CMPSCI 611 Advanced Algorithms **Fall 2000**

Lecture 19: November 16

Lecturer: Micah Adler *Scribes: Charles Sutton and Yoshiya Kinuta*

19.1 Max-Cut Problem

We want to prove that the max-cut problem is **NP–complete**. To do this, we'll first prove that a problem called not-all-equal 3-SAT (NA= 3-SAT) is **NP–complete**. We'll then reduce NA= 3-SAT to max-cut.

19.2 Not All = 3-SAT (NA = 3-SAT)

The NA= 3-SAT problem is defined as follows.

INPUT: A 3-CNF formula Φ.
QUESTION: Is there a satisfying assignment to Φ that has ≥ 1 false literal per clause?

Theorem 19.1 *NA= 3-SAT is* **NP–complete***.*

Proof: We can verify that an assignment is NA= satisfying in polynomial time. So NA= 3-SAT is in **NP**.

Now we show that 3-SAT $\leq_p$ NA= 3-SAT.

We transform an input Φ to 3-SAT problem into an input $\Phi^{'}$ to NA= 3-SAT. For each clause $(l_{i1} \vee l_{i2} \vee l_{i3})$, we add two variables x_i and y_i and three clauses as follows. (The variable α is another one we introduce that we share between clauses.)

$$\Phi = \ldots \wedge (l_{i1} \vee l_{i2} \vee l_{i3}) \wedge \ldots$$
$$\Phi^{'} = \ldots \wedge (l_{i1} \vee l_{i2} \vee x_i) \wedge (\neg x_i \vee l_{i3} \vee y_i) \wedge (x_i \vee y_i \vee \alpha) \ldots$$

Note that the transformation from Φ to $\Phi^{'}$ can be done in polynomial time because there is constant amount of work being done per clause.

Claim 19.2 *If Φ is satisfiable, then $\Phi^{'}$ is NA= satisfiable.*

Proof: Given a satisfying assignment to Φ, we construct a satisfying assignment to $\Phi^{'}$. Set α = true. Ignoring the case where l_{i1}, l_{i2}, and l_{i3} are all false, there are 7 possible settings of l_{i1}, l_{i2}, and l_{i3}. Examine all possible settings. In each setting, $\Phi^{'}$ is satisfied.

ex. if l_{i1}, l_{i2}, and l_{i3} are all *true*,

$l_{i1} \quad = \textit{true}$

$$\begin{aligned}
l_{i1} &= \textit{true} \\
l_{i1} &= \textit{true} \\
x_i &= \textit{false} \\
\neg x_i &= \textit{true} \\
y_i &= \textit{false} \\
\alpha &= \textit{true}
\end{aligned}$$

■

Claim 19.3 *If $\Phi^{'}$ is NA-satisfiable, then Φ is satisfiable.*

Proof: Assume $\alpha = \textit{true}$ ($\Phi^{'}$ has at least 1 *true* and and 1 *false* per clause. If α is *false* in $\Phi^{'}$, we can flip it to *true*, and some other literal in the same clause that is *true* to *false*).

At least one of l_{i1}, l_{i2}, or l_{i3} must be true. ■

This concludes the proof that NA= 3-SAT is **NP–complete**. ■

19.3 Proof that Max-Cut is NP–complete

Recall that the decision version of max-cut is as follows.

INPUT: A graph G and an integer k.
QUESTION: Does G have a cut of size at least k?

Theorem 19.4 *Max-cut is* **NP–complete**.

Proof: Given binary string X, it is easy to verify in polynomial time that X describes a valid cut in G of size k. So max-cut is in **NP**.

We now show that max-cut is **NP–hard**. We do this by reducing NA= 3-SAT to max-cut. Let ϕ be an input to NA= 3-SAT. First, to simplify the reduction, we convert ϕ into a restricted but equivalent form. Convert ϕ into a formula $\bar{\phi}$ such that

1. No clause has the form $(A \vee \neg A \vee B)$, where A and B are any literals. This is easy to ensure because the clause will always be satisfied, so we can simply remove it from $\bar{\phi}$.
2. No two clauses share a pair of literals. That is, no two clauses have the form $(A \vee B \vee C)$ and $(A \vee B \vee D)$, for literals A, B, C, and D. If ϕ contains any clauses of this form, we can introduce two variables X and Y and substitute for $(A \vee B \vee C) \wedge (A \vee B \vee D)$ the equivalent expression $(A \vee B \vee C) \wedge (A \vee X \vee D) \wedge (B \vee \bar{X} \vee Y) \wedge (\bar{B} \vee X \vee \bar{Y})$.

