The Design and Analysis of Algorithms

Dexter C. Kozen

With 72 Illustrations

Springer-Verlag
New York Berlin Heidelberg London Paris
Tokyo Hong Kong Barcelona Budapest

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.

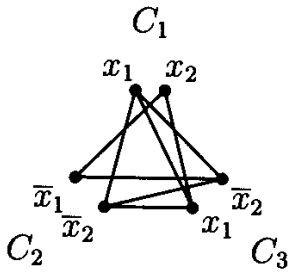
Definition 21.1 Let $\mathcal{B}$ be a Boolean formula. A *literal* is either a variable or the negation of a variable (we write $\neg x$ and $\overline{x}$ interchangeably). A *clause* is a disjunction of literals, *e.g.* $C = (x_1 \vee \neg x_2 \vee x_3)$. The formula $\mathcal{B}$ is said to be in *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

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

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 $\tau : \{x_1, \ldots, x_n\} \to \{\mathit{true}, \mathit{false}\}$ 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."

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

- $\mathcal{D}$ = "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.

Lecture 22 More on Reductions and *NP*-Completeness

Before we give a formal definition of reduction, let us clarify the notion of a *decision problem.* Informally, a decision problem is a yes-or-no question. A decision problem is given by a description of the *problem domain*, *i.e.* the set of all possible instances of the problem, along with a description of the set of "yes" instances.

For example, consider the problem of determining whether a given undirected graph G has a k-clique. An instance of the problem is a pair (G, k), and the problem domain is the set of all such pairs. The "yes" instances are the pairs (G, k) for which G has a clique of size k.

There are many interesting discrete problems that are not decision problems. For example, many optimization problems like the traveling salesman problem or the integer programming problem ask for the calculation of an object that maximizes some objective function. However, many of these problems have closely related decision problems that are no simpler to solve than the optimization problem. For the purposes of this discussion of reductions and *NP*-completeness, we will restrict our attention to decision problems.

Definition 22.1 Let $A \subseteq \Sigma$ and $B \subseteq \Gamma$ be decision problems. (Here Σ and Γ are the problem domains, and A and B are the "yes" instances.) We write $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and say that A *reduces to* B *in polynomial time* if there is a function $\sigma : \Sigma \to \Gamma$ such that

- σ is computable by a deterministic Turing machine in polynomial time;

- for all problem instances $x \in \Sigma$,

$$x \in A \quad \text{iff} \quad \sigma(x) \in B \; .$$

We write $A \equiv_{\mathrm{m}}^{\mathrm{p}} B$ if both $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and $B \leq_{\mathrm{m}}^{\mathrm{p}} A$. □

The reducibility relation $\leq_{\mathrm{m}}^{\mathrm{p}}$ is often called *polynomial-time many-one* or *Karp* reducibility. The superscript p stands for *polynomial-time*. The subscript m stands for *many-one* and describes the function σ, and is included to distinguish $\leq_{\mathrm{m}}^{\mathrm{p}}$ from another popular polynomial-time reducibility relation $\leq_{\mathrm{T}}^{\mathrm{p}}$, often called *polynomial-time Turing* or *Cook* reducibility. The relation $\leq_{\mathrm{m}}^{\mathrm{p}}$ is stronger than $\leq_{\mathrm{T}}^{\mathrm{p}}$ in the sense that

$$A \leq_{\mathrm{m}}^{\mathrm{p}} B \;\rightarrow\; A \leq_{\mathrm{T}}^{\mathrm{p}} B \; .$$

The formal definition of $\leq_{\mathrm{T}}^{\mathrm{p}}$ involves oracle Turing machines and can be found in [39, pp. 111ff.].

Intuitively, if $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ then A is no harder than B. In particular,

Theorem 22.2 *If $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ and B has a polynomial-time algorithm, then so does A.*

Proof. Given an instance x of the problem A, compute $\sigma(x)$ and ask whether $\sigma(x) \in B$. Note that the algorithm for B runs in polynomial time in the size of its input $\sigma(x)$, which might be bigger than x; but since σ is computable in polynomial time on a Turing machine, the size of $\sigma(x)$ is at most polynomial in the size of x, and the composition of two polynomials is still a polynomial, so the overall algorithm is polynomial in the size of x. □

In the last lecture we showed that CNFSat $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Clique. Below we give some more examples of polynomial-time reductions between problems.

Definition 22.3 (Independent Set) An *independent set* in an undirected graph $G = (V, E)$ is a subset U of V such that $U^2 \cap E = \emptyset$, *i.e.* no two vertices in U are connected by an edge in E. The *independent set problem* is to determine, given $G = (V, E)$ and $k \geq 0$, whether G has an independent set U of cardinality at least k. □

Note that the use of "independent" here is *not* in the sense of matroids.

There exist easy polynomial reductions from/to the clique problem. Consider the complementary graph $\overline{G} = (V, \overline{E})$, where

$$\overline{E} \;=\; \{(u, v) \mid u \neq v,\ (u, v) \notin E\} \; .$$

Then G has a clique of size k iff $\overline{G}$ has an independent set of size k. This simple one-to-one correspondence gives reductions in both directions, therefore Independent Set $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Clique.

