Lecture 21 Reductions and NP-Completeness

We have seen several problems such as maximum flow and matching that at first glance appear intractible, but upon closer study admit very efficient algorithms. Unfortunately, this is the exception rather than the rule. For every interesting problem with a polynomial-time algorithm, there are dozens for which all known solutions require exponential time in the worst case. These problems occur in various fields, to wit:

Logic:

- *CNF satisfiability (CNFSat)*: given a Boolean formula $\mathcal{B}$ in conjunctive normal form (CNF), is there a truth assignment that satisfies $\mathcal{B}$?

Graph Theory:

- *Clique*: given a graph $G = (V, E)$ and an integer m, does G contain K_m (the complete graph on m vertices) as a subgraph?
- *k-Colorability*: given a graph $G = (V, E)$ and an integer k, is there a coloring of G with k or fewer colors? A *coloring* is a map $\chi : V \to C$ such that no two adjacent vertices have the same color; *i.e.*, if $(u, v) \in E$ then $\chi(u) \neq \chi(v)$.

Operations Research:

- Any of a number of generalizations of the one-processor scheduling problem of Miscellaneous Exercise 4.
- *Integer Programming*: given a set of linear constraints A and a linear function f, find an integer point maximizing f subject to the constraints A.
- The *Traveling Salesman Problem (TSP)*: given a set of cities and distances between them, find a tour of minimum total distance visiting all cities at least once.

None of these problems are known to have a polynomial time solution. For example, the best known solutions to the Boolean satisfiability problem are not much better than essentially evaluating the given formula on all 2^n truth assignments. On the other hand, no one has been able to prove that no substantially better algorithm exists, either.

However, we can show that all these problems are computationally equivalent in the sense that if one of them is solvable by an efficient algorithm, then they all are. This involves the concept of *reduction*. Intuitively, a problem A is said to be *reducible* to a problem B if there is a way to encode instances x of problem A as instances $\sigma(x)$ of problem B. The encoding function σ is called a *reduction*. If σ is suitably efficient, then any efficient algorithm for B will yield an efficient algorithm for A by composing it with σ.

The theory has even deeper implications than this. There is a very general class of decision problems called *NP*, which roughly speaking consists of problems that can be solved efficiently by a nondeterministic guess-and-verify algorithm. A problem is said to be *NP-complete* if it is in this class and every other problem in *NP* reduces to it. Essentially, it is a hardest problem in the class *NP*. If an *NP*-complete problem has an efficient deterministic solution, then so do all problems in *NP*. All of the problems named above are known to be *NP*-complete.

The theory of efficient reductions and *NP*-completeness was initiated in the early 1970s. The two principal papers that first demonstrated the importance of these concepts were by Cook [22], who showed that Boolean satisfiability was *NP*-complete, and Karp [57, 58] who showed that many interesting combinatorial problems were interreducible and hence *NP*-complete. Garey and Johnson's text [39] provides an excellent introduction to the theory of *NP*-completeness and contains an extensive list of *NP*-complete problems. By now the problems known to be *NP*-complete number in the thousands.

21.1 Some Efficient Reductions

We have seen examples of reductions in previous lectures. For example, Boolean matrix multiplication and transitive closure were shown to be re-

ducible to each other. To illustrate the concept further, we show that CNFSat, the satisfiability problem for Boolean formulas in conjunctive normal form, is reducible to the clique problem.

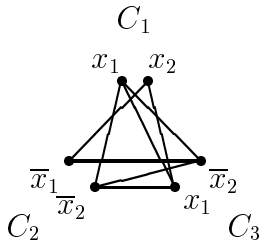
Definition 21.1 Let $\mathcal{B}$ be a Boolean formula. A *literal* is either a variable or the negation of a variable (we write $\neg x$ and $\overline{x}$ interchangeably). A *clause* is a disjunction of literals, *e.g.* $C = (x_1 \vee \neg x_2 \vee x_3)$. The formula $\mathcal{B}$ is said to be in *conjunctive normal form (CNF)* if it is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_m$. □

Note that to satisfy a formula in CNF, a truth assignment must assign the value *true* to at least one literal in each clause, and different occurrences of the same literal in different clauses must receive the same truth value.

Given a Boolean formula $\mathcal{B}$ in CNF, we show how to construct a graph G and an integer k such that G has a clique of size k iff $\mathcal{B}$ is satisfiable. We take k to be the number of clauses in $\mathcal{B}$. The vertices of G are all the *occurrences* of literals in $\mathcal{B}$. There is an edge of G between two such occurrences if they are in different clauses and the two literals are not complementary. For example, the formula

$$\underset{(x_1 \vee x_2)}{C_1} \wedge \underset{(\overline{x}_1 \vee \overline{x}_2)}{C_2} \wedge \underset{(x_1 \vee \overline{x}_2)}{C_3}$$

would yield the graph



The graph G is k-partite and has a k-clique iff $\mathcal{B}$ is satisfiable. Essentially, an edge between two occurrences of literals represents the ability to assign them both *true* without a local conflict; a k-clique thus represents the ability to assign *true* to at least one literal from each clause without global conflict. In the example above, $k = 3$ and there are two 3-cliques (triangles) corresponding to two ways to satisfy the formula.