So $\bar{\phi}$ is NA= satisfiable if and only if ϕ is NA= satisfiable.

Now, we can use $\bar{\phi}$ to construct an input to the max cut problem. Define a graph G_ϕ as follows: add two vertices X_i and $\bar{X}_i$ for every variable in $\bar{\phi}$. Add an edge between two vertices X_i and X_j if and only if either (a) X_i and X_j occur in the same clause, or (b) $X_i = \neg X_j$. Let $k_\phi = 2m + v$, where m is the number of clauses in $\bar{\phi}$ and v is the number of variables in $\bar{\phi}$.

Here is an example of this reduction. Suppose $\phi = (A \vee \bar{B} \vee C) \wedge (\bar{A} \vee \bar{B} \vee \bar{C})$. Then G_ϕ is given in figure 19.1 and $k_\phi = v + 2m = 6$.

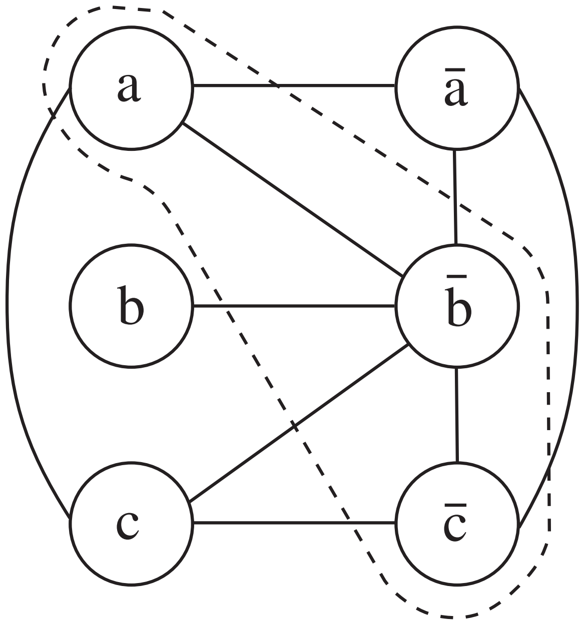

Figure 19.1: Reduction of NA=3-SAT to max-cut. The dashed region is a maximum cut of the graph.

Clearly, G_ϕ and k_ϕ can be calculated in polynomial time.

So we need to prove that this is a valid reduction. First, let us show that if ϕ is NA= satisfiable, then G_ϕ has a cut of size k_ϕ. Suppose that ϕ has some satisfying assignment. Construct a cut in G_ϕ as follows. Put the vertices that correspond to true literals one side of the cut, and other vertices on the other side.

The cut will cross one edge for each variable in ϕ. Also, for each clause, two literals will be on one side of the cut and one literal will be on the other, becuase ϕ is NA= satisfied. Therefore, the cut will cross 2 edges for each clause. So the size of the cut is $v + 2m$.

Second, we show that if G_ϕ has a cut of size k_ϕ, then ϕ is NA= satisfiable. We added one edge in the graph for every variable, and three for every clause. None of these edges are shared between clauses because we specified that no pairs of literals appears in two different clauses. In addition, no clause shares an edge with a variable because we specified that no variable and its negation appears in the same clause. So the number of edges in the graph is $v + 3m$.

Each variable can contribute at most one edge to the cut. Each clause can contribute at most two vertices to the cut. So the maximum cut possible has size $v + 2m$. If there exists a cut of this size, then it must cut every clause and every variable.

This means that we can construct a NA= satisifying assignment for ϕ by choosing either side of the cut to represent true literals. ■

19.4 Other NP–complete problems

19.4.1 A chart

Often we find that problems that seem similar are in different complexity classes. The following chart lists some problems in **P**, with **NP–complete** problems that seem similar at first glance. (Some of the problems below will be defined in the next section.)

P	**NP–complete**
min-cut	max-cut min-bisection max-bisection
2-SAT	3-SAT
shortest-path Eulerian path	longest-path Hamiltonian path
matchings	3-D matchings
MST	Steiner tree
edge-cover	vertex-cover

The moral of this table is that looks can be deceiving. Just becuase an problem is similar to a **NP–complete** problem (or a problem in **P**) doesn't mean that it is. A polynomial time algorithm or a reduction is necessary to be sure.

19.4.2 Some NP–complete problems

All of the following problems are **NP–complete**. A good reference that lists hundreds of **NP–complete** problems is [GJ79].

19.4.2.1 Max-bisection

Definition 19.5 *A* **bisection** $(A, V - A)$ *of a graph* $G = (V, E)$ *is a partition of* V *such that* $|A| = \lfloor |V| / 2 \rfloor$.