Definition 22.4 (Vertex Cover) A *vertex cover* in an undirected graph $G = (V, E)$ is a set of vertices $U \subseteq V$ such that every edge in E is adjacent to some vertex in U. The *vertex cover problem* is to determine, given $G = (V, E)$ and $k \geq 0$, whether there exists a vertex cover U in G of cardinality at most k. □

Again, there exist easy polynomial reductions from/to Independent Set: $U \subseteq V$ is a vertex cover iff $V - U$ is an independent set. Therefore Vertex Cover $\equiv_{\mathrm{m}}^{\mathrm{p}}$ Independent Set.

Definition 22.5 (k-CNFSat) A Boolean formula is in *k-conjunctive normal form (k-CNF)* if it is in conjuctive normal form and has at most k literals per clause. The problem k-CNFSat is just CNFSat with input instances restricted to formulas in k-CNF. In other words, given a Boolean formula in k-CNF, does it have a satisfying assignment? □

In the general CNFSat problem, the number of literals per clause is not restricted and can grow as much as linearly with the size of the formula. In the k-CNFSat problem, the number of literals per clause is restricted to k, independent of the size of the formula. The k-CNFSat problem is therefore a restriction of the CNFSat problem, and could conceivably be easier to solve than CNFSat. It turns out that 2CNFSat (and hence 1CNFSat also) is solvable in linear time, whereas k-CNFSat is as hard as CNFSat for any $k \geq 3$. We prove the latter statement by exhibiting a reduction CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3CNFSat.

Let $\mathcal{B}$ be an arbitrary Boolean formula in CNF. For each clause of the form

$$(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_{m-1} \vee \ell_m) \tag{27}$$

with $m \geq 4$, let $x_1, x_2, \ldots, x_{m-3}$ be new variables and replace the clause (27) in $\mathcal{B}$ with the formula

$$\begin{aligned}(\ell_1 \vee \ell_2 \vee x_1) \wedge (\neg x_1 \vee \ell_3 \vee x_2) \wedge (\neg x_2 \vee \ell_4 \vee x_3) \wedge \cdots \\ \wedge (\neg x_{m-4} \vee \ell_{m-2} \vee x_{m-3}) \wedge (\neg x_{m-3} \vee \ell_{m-1} \vee \ell_m)\ .\end{aligned}$$

Let $\mathcal{B}'$ be the resulting formula. Then $\mathcal{B}'$ is in 3CNF, and $\mathcal{B}'$ is satisfiable iff $\mathcal{B}$ is. This follows from several applications of the following lemma:

Lemma 22.6 *For any Boolean formulas $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$ and variable x not appearing in $\mathcal{C}, \mathcal{D}$, or $\mathcal{E}$, the formula*

$$(x \vee \mathcal{C}) \wedge (\neg x \vee \mathcal{D}) \wedge \mathcal{E} \tag{28}$$

is satisfiable if and only if the formula

$$(\mathcal{C} \vee \mathcal{D}) \wedge \mathcal{E} \tag{29}$$

is satisfiable.

Proof. This is just the *resolution rule* of propositional logic. Any satisfying truth assignment for (28) gives a satisfying truth assignment for (29), since one of x, $\neg x$ is false, so either $\mathcal{C}$ or $\mathcal{D}$ is true. Conversely, in any satisfying truth assignment for (29), one of $\mathcal{C}, \mathcal{D}$ is true. If $\mathcal{C}$, assign $x :=$ *false*. If $\mathcal{D}$, assign $x :=$ *true*. We can assign x freely since it does not appear in $\mathcal{C}, \mathcal{D}$ or $\mathcal{E}$. In either case (28) is satisfied. □

The formula $\mathcal{B}'$ is easily constructed from $\mathcal{B}$ in polynomial time. This constitutes a polynomial-time reduction from CNFSat to 3CNFSat. Furthermore, 3CNFSat is trivially reducible to k-CNFSat for any $k \geq 3$, which in turn is trivially reducible to CNFSat. Since $\leq_m^p$ is transitive, k-CNFSat $\equiv_m^p$ CNFSat for $k \geq 3$.

The problem 2CNFSat is solvable in linear time. In this case the clauses in $\mathcal{B}$ contain at most two literals, and we can assume exactly two without loss of generality by replacing any clause of the form (ℓ) with $(\ell \vee \ell)$. Now we think of every two-literal clause $(\ell \vee \ell')$ as a pair of implications

$$(\neg\ell \rightarrow \ell') \quad \text{and} \quad (\neg\ell' \rightarrow \ell) \,. \tag{30}$$

Construct a directed graph $G = (V, E)$ with a vertex for every literal and directed edges corresponding to the implications (30).

We claim that $\mathcal{B}$ is satisfiable iff no pair of complementary literals both appear in the same strongly connected component of G. Under any satisfying truth assignment, all literals in a strong component of G must have the same truth value. Therefore, if any variable x appears both positively and negatively in the same strong component of G, $\mathcal{B}$ is not satisfiable.

