Given $G(V,E)$ and $k$, is there such coloring which uses only $k$ colors?

Optimization problem: Given a weighted graph, what is the minimum weight cycle which visits each node exactly once?

Decision problem: Given a weighted graph and integer $k$ is there a cycle with weight at most $k$ which visits each node exactly once?

Since decision problems seem easier, if we can show decision problem is hard, then associated optimization problem is also hard.

e.g. 25 puzzle pieces and some constraints, how they should fit together; we need to find out whether we can arrange them in such a way that they will fit into 5x5 square.

- Decision problem: Does such an arrangement exist?
- Solution: Try all possibilities $25!$ (26 digit number)

Even if we can realize million arrangements per second it would take 460 million years ... This is the worst case analysis...
we can get many partial solutions - but there are just few which are complete.

Many problems arising in economics, graph theory, logic, operation research are of this nature >> they only admit exponential solutions and the best know lower bounds are linear

- General jigsaw puzzle
- Path finding problems - traveling salesmen, Hamiltonian path
- Scheduling and matching problems
- Determining the truth of logical formulas

Traveling salesman

Traveling salesman - find a minimal cost tour such that you visit each city at least once and return to the starting point.

Naive solution try all possible tours $O(N!)$

Hamiltonian path

Is there a path which passes though all points exactly once? (similar then traveling salesman with no costs).

If you can visit some of the vertices more than once, but each edge exactly once >> simpler Eulerian path

Eulerian path is simpler because we can clearly relate the condition on the graph which must be satisfied in order for the solution to exist (then we can just check the property)

E.g. if the graph is connected and number of edges emanating from any point is even (except two points) then you can do it
More Intractable problems

**Timetable problem** - teachers, classes, hours, schedule such that no two teachers teach the same class at the same time and no two classes are taught by the same teacher at the same time.

**Satisfiability problem** - formula in a predicate logic, does there exist a truth assignment to the assertions such the formula is satisfied.

If I have n assertions - all possible evaluations is $2^n$ - exponential.

**Graph coloring problem** - 4 (2) coloring problem for maps tractable, 3 coloring problem hard.

NP class

Hard to find the solution in general:
- it is quite easy to check whether given instance is a solution
- partial solutions are easy
- verification of an instance can be done in polynomial time.

**Nondeterministic algorithm**
1. Guessing a string - try to interpret it as a guess of the solution (string s - is the certificate)
2. Verify whether the string is a solution to the instance of the decision problem - answer true, false, no answer
3. Output - if certificate passed - answer of the verification true

Example: Graph Coloring example

NP Class

Nondeterministic algorithm - every time we run it, we get different answer.
Verification Phase - given instance string - it can be determined in polynomial time whether it's a valid solution.

**Example**:
- given particular coloring - it is easy to check whether is valid coloring
- Hamiltonian cycle - given particular cycle
  - If easy to check whether it is a Hamiltonian cycle

How do we know that there is no better algorithm?
- There are many problems of this sort which admit only unreasonable solutions (e.g. exponential bounds)
- But best know lower bound are at most polynomial

P = NP

P - solvable in polynomial time - solution can be obtained in polynomial time.

NP - verifiable in polynomial time - solution can be verified in polynomial time (nondeterministic polynomial).

NP complete problems - problems which are in NP but are as hard as any other problem in NP.

- If there were a polynomially bounded algorithm for one of the problems from this class then there would be a polynomially bounded algorithm for each problem in NP class (this would mean that NP = P).

We need general means of determining whether the problem is hard (intractable). Why?

1. You can use known algorithm for it and accept that it will take long
2. Look for approximating solutions
3. Change your problem formulation
**Examples of problems**

Many problems which look very similar but the known solutions are very different:

- **Minimum spanning tree** - Given a weighted graph and integer $k$, is there a spanning tree whose total weight is less than $k$?
- **Traveling salesmen** - Given a weighted graph and integer $k$, is there a cycle that visits all nodes exactly once and has a total weight $k$ or less?
- **Eulerian graph** - Given a directed graph, is there a closed path that visits each edge exactly once?
- **Hamiltonian graph** - Given a directed graph, is there a closed path that visits each node of the graph exactly once?
- **Circuit value** - Given a Boolean circuit and its inputs is the value true?
- **Circuit Satisfiability** - Given a Boolean circuit, is there a set of inputs that the output is true?
- **2-SAT** - Given a Boolean formula in 2-CNF is there a satisfying truth assignment to input variables?
- **3-SAT** - Given a Boolean formula in 3-CNF is there a satisfying truth assignment to input variables?

---

**NP completeness**

How to compare relative hardness of problems? Notion of reduction of one problem to another.

Given two problems $P$ and $Q$, there exist polynomial time reduction which reduces one problem to another.

Idea: Given input to problem $P$, we transform the input of $P$ to the input for problem $Q$, and if $Q$ has answer YES (NO), then $P$ has answer YES (NO).

If we can reduce $P$ to $Q$ in polynomial time, then $P$ is no harder than $Q$ (up to a polynomial)

**Theorem:** If $P \leq Q$ and $Q$ is in $P$, then $P$ is in $P$ class.

1. $q(n)$ to solve $Q$, $T(x)$ is at most $p(n)$ so the total amount of work for solving $P$ is $p(n) + q(p(n))$, i.e. is polynomially bounded.

**NP complete and NP hard problems**

A problem $Q$ is NP-hard if every problem in NP is reducible to $Q$: $P \leq Q$.

A problem $Q$ is NP-complete if it is NP-hard and is in NP.

To show that a problem is NP-complete:

1. Show that it is NP, 2. Show that every other problem in NP is polynomially reducible to it.

How to show that a problem $Q$ is NP complete? (without trying to show that every other problem in NP is reducible to it):

1. Select a known NP complete problem $P$, 2. Show there exist a polynomial time reduction of $P$ to $Q$.

i.e. we need to show for single NP complete problem that it is reducible to $Q$.

**Circuit Satisfiability is NP-complete**

1. Circuit SAT is NP can be verified in polynomial time (i.e., given a circuit and an input we can verify in polynomial time whether the input is a satisfying assignment).
2. Circuit SAT is NP-hard Every problem in NP is reducible to circuit SAT.

1. Problem is in NP -> can be verified in polynomial time by some algorithm.
2. Each step of the algorithm runs on a computer - huge boolean circuit.
3. Chaining together all circuits which correspond to the steps of the algorithm - we get large circuit which describes the run of the algorithm.
4. If we plug in the input of a problem $A$ then YES / NO answer when circuit is / is not satisfiable.

**Problem is in NP if and only if it can be reduced to circuit SAT.**
Unary knapsack - Given $a_1, \ldots, a_n$ and $K$ in unary is there a subset of integers that sum exactly to $K$?

Knapsack - given $a_1, \ldots, a_n$ and $k$ in binary is there a subset of integers that sum exactly to $K$?

All the NP problems are verifiable in polynomial time - given a certificate (a solution that can be easily checked)

NP complete problems - problems which are in NP but are as hard as any other problem in NP

$P \neq NP$

Quite unlikely - since NP complete problems are inherently intractable, but it has not been proven.

Alternative solutions to intractable problems:
Approximation algorithms
Randomized algorithms
Partial solutions