Integer Programming
ISE 418

Lecture 26

Dr. Ted Ralphs

Reading for This Lecture

- N&W Sections I.5.3-I.5.6
- Wolsey Chapter 6
- CCZ Chapter 1
- "Computers and Intractability: A Guide to the Theory of NP-Completeness," Garey and Johnson

Complexity Classes

- In the last lecture, we discussed the concept of a running time function, as a way of comparing both algorithms and problems.
- The running time function can be used to separate problems into equivalence classes, but this classification is too fine-grained.
- We typically want to know something simpler than the exact running time function, e.g., whether the problem is solvable in polynomial time.
- In this lecture, we'll describe the classes into which problems are divided in the classical theory of NP-completeness.
- This is the scheme used in the vast majority of the literature on mathematical optimization.
- We'll also see that placing problems into classes can be done not only by determining the running time function, but also by reduction.

Decision Problems and Optimization Problems

- A *decision problem* or *feasibility problem* is a problem for which the answer is either yes or no.
- For primarily historical reasons, complexity theory is defined in terms of decision problems.
- Any optimization problem can be solved by a sequence of decision problems (why?).
- Example: The Bin Packing Problem
  - We are given a set S of items, each with a specified integral size, and a specified constant C, the size of a *bin*.
  - Optimization problem: Determine the smallest number of subsets into which one can partition S such that the total size of the items in each subset is at most C.
  - Decision problem: For a given constant K, determine whether S can be partitioned into K subsets such that that the total size of the items in each subset is at most C.

Certificates

- A *certificate* is a "proof" we can check that certifies the output of a given computation is correct.
- It is often possible to check the validity of such a certificate more efficiently than to solve the original problem.
- Suppose you had the optimal solution to an LP and wanted to prove to someone else it was optimal.
- You could simply produce the primal and dual solutions.
- Can optimality be verified in polynomial time?
  - In $O(mn)$ operations, one could verify optimality.
  - However, what is the magnitude of the numbers?
  - They are the ratio of two integers, each of which can be represented in a size that is polynomially bounded.
- The information that can be used to check the results of a computation is called a *certificate*.

Certificate and Algorithms

- A certificate that can be checked in polynomial time is said to be *short*.
- One way of producing a certificate is just to record the sequence of steps that resulted in the answer.
- Thus, every polynomially solvable problem has a short certificate.
- It is not known whether every problem with a short certificate is polynomially solvable.
- Until 1979, linear programming was one problem with a short certificate that was not known to be polynomially solvable.
- The Perfect Matching Problem
  - Recall we derived a complete description of the perfect matching polytope.
  - Although the formulation has an exponential number of constraints, this yields a polynomial certificate.
  - This problem can in fact be solved in polynomial time.

Certificates for Decision Problems

- For many decision problems, there is a short certificate only in the case of one of the two possible answers.
- Typically, the problem is formulated such that the answer with a short certificate is YES).
- (Imperfect) Example: The meeting room problem
  - Decision: Is there anyone in this room that I don't know?
  - There is a short certificate for the YES answer. What is it?
- Example: General integer programming
  - What is the decision version of this problem?
  - Is there a short certificate?

Problem Reduction

- Recall that mixed integer linear optimization is a *special case* of general mathematical optimization.
- If we had a fast algorithm for solving general mathematical optimization problems, we would be able to solve MILPs as well.
- Similarly, the Traveling Salesman Problem is a special case of pure integer linear optimization.
- Hence, general integer programming is, in some sense, *at least as difficult as* the TSP.
- Starting from a few problems that we analyze from first principles, we can divide problems into classes using the concept of *reduction*.