Conversely, suppose that no pair of complementary literals both appear in the same strong component of G. Consider the quotient graph G' obtained by collapsing the strong components of G as described in Lecture 4. As proved in that lecture, the graph G' is acyclic, therefore induces a partial order on its vertices. This partial order extends to a total order. We assign $x :=$ *false* if the strong component of x occurs before the strong component of $\neg x$ in this total order, and $x :=$ *true* if the strong component of $\neg x$ occurs before the strong component of x. It can be shown that this gives a satisfying assignment.

We know how to find the strong components of G in linear time. This gives a linear-time algorithm test for 2CNF satisfiability. We can also produce a satisfying assignment in linear time, if one exists, using topological sort to totally order the strong components.

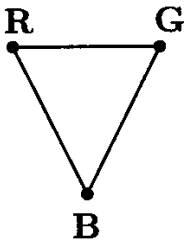
Definition 22.7 (k-Colorability) Let C a finite set of *colors* and $G = (V, E)$ an undirected graph. A *coloring* is a map $\chi : V \rightarrow C$ such that $\chi(u) \neq \chi(v)$ for $(u, v) \in E$. Given G and k, the *k-colorability problem* is to determine whether there exists a coloring using no more than k colors. □

For $k = 2$, the problem is easy: a graph is 2-colorable iff it is bipartite iff it has no odd cycles. This can be checked by BFS or DFS in linear time. We

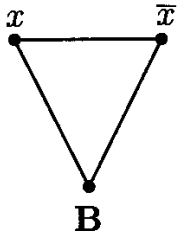
show that for $k = 3$, the problem is as hard as CNFSat by giving a reduction CNFSat $\leq_m^p$ 3-colorability.

Let $\mathcal{B}$ be a Boolean formula in CNF. We will construct a graph G that is 3-colorable iff $\mathcal{B}$ is satisfiable.

There will be three special vertices called **R**, **B**, and **G**, which will be connected in a triangle. In any 3-coloring, they will have to be colored with different colors, so we assume without loss of generality that they are colored red, blue, and green, respectively.

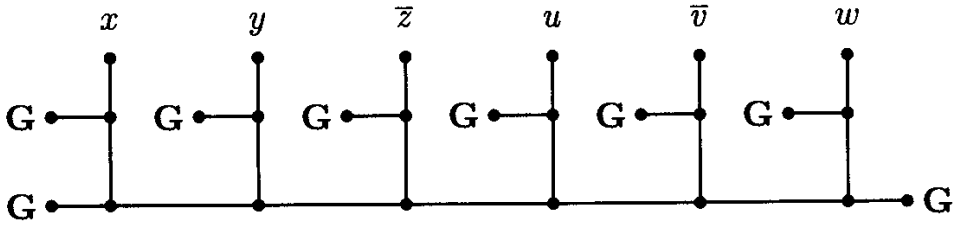


We include a vertex for each literal, and connect each literal to its complement and to the vertex **B** as shown.



In any 3-coloring, the vertices corresponding to the literals x and $\overline{x}$ will have to be colored either red or green, and not both red or both green. Intuitively, a legal 3-coloring will represent a satisfying truth assignment in which the green literals are true and the red literals are false.

To complete the graph, we add a subgraph like the one shown below for each clause in $\mathcal{B}$. The one shown below would be added for the clause $(x \vee y \vee \overline{z} \vee u \vee \overline{v} \vee w)$. The vertices in the picture labeled **G** are all the same vertex, namely the vertex **G**.



This subgraph has the property that a coloring of the vertices on the top row with either red or green can be extended to a 3-coloring of the whole subgraph iff at least one of them is colored green. If all vertices on the top row are colored red, then all the vertices on the middle row adjacent to vertices on the top row must be colored blue. Starting from the left, the vertices along the bottom row must be colored alternately red and green. This will lead to

a conflict with the last vertex in the bottom row. (If the number of literals in the clause is odd instead of even as pictured, then the rightmost vertex in the bottom row is **R** instead of **G**.)

Conversely, suppose one of the vertices on the top row is colored green. Pick one such vertex. Color the vertex directly below it in the middle row red and the vertex directly below that on the bottom row blue. Color all other vertices on the middle row blue. Starting from the left and right ends, color the vertices along the bottom row as forced, either red or green. The coloring can always be completed.

Thus if there is a legal 3-coloring, then the subgraph corresponding to each clause must have at least one green literal, and truth values can be assigned so that the green literals are true. This gives a satisfying assignment. Conversely, if there is a satisfying assignment, color the true variables green and the false ones red. Then there is a green literal in each clause, so the coloring can be extended to a 3-coloring of the whole graph.

From this it follows that $\mathcal{B}$ is satisfiable iff G is 3-colorable, and the graph G can be constructed in polynomial time. Therefore CNFSat $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3-colorability.

One can trivially reduce 3-colorability to k-colorability for $k > 3$ by appending a $k-3$ clique and edges from every vertex of the $k-3$ clique to every other vertex.

One may be tempted to conclude that in problems like k-CNFSat and k-colorability, larger values of k always make the problem harder. On the contrary, we shall see in the next lecture that the k-colorability problem for planar graphs is easy for $k \leq 2$ and $k \geq 4$, but as hard as CNFSat for $k = 3$.