Definition 19.6 *The* **size of a bisection** $(A, V - A)$ *is the number of edges that cross the cut defined by* $(A, V - A)$.

INPUT: A graph G and an integer k.

QUESTION: Does G have a bisection of size at least k?

19.4.2.2 Min-Bisection

INPUT: A graph G and an integer k.

QUESTION: Does G have a bisection of size at most k?

This problem can be proven by reduction from the max-bisection problem using the complement of G.

19.4.2.3 Hamiltonian Path

INPUT: A graph G, and two vertices $a, b \in V_G$.

QUESTION: Is there a path from a to b that goes through every node exactly once?

19.4.2.4 3D Matching

The 3D matching problem is similar to the matching problem, except that each "edge" is incident to three vertices instead of two.

INPUT: A set of vertices V, a set of triples $E \subseteq V^3$, and an integer k.

QUESTION: Is there a subset of triples of size k such that no vertex appears in more than one triple?

An example of a 3D matching problem is given in figure 19.2. Each triple in figure 19.2 is represented by a loop enclosing three vertices. The maximum matching has size 4.

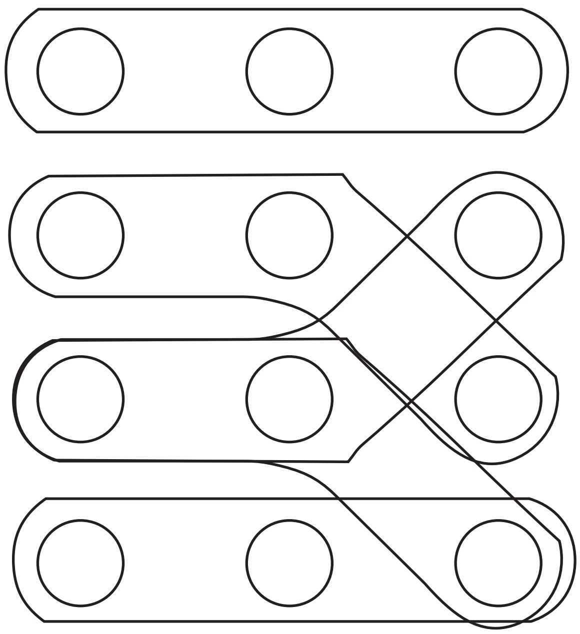

Figure 19.2: A 3D matching problem

19.4.2.5 Steiner Tree

INPUT: A graph $G = (V, E)$, a subset $R \subseteq V$, and an integer k.

QUESTION: Is there a tree in G with at most k edges that includes all of the vertices in R?

19.5 Structure of NP

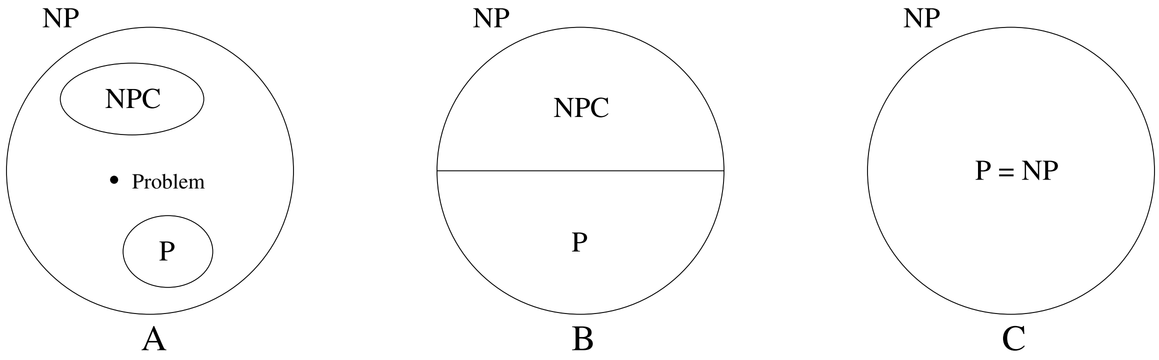

Figure 19.3: Possible structures of **NP**

We know that **B** is not possible because if $\mathbf{P} \neq \mathbf{NP}$, there is an infinite hierarchy of problems from **P** to **NPC**. Nonetheless, most natural problems have been shown either to be in **P** or **NP–complete**.

Exceptions:

1. Factoring, discrete log, primality testing;
2. **Graph Isomorphism**
   INPUT: Two graphs G and G'.
   QUESTION: Is there a permutation of vertices σ such that $\sigma(G) = G'$?

References

GJ79 M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.