Let us prove formally that G has a k-clique iff $\mathcal{B}$ is satisfiable. First assume that $\mathcal{B}$ is satisfiable. Let $\tau : \{x_1, \ldots, x_n\} \to \{true, false\}$ be a truth assignment satisfying $\mathcal{B}$. At least one literal in each clause must be assigned *true* under τ. Choose one such literal from each clause. The vertices of G corresponding to these true literals are all connected to each other because no pair is complementary, so they form a k-clique. Conversely, suppose G has a k-clique. Since G is k-partite and the partition elements correspond to the

clauses, the k-clique must have exactly one vertex in each clause. Assign *true* to the literals corresponding to the vertices in the clique. This can be done without conflict, since no pair of complementary literals appears in the clique. Assign truth values to the remaining variables arbitrarily. The resulting truth assignment assigns *true* to at least one literal in each clause, thus satisfies $\mathcal{B}$.

We have just shown how to encode a given instance of the CNFSat problem in an instance of the clique problem, or in the accepted parlance, *reduced* the CNFSat problem to the clique problem.

An important caveat: a reduction reduces the problem being encoded to the problem encoding it. Sometimes you hear it said backwards; for example, that the construction above reduces Clique to CNFSat. This is incorrect.

Although we do not know how to solve Clique or CNFSat in any less than exponential time, we do know by the above reduction that if tomorrow someone were to come up with a polynomial-time algorithm for Clique, we would immediately be able to derive a polynomial-time algorithm for CNFSat: given $\mathcal{B}$, just produce the graph G and k as above, and apply the polynomial-time algorithm for Clique. For the same reason, if tomorrow someone were to show an exponential lower bound for CNFSat, we would automatically have an exponential lower bound for Clique.

We show for purposes of illustration that there is a simple reduction in the other direction as well. To reduce Clique to CNFSat, we must show how to construct from a given undirected graph $G = (V, E)$ and a number k a Boolean formula $\mathcal{B}$ in CNF such that G has a clique of size k if and only if $\mathcal{B}$ is satisfiable.

Given $G = (V, E)$ and k, take as Boolean variables x_i^u for $u \in V$ and $1 \leq i \leq k$. Intuitively, x_i^u says, "u is the i^{th} element of the clique." The formula $\mathcal{B}$ is the conjunction of three subformulas $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$, with the following intuitive meanings and formal definitions:

- $\mathcal{C}$ = "For every i, $1 \leq i \leq k$, there is at least one $u \in V$ such that u is the i^{th} element of the clique."

$$\mathcal{C} \;=\; \bigwedge_{i=1}^{k} \Big(\bigvee_{u \in V} x_i^u \Big) \,.$$

- $\mathcal{D}$ = "For every i, $1 \leq i \leq k$, no two distinct vertices are both the i^{th} element of the clique."

$$\mathcal{D} \;=\; \bigwedge_{i=1}^{k} \bigwedge_{\substack{u,\, v \,\in\, V \\ u \,\neq\, v}} (\neg x_i^u \vee \neg x_i^v) \,.$$

- $\mathcal{E}$ = "If u and v are in the clique, then (u, v) is an edge of G. Equivalently, if (u, v) is not an edge, then either u is not in the clique or v is not in

the clique."

$$\mathcal{E} \;=\; \bigwedge_{(u,v)\notin E} \;\bigwedge_{1\le i,j\le k} (\neg x_i^u \vee \neg x_j^v)\ .$$

We take $\mathcal{B} = \mathcal{C}\wedge\mathcal{D}\wedge\mathcal{E}$. Any satisfying assignment τ for $\mathcal{C}\wedge\mathcal{D}$ picks out a set of k vertices, namely those u such that $\tau(x_i^u) = \mathit{true}$ for some i, $1 \le i \le k$. If τ also satisfies $\mathcal{E}$, then those k vertices form a clique. Conversely, if $u_1, \ldots, u_k$ is a k-clique in G, set $\tau(x_i^{u_i}) = \mathit{true}$, $1 \le i \le k$, and set $\tau(y) = \mathit{false}$ for all other variables y; this truth assignment satisfies $\mathcal{B}$.

It is perhaps surprising that two problems so apparently different as CNFSat and Clique should be computationally equivalent. However, this turns out to be a widespread phenomenon.