Lecture 23 More NP-Complete Problems

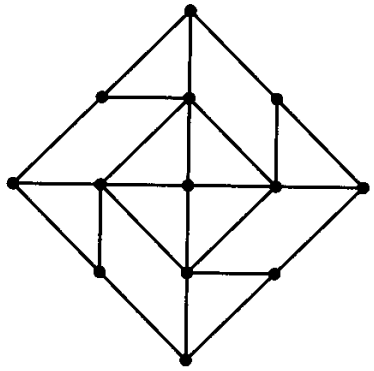
23.1 Planar Graph Colorability

Often in problems with a parameter k like k-CNFSat and k-colorability, larger values of k make the problem harder. This is not always the case. Consider the problem of determining whether a *planar* graph has a k-coloring. The problem is trivial for $k = 1$, easy for $k = 2$ (check by DFS or BFS whether the graph is bipartite, *i.e.* has no odd cycles), and trivial for $k = 4$ or greater by the Four Color Theorem, which says that every planar graph is 4-colorable. This leaves $k = 3$. We show below that 3-colorability of planar graphs is no easier than 3-colorability of arbitrary graphs. This result is due to Garey, Johnson, and Stockmeyer [40]; see also Lichtenstein [72] for some other NP-completeness results involving planar graphs.

We will reduce 3-colorability of an arbitrary graph to the planar case. Given an undirected graph $G = (V, E)$, possibly nonplanar, embed the graph in the plane arbitrarily, letting edges cross if necessary. We will replace each

edge crossing with the planar widget W shown below.

W =

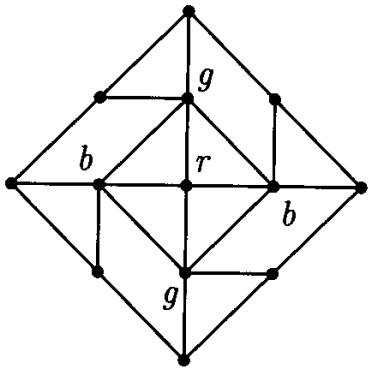


The widget W is a planar graph with the following interesting properties:

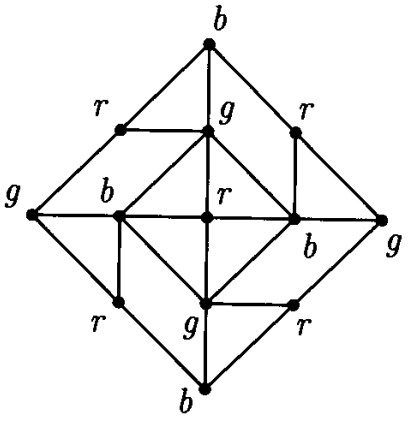
(i) in any legal 3-coloring of W, the opposite corners are forced to have the same color;

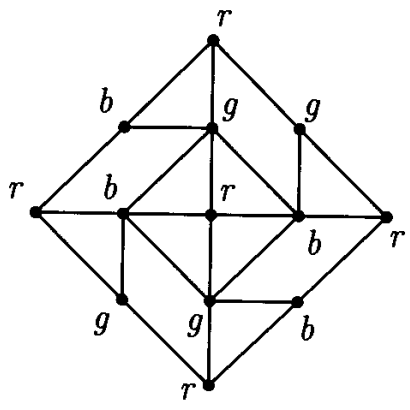
(ii) any assignment of colors to the corners such that opposite corners have the same color extends to a 3-coloring of all of W.

To see this, color the center of W red; then the vertices adjacent to the center must be colored blue or green alternately around the center, say



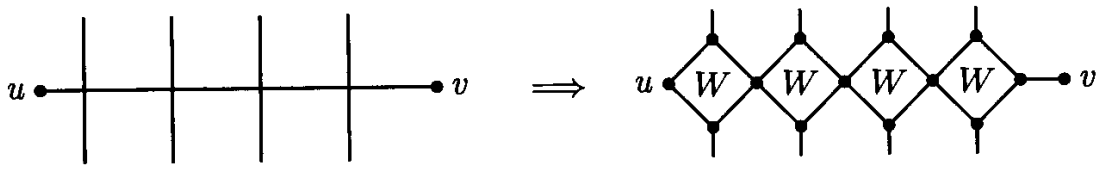
Now the northeast vertex can be colored either red or green. In either case, the colors of all the remaining vertices are forced (proceed counterclockwise to obtain the left hand coloring and clockwise to obtain the right hand coloring):





All other colorings are obtained from these by permuting the colors.

For each edge (u, v) in E, replace each point at which another edge crosses (u, v) in the embedding with a copy of W. Identify the adjacent corners of these copies of W and identify the outer corners of the extremal copies with u and v, all except for one pair, which are connected by an edge. The following diagram illustrates an edge (u, v) with four crossings before and after this operation. In this diagram, the copy of W closest to v is connected to v by an edge, and all other adjacent corners of copies of W are identified.



The resulting graph $G' = (V', E')$ is planar. If

$$\chi : V' \rightarrow \{\text{red, blue, green}\}$$

is a 3-coloring of G', then property (i) of W implies that χ restricted to V is a 3-coloring of G. Conversely, if $\chi : V \rightarrow \{\text{red, blue, green}\}$ is a 3-coloring of G, then property (ii) of W allows χ to be extended to a 3-coloring of G'.