Polynomial Reduction

- Suppose we are given two problems X_1 and X_2.
- We want to show that if we solve one, we can also solve the other.
- We say X_1 is polynomially reducible to X_2 if
  1. there is an algorithm for X_1 that uses the algorithm for X_2 as a subroutine, and
  2. the algorithm runs in polynomial time under the assumption that the subroutine runs in constant time.
- This implies immediately that if X_2 is polynomially solvable and X_1 is polynomially reducible to X_2, then X_1 is polynomially solvable.
- A subroutine that we assume runs in constant time for the purpose of doing a reduction is called an *oracle*.
- Using polynomial reduction to divide problems has pros and cons.
  - One one hand, "equivalence" can be determined without knowing the precise running time functions.
  - On the other hand, the classes obtained in this way are much less "fine-grained."

More Formally

- The formal model of computation underlying the analysis of decision problems is referred to as a *deterministic Turing machine* (DTM).
  - A DTM specifies an *algorithm* computing the value of a Boolean function.
  - The DTM executes a program, reading the input from a *tape*.
  - We equate a given DTM with the program it executes.
  - The output is YES or NO.
  - A YES answer is returned if the machine reaches an *accepting state*.
- A problem is specified in the form of a *language*.
- The language is the subset of the possible inputs over a given *alphabet* (Γ) that are expected to output YES.
- A DTM that produces the correct output for inputs w.r.t. a given language is said to *recognize the language*.
- Informally, we can then say that the DTM represents an “algorithm that solves the given problem correctly.”

Non-deterministic Turing Machines

- A *non-deterministic Turing machine* (NDTM) can be thought of as a Turing machine with an infinite number of parallel processors.
- An NDTM follows all possible execution paths simultaneously.
- It returns YES if an accepting state is reached on *any* path.
- The running time of an NDTM is the *minimum* running time (length) of any execution paths that end in an accepting state.
- The running time is the minimum time required to verify that some path (given as input) leads to an accepting state.

Complexity Classes

- As described previously, languages are grouped into *classes* based on the *best worst-case running time function* of any TM that recognizes the language.
- The running time function is as described previously.
  - The class P is the set of all languages for which there exists a DTM that recognizes the language in time polynomial in the length of the input.
  - The class NP is the set of all languages for which there exists an NDTM that recognizes the language in time polynomial in the length of the input.
  - The class coNP is the set of languages whose complements are in NP.
- Additional classes are formed hierarchically by the use of *oracles*.

Reduction

- A problem specified by a language L_1 can be *reduced* to a problem specified by a language L_2 if there is a polynomial time transformation of strings that maps
  - each string in L_1 to a string in L_2, and
  - each string not in L_1 to a string not in L_2.
- A problem specified by a language L is said to be *complete* for a class if all problems in the class can be reduced to it.

Another Way to Think About It

- A nondeterministic algorithm is an algorithm that corresponds to an NDTM.
- The input to the algorithm is a string $s \in \Gamma^*$.
- Conceptually we can think of the algorithm as having two stages
  - Guessing Stage: Randomly guess a string q (the certificate).
  - Checking Stage: Check whether q can be used to verify that $d \in L$. If so, output YES. If not, there is no output.
- There are two properties required.
  - We require that if $d \in L$, then there must exist a certificate that verifies the feasibility of d.
  - The running time of the algorithm is the maximum time it takes to check a certificate that verifies $d \in L$.

NDTMs and Certificates

- Non-deterministic algorithms are so called because the guessing stage is random.
- We can use the description of the path that eventually leads to an accepting state as the certificate.
- If the running time of the NDTM is polynomial, then the certificate for the YES answer is therefore short.
- If no accepting path is found, there is no short certificate in general.
  - The *YES* answer is an "existential" statement ($?\exists x$s.t. $\ldots$).
  - The *NO* answer is a "universal" statement ($\forall x \ldots$)
- Another way of describing the class NP is the class for which there exists a certificate for the YES answer that can be checked in polynomial time.
- Examples of problems in NP.
  - General integer programming feasibility.
  - The decision version of bin packing.