We have given a reduction of the 3-colorability problem for an arbitrary graph to the same problem restricted to planar graphs. Thus the latter problem is as hard as the former.

23.2 *NP*-Completeness

The following definitions lay the foundations of the theory of *NP*-completeness. More detail can be found in [3, 39].

We fix once and for all a finite alphabet Σ consisting of at least two symbols. From now on, we take Σ to be the problem domain, and assume that instances of decision problems are encoded as strings in Σ^* in some reasonable way.

Definition 23.1 The complexity class *NP* consists of all decision problems $A \subseteq \Sigma^*$ such that A is the set of input strings accepted by some polynomial-time-bounded nondeterministic Turing machine. The complexity class P consists of all decision problems $A \subseteq \Sigma^*$ such that A is the set of input strings accepted by some polynomial-time-bounded deterministic Turing machine. □

Note that $P \subseteq NP$ since every deterministic machine is a nondeterministic one that does not happen to make any choices. It is not known whether $P = NP$; this is arguably the most important outstanding open problem in computer science.

Definition 23.2 The set A is *NP-hard (with respect to the reducibility relation* $\leq_m^p$*)* if $B \leq_m^p A$ for all $B \in NP$. □

Theorem 23.3 *If A is NP-hard and $A \in P$, then $P = NP$.*

Proof. For any $B \in NP$, compose the polynomial-time algorithm for A with the polynomial-time function reducing B to A to get a polynomial time algorithm for B. □

Definition 23.4 The set A is *NP-complete* if A is NP-hard and $A \in NP$. □

Theorem 23.5 *If A is NP-complete, then*

$$A \in P \leftrightarrow P = NP .$$

Definition 23.6 The complexity class *coNP* is the class of sets $A \subseteq \Sigma^*$ whose complements $\overline{A} = \Sigma^* - A$ are in NP. A set B is *coNP-hard* if every problem in *coNP* reduces in polynomial time to B. It is *coNP-complete* if in addition it is in *coNP*. □

The following theorem is immediate from the definitions.

Theorem 23.7

1. *$A \leq_m^p B$ iff $\overline{A} \leq_m^p \overline{B}$.*
2. *A is NP-hard iff $\overline{A}$ is coNP-hard.*
3. *A is NP-complete iff $\overline{A}$ is coNP-complete.*
4. *If A is NP-complete then $A \in coNP$ iff $NP = coNP$.*

It is unknown whether $NP = coNP$.

We will show later that the problems CNFSat, 3CNFSat, Clique, Vertex Cover, and Independent Set, which we have shown to be $\equiv_m^p$-equivalent, are all in fact NP-complete.

23.3 More *NP*-complete problems

Before we prove the NP-completeness of the problems we have been considering, let us consider some more problems in this class. Some of these problems, such as Traveling Salesman, Bin Packing, and Integer Programming, are very natural and important in operations research and industrial engineering. We start with the *exact cover problem.*

Definition 23.8 (Exact Cover) Given a finite set X and a family of subsets S of X, is there a subset $S' \subseteq S$ such that every element of X lies in exactly one element of S'? □

We show that the problem Exact Cover is *NP*-hard by reduction from the problem of 3-colorability of undirected graphs. See [39] for a different approach involving the 3-dimensional matching problem.

Lemma 23.9 *3-Colorability* $\leq_m^p$ *Exact Cover.*

Proof. Suppose we are given an undirected graph $G = (V, E)$. We show how to produce an instance (X, S) of the exact cover problem for which an exact cover exists iff G has a 3-coloring.

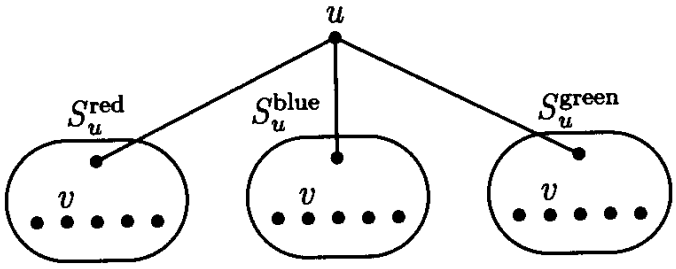
Let $C = \{\text{red, blue, green}\}$. For each $u \in V$, let $N(u)$ be the set of neighbors of u in G. Since G is undirected, $u \in N(v)$ iff $v \in N(u)$.

For each $u \in V$, we include u in X along with $3(|N(u)| + 1)$ additional elements of X. These $3(|N(u)| + 1)$ additional elements are arranged in three disjoint sets of $|N(u)| + 1$ elements each, one set corresponding to each color. Call these three sets S_u^{red}, S_u^{blue}, S_u^{green}. For each color $c \in C$, pick a special element p_u^c from S_u^c and associate the remaining $|N(u)|$ elements of S_u^c with the elements of $N(u)$ in a one-to-one fashion. Let q_{uv}^c denote the element of S_u^c associated with $v \in N(u)$.

The set S will contain all two element sets of the form

$$\{u, p_u^c\} \tag{31}$$

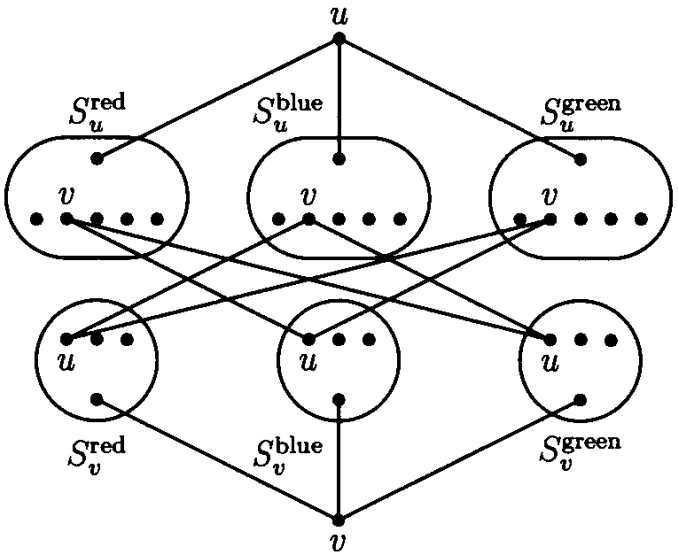
for $u \in V$ and $c \in C$, as well as all the sets S_u^c for $u \in V$ and $c \in C$. Here is a picture of what we have so far for a vertex u of degree 5 with $v \in N(u)$. The ovals represent the three sets S_u^c and the lines represent the three two-element sets (31).



To complete S, we include all two element sets of the form

$$\{q_{uv}^c, q_{vu}^{c'}\} \tag{32}$$

for all $(u, v) \in E$ and $c, c' \in C$ with $c \neq c'$. Here is a picture showing a part of the construction for two vertices u and v of degrees 5 and 3 respectively, where (u, v) in E. The six lines in the center represent the two-element sets (32).



We now argue that the instance (X, S) of Exact Cover just constructed is a "yes" instance, *i.e.* an exact cover $S' \subseteq S$ of X exists, iff the graph G has a 3-coloring. Suppose first that G has a 3-coloring $\chi : V \to C$. We construct an exact cover $S' \subseteq S$ as follows. For each vertex u, let S' contain the sets $\{u, p_u^{\chi(u)}\}$ and S_u^c for $c \neq \chi(u)$. This covers everything except points of the form $q_{uv}^{\chi(u)}$, where $(u, v) \in E$. For each edge (u, v), let S' also contain the set $\{q_{uv}^{\chi(u)}, q_{vu}^{\chi(v)}\}$. This set is in S since $\chi(u) \neq \chi(v)$. This covers all the remaining points, and each point is covered by exactly one set in S'.

Conversely, suppose S' is an exact cover. Each u is covered by exactly one set in S', and it must be of the form $\{u, p_u^c\}$ for some c. Let $\chi(u)$ be that c; we claim that χ is a valid coloring, *i.e.* that if $(u, v) \in E$ then $\chi(u) \neq \chi(v)$. For each u, since $\{u, p_u^{\chi(u)}\} \in S'$, we cannot cover p_u^c for $c \neq \chi(u)$ by any set of the form (31), since u is already covered; therefore they must be covered by the sets S_u^c, which are the only other sets containing the points p_u^c. The sets $\{u, p_u^{\chi(u)}\}$ and S_u^c, $c \neq \chi(u)$ cover all points except those of the form $q_{uv}^{\chi(u)}$, $(u, v) \in E$. The only way S' can cover these remaining points is by the sets (32). By construction of S, these sets are of the form $\{q_{uv}^{\chi(u)}, q_{vu}^{\chi(v)}\}$ for $(u, v) \in E$ and $\chi(u) \neq \chi(v)$. □

Lecture 24 Still More *NP*-Complete Problems

In this lecture we use the basic *NP*-complete problems given in previous lectures, which may have appeared contrived, to show that several very natural and important decision problems are *NP*-complete.

We first consider a collection of problems with many applications in operations research and industrial engineering.

Definition 24.1 (Knapsack) Given a finite set S, integer weight function $w : S \to \mathcal{N}$, benefit function $b : S \to \mathcal{N}$, weight limit $W \in \mathcal{N}$, and desired benefit $B \in \mathcal{N}$, determine whether there exists a subset $S' \subseteq S$ such that

$$\begin{aligned} \sum_{a \in S'} w(a) &\leq W \\ \sum_{a \in S'} b(a) &\geq B \,. \end{aligned}$$

□

The name is derived from the problem of trying to decide what you really need to take with you on your camping trip. For another example: you are the coach of a crew team, and you wish to select a starting squad of rowers with a combined weight not exceeding W and combined strength at least B.