P, NP, and coNP

- The class of problem for which *deterministic* polynomial-time algorithms exist is denoted P.
- Obviously, P is a subset of NP.
- It is not known whether P $=$ NP (the million dollar question).
- $coNP$ is the class of problems for which the complement is in NP.
- In other words, it is the class of decision problem for which there is a certificate verifying a no answer.
- P is also a subset of coNP.
- If the decision version of an optimization problem is in $\mathrm{NP} \cap \mathrm{coNP}$, then there exists a certificate of optimality.
- It is unlikely that there exist many problems in $\mathrm{NP} \cap \mathrm{coNP}$ that are not also in P.

The Class NP-complete

- It is interesting to ask what are the hardest problems in NP?
- We say that a problem X is *complete* for NP if every problem in NP is polynomially reducible to X.
- Surprisingly, such problems exist!
- Even more surprisingly, this class contains almost every interesting integer programming problem that is not known to be in P!

Proposition 1. *If $X \in NPC$, then $X \in P \Leftrightarrow P = NP$.*

Proposition 2. *If $X_1 \in NPC$ and X_1 is polynomially reducible to X_2, then $X_2 \in NPC$.*

The Satisfiability Problem

- This is the first problem proven to be NP-complete.
- The problem is described by
  1. a finite set $N = \{1, \ldots, n\}$ (the *literals*), and
  2. m pairs of subsets of N, $C_i = (C_i^+, C_i^-)$ (the *clauses*).
- An instance is feasible if the set

$$\left\{ x \in \mathbb{B}^n \| \sum_{j \in C_i^+} x_j + \sum_{j \in C_i^-} (1 - x_j) \geq 1 \text{ for } i = 1, \ldots, m \right\}$$

  is nonempty.
- This problem is obviously in NP (why?).
- In 1971, Cook defined the class NP and showed that satisfiability was NP-complete, even if each clause only contains three literals.
- The proof is beyond the scope of this course.

Proving NP-completeness

- After satisfiability was proven to be NP-complete, it was easy to prove many other problems NP-complete.
- This is done by polynomial reduction.
- Example: The k-Clique Problem
  - Does a given graph have a clique of size k?
  - Although it seems simple, this problem is $\mathcal{NP}$-complete.
  - This problem is easily shown to be in NP.
  - To prove it is in NP-complete, we reduce 3-satisfiability to it.

The Line Between P and NP-complete

- Generally speaking, most interesting problems are either known to be in P or are NP-complete.
  - The problems known to be in P are generally "easy" to solve.
  - The problems in NPC are generally "hard" to solve.
- This is very intriguing!
- The line between these two classes is also very thin!
  - Consider a 0-1 matrix A, an cost vector $c \in \mathbb{Z}^n$, $z \in \mathbb{Z}$ defining the decision problem

$$\{x \in \mathbb{B}^n \| Ax \leq 1, cx \geq z\}$$

  - If we limit the number of nonzero entries in each column to 2, then this problem is known to be in P (what is it?).
  - If we allow the number of nonzero entries in each column to be three, then this problem is NP-complete!

NP-hard Problems

- The class NP-hard extends NP-complete to include problems that are not in NP.
- If $X_1 \in NPC$ and X_1 reduces to X_2, then X_2 is said to be NP-hard.
- Thus, all NP-complete problems are NP-hard.
- The primary reason for this definition is so we can classify optimization problems that are not in NP.
- It is common for people to refer to optimization problems as being NP-complete, but this is technically incorrect.

Theory versus Practice

- In practice, it is true that most problem known to be in P are “easy” to solve.
- This is because most known polynomial time algorithms are of relatively low order.
- It seems very unlikely that $P = NP$.
- If so, the reduction is likely to be prohibitively expensive.
- For similar reasons, although all NP-complete problems are “equivalent” in theory, they are not in practice.
- TSP vs. QAP

Wrap-up

- There are many other possible ways of analyzing complexity.
- Others have been discussed, but this one seems to be the best anyone has come up with.
- If someone resolves whether $\mathsf{P} = \mathsf{NP}$, we will have to come up with something new.