Definition 24.2 (Subset Sum) Given a finite set S, integer weight function $w : S \to \mathcal{N}$, and target integer B, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) \;=\; B \;?$$

□

Definition 24.3 (Partition) Given a finite set S and integer weight function $w : S \to \mathcal{N}$, does there exist a subset $S' \subseteq S$ such that

$$\sum_{a \in S'} w(a) \;=\; \sum_{a \in S - S'} w(a) \;?$$

□

Trivially, Partition reduces to Subset Sum by taking

$$B \;=\; \frac{1}{2} \sum_{a \in S} w(a) \;.$$

Also, Subset Sum reduces to Partition by introducing two new elements of weight $N - B$ and $N - (\Sigma - B)$, respectively, where

$$\Sigma \;=\; \sum_{a \in S} w(a)$$

and N is a sufficiently large number (actually $N > \Sigma$ will do). The number N is chosen large enough so that both new elements cannot go in the same partition element, because together they outweigh all the other elements. Now we ask whether this new set of elements can be partitioned into two sets of equal weight (which must be N). By leaving out the new elements, this gives a partition of the original set into two sets of weight B and $\Sigma - B$.

Both Subset Sum and Partition reduce to Knapsack. To reduce Partition to Knapsack, take $b = w$ and $W = B = \frac{1}{2}\Sigma$.

We show that these three problems are as hard as Exact Cover by reducing Exact Cover to Subset Sum. Assume that $X = \{0, 1, \ldots, m-1\}$ in the given instance (X, S) of Exact Cover. For $x \in X$, define

$$\#x \;=\; |\{A \in S \mid x \in A\}| \;,$$

the number of elements of S containing x. Let p be a number exceeding all $\#x$, $0 \leq x \leq m-1$. Encode $A \in S$ as the number

$$w(A) \;=\; \sum_{x \in A} p^x$$

and take

$$B \;=\; \sum_{x=0}^{m-1} p^x \;=\; \frac{p^m - 1}{p - 1} \;.$$

In p-ary notation, $w(A)$ looks like a string of 0's and 1's with a 1 in position x for each $x \in A$ and 0 elsewhere. The number B in p-ary notation looks like a string of 1's of length m. Adding the numbers $w(A)$ simulates the union of the sets A. The number p was chosen big enough so that we do not get into trouble with carries. Asking whether there is a subset sum that gives B is the same as asking for an exact cover of X.

The *bin packing problem* is an important problem that comes up in industrial engineering and computer memory management.

Definition 24.4 (Bin Packing) Given a finite set S, volumes $w : S \to \mathcal{N}$, and bin size $B \in \mathcal{N}$, what is the minimum number of bins needed to contain all the elements of S? Expressed as a decision problem, given the above data and a natural number k, does there exist a packing into k or fewer bins? □

We can easily reduce Partition to Bin Packing by taking B to be half the total weight of all elements of S and $k = 2$.

An extremely important and general problem in operations research is the integer programming problem.

Definition 24.5 (Integer Programming) Given rational numbers a_{ij}, c_j, and b_i, $1 \leq i \leq m$, $1 \leq j \leq n$, find integers $x_1, x_2, \ldots, x_n$ that maximize the linear function

$$\sum_{j=1}^{n} c_j x_j$$

subject to the linear constraints

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i , \quad 1 \leq i \leq m . \tag{33}$$

The corresponding decision problem is to test whether there exists a point with integer coordinates in a region defined by the intersection of half-spaces: given a_{ij} and b_i, $1 \leq i \leq m$, $1 \leq j \leq n$, test whether there exists an integer point $x_1, \ldots, x_n$ in the region (33). □

In *linear programming*, the x_i's are not constrained to be integers, but may be real. The linear programming problem was shown to be solvable in polynomial time in 1980 by Khachian [60] using a method that has become known as the *ellipsoid method*. In 1984, a more efficient polynomial time algorithm was given by Karmarkar [56]; his method has become known as the *interior point method*. Since that time, several refinements have appeared [90, 102]. The older *simplex method*, originally due to Dantzig (see [19]), is used successfully in practice but is known to be exponential in the worse case.

The integer programming problem is *NP*-hard, as the following reduction from Subset Sum shows: the instance of Subset Sum consisting of a set S with

weights $w : S \to \mathcal{N}$ and threshold B has a positive solution iff the Integer Programming instance

$$0 \leq x_a \leq 1, \quad a \in S$$

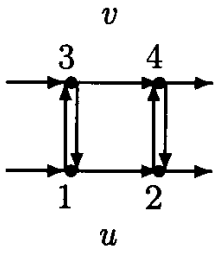
$$\sum_{a \in S} w(a)x_a = B$$

has an integer solution. It is also possible to show that Integer Programming is in *NP* by showing that if there exists an integer solution, then there exists one with only polynomially many bits as a function of the size of the input (n, m, and number of bits in the a_{ij}, b_i, and c_j) [16]. The integer solution can then be guessed and verified in polynomial time.

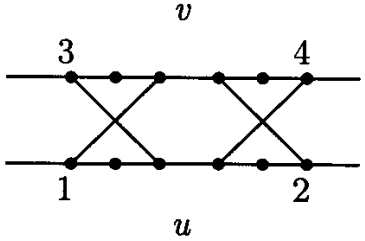
Definition 24.6 (Hamiltonian Circuit) A *Hamiltonian circuit* in a directed or undirected graph $G = (V, E)$ is a circuit that visits each vertex in the graph exactly once. It is like an Euler circuit, except the constraint is on vertices rather than edges. The *Hamiltonian circuit problem* is to determine for a given graph G whether a Hamiltonian circuit exists. □

We reduce Vertex Cover to Hamiltonian Circuit. Recall that a *vertex cover* in an undirected graph $G = (V, E)$ is a set of vertices $U \subseteq V$ such that every edge in E is adjacent to some vertex in U. The *vertex cover problem* is to determine, given $G = (V, E)$ and $k \geq 0$, whether there exists a vertex cover U in G of cardinality at most k.

We will build a graph H which will have a Hamiltonian circuit iff G has a vertex cover of size k. The main building block of H for the directed Hamiltonian circuit problem is a widget consisting of four vertices connected as shown.



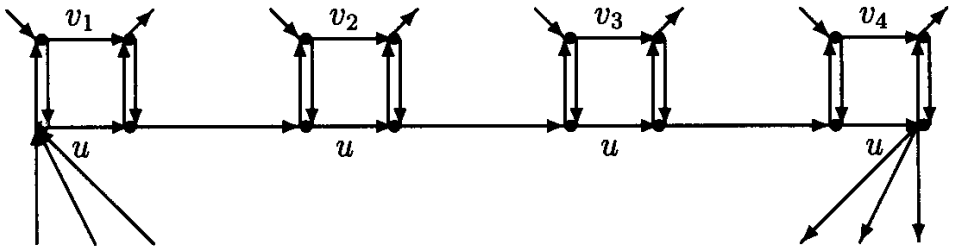
For the undirected version, we use an undirected widget with twelve vertices:



There is one widget corresponding to each edge $(u, v) \in E$. In the widget corresponding to the edge (u, v), one side corresponds to the vertex u and the other to the vertex v.

These widgets have the following interesting property: any Hamiltonian circuit that enters at vertex 1 must leave at vertex 2, and there are only two ways to pass through, either straight through or in a zigzag pattern that crosses to the other side and back. If it goes straight through, then all the vertices on the u side and none of the vertices on the v side are visited. If it crosses to the other side and back, then all the vertices on both sides are visited. Any other path through the widget leaves some vertex stranded, so the path could not be a part of a Hamiltonian circuit. Thus any Hamiltonian circuit that enters at 1 either picks up the vertices in the widget all at once using the zigzag path, or goes straight through and picks up only the vertices on one side, then re-enters at 3 later on to pick up the vertices on the other side.

The graph H is formed as follows. For each vertex u, we string together end-to-end all the u sides of all the widgets corresponding to edges in E incident to u. Call this the *u loop*. In addition, H has a set K of k extra vertices, where k is the parameter of the given instance of Vertex Cover denoting the size of the vertex cover we are looking for. There is an edge from each vertex in K to the first vertex in the u loop, and an edge from the last vertex in the u loop to each vertex in K.



We now show that there is a vertex cover of size k in G iff H has a Hamiltonian circuit. Suppose there is a vertex cover $\{u_1, \ldots, u_k\}$ of G of size k. Then H has a Hamiltonian circuit: starting from the first vertex of K, go through the u_1 loop. When passing through the widget corresponding to an edge (u_1, v) of G, take the straight path if v is in the vertex cover, *i.e.* if $v = u_j$ for some j (the other side of the widget will be picked up later when we traverse the u_j loop), and take the zigzag path if v is not in the vertex cover. When leaving the u_1 loop, go to the second vertex of K, then through the u_2 loop, and so on, all the way around and back to the first vertex of K.

Conversely, if H has a Hamiltonian circuit, the number of u loops traversed must be exactly k, and that set of vertices u forms a vertex cover of G.

This argument holds for both the directed and undirected case. Thus, determining the existence of a Hamiltonian circuit in a directed or undirected

graph is *NP*-hard. It is also in *NP*, since a Hamiltonian circuit can be guessed and verified in polynomial time.

Finally, we consider the *Traveling Salesman Problem (TSP)*. The optimization version of this problem asks for a tour through a set of cities minimizing the total distance. There are several versions of TSP, depending on the properties of the graph and distance function and the type of tour desired. We consider here a quite general formulation.

Definition 24.7 (Traveling Salesman (TSP)) Given a number $k \geq 0$ and a directed graph $G = (V, E)$ with nonnegative edge weights $w : E \to \mathcal{N}$, does there exist a tour of total weight at most k visiting every vertex at least once and returning home? □

Garey and Johnson [39] use a slightly more restricted version which asks for a tour visiting each vertex exactly once. We prefer the more general version above, since to get anywhere from Ithaca and back usually involves at least two stops in Pittsburgh.

TSP is in *NP* provided we can argue that optimal tours are short enough that they can be guessed and verified in polynomial time. Each vertex can be visited at most n times in an optimal tour, because otherwise we could cut out a loop and still visit all vertices. We can thus guess a tour of length at most n^2 and verify that its total weight is at most k.

TSP is *NP*-hard, since there is a straightforward reduction from Hamiltonian Circuit: give all edges unit weight and ask for a TSP tour of weight n.

Combining arguments from the last several lectures, we have:

Theorem 24.8 *The CNF Satisfiability problem reduces via* $\leq_m^p$ *to all the following problems: Knapsack, Partition, Subset Sum, Exact Cover, Bin Packing, Integer Programming, directed and undirected Hamiltonian Circuit, and Traveling Salesman.*