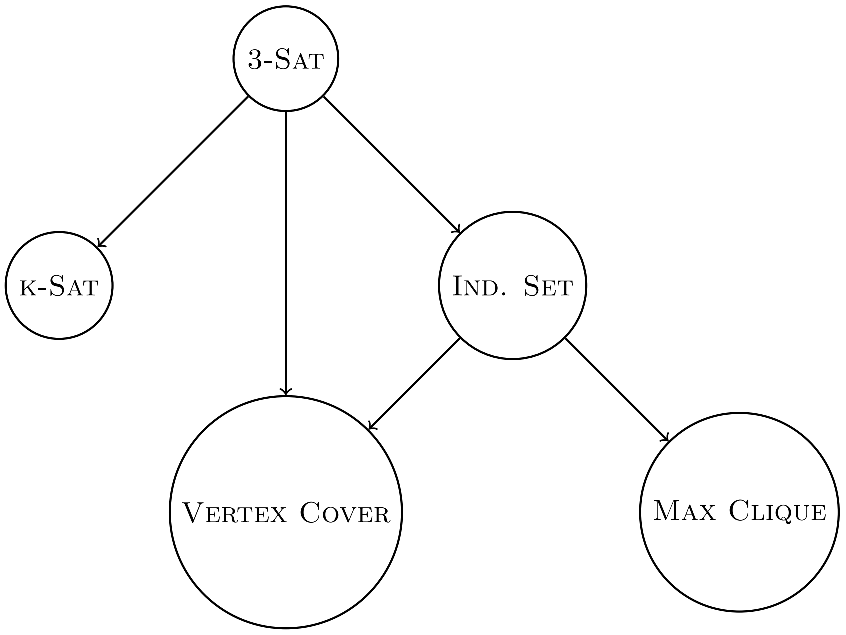
CS 3510 Algorithms Last updated: November 7, 2023

Lecture 22: k-SAT and Approximate Algorithms

Lecturer: Sahil Singla *Scribe(s): Allen Chang*

We'll begin with a quick recap in what we have seen in previous lectures:



First, we showed the NP-Completeness of 3-SAT with the Cook-Levin Theorem. We then reduced 3-SAT to INDEPENDENT SET, then INDEPENDENT SET to VERTEX COVER. Homework 7 shows 3-SAT to VERTEX COVER directly, and Homework 8 will reduce MAX CLIQUE to a problem called the SUBGRAPH ISOMORPHISM problem.

Today, we will first show the equivalence between 3-SAT and k-SAT.

1 Equivalence between 3-SAT and k-SAT

Recall the definition of the k-SAT problem. We have n boolean variables $x_1, x_2, \ldots, x_n$ and m clauses $C_i = (x_{i_1} \vee x_{i_2} \vee \cdots \vee x_{i_k})$. We are asked to find an assignment of each x_i such that all clauses are simultaneously satisfied.

Lemma. 3-SAT $\rightarrow$ k-SAT for $k \geq 3$.

Proof. To motivate intuition, we will provide a constructive proof with $k = 4$ as an example. For each clause $C_i = (x_1 \vee \overline{x_2} \vee x_3)$, add a dummy variable y. Then, replace C_i with

$$(x_1 \vee \overline{x_2} \vee x_3 \vee y) \wedge (x_1 \vee \overline{x_2} \vee x_3 \vee \overline{y})$$

and we are done. Irrespective of whether we satisfy y to be true or false, it must be that at least one of the new clauses will evaluate to false. As a result, fundamentally, the problem has not changed.

For general k, we can provide a similar construction. Instead of adding only one variable, add variables $y_{i_1}, y_{i_2}, \ldots$ for each C_i as necessary to make the length of each clause exactly k. Then, replace C_i with $(C_i \vee y_{i_1} \vee \ldots) \wedge (C_i \wedge \overline{y_{i_1}} \vee \ldots)$. Clearly, this new construction is satisfiable if and only if the original clauses $C_1 \wedge \cdots \wedge C_m$ were satisfiable, and we are done.

Lemma. k-Sat $\rightarrow$ 3-Sat.

Proof. Now, the key idea is that we need to take a clause with k variables and somehow reduce it down to clause(s) with exactly 3 variables.

To do this, we first take a clause $C = (x_1 \vee x_2 \vee \cdots \vee x_k)$. In the 3-Sat problem, add boolean variables $y_1, y_2, \ldots, y_{k-3}$. Then, replace clause C with $k-2$ new clauses in the following way:

$$C = (x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge \cdots \wedge (\overline{y_i} \vee x_{i+2} \vee y_{i+1}) \wedge \ldots$$

Now, if there were originally n variables and m clauses, we have a 3-Sat problem instance with $n + m(k-3)$ variables and $m(k-2)$ clauses. We claim that the new set of clauses are satisfiable if and only if the original set of clauses were satisfiable.

To see this, we can see how we can satisfy the new set of clauses. In the original clause, one of x_i must be true. Take this true variable, and we start "chaining" the true/falses such that y_1 is true, y_2 is false, y_3 is true, etc. (alternating) so that each clause in the new set of clauses has at least one $\overline{y_i}, y_{i+1}$ that is set to true.

The details of the other direction are shown in the textbook, but it has a similar idea. Importantly, we have shown that 3-Sat is *exactly as hard* as k-Sat. Intuitively, one may think that k-Sat is harder than 3-Sat, but this reduction shows that they are of the same difficulty.

2 Beyond NP-Hardness

In practice, we may want to solve some of these NP-Hard problems, even if they are computationally difficult; of course, however, there does not exist (currently) a polynomial time algorithm to solve them. If we want to compute even an approximate solution in polynomial time, however, we have to give up something:

1. *Polytime.* We can use more than polynomial time (maybe we are alright with an exponential time algorithm), or

2. *Optimality.* We can compute an optimal solution (maybe not all of the clauses in a 3-Sat problem needs to be solved, and an approximation suffices), or

3. *Generality.* We can compute a solution for a problem if we make some assumptions (maybe we assume a graph is a tree).

The focus of today's lecture will be ways to give up optimality in order to produce polynomial time approximation algorithms for NP-Hard problems.

3 Approximation Algorithms

Consider any optimization problem. The goal in such a problem is to find the maximum or minimum solution for an objective such that certain constraints are met.

For a max-problem, we can say we have a α-approximation algorithm ($\alpha \geq 1$) if the solution value for this algorithm is guaranteed to be at least $\frac{1}{\alpha}$ times the maximum ("optimal") value.

For a min-problem, we can say we have a α-approximation algorithm ($\alpha \geq 1$) if the solution value for this algorithm is guaranteed to be at most α times the minimum ("optimal") value.

3.1 Approximating Solutions to Vertex Cover

Consider the Vertex-Cover problem. Recall that in this problem, we have a graph $G = (V, E)$ and we are asked to find a set $S \subseteq V$ such that for all $(u, v) \in E$, at least one of $u, v \in S$; we want $|S|$ to be minimized.

Now, we would like to construct a 2-approximate algorithm for the Vertex-Cover problem (i.e. find some S such that $|S| \leq 2 \cdot \text{Optimal Soln Size}$).

To do this, we will find the maximum matching M in G. This can be done in polynomial time. Now, return S as the set of vertices that are incident to M (recall that a matching is a set of edges).

First, we will claim that S is a vertex cover. Suppose for contradiction that S is not a vertex cover. Then there must exist vertices u, v such that $(u, v) \in E$ and $u, v \notin S$. Thus, M can be increased by one with (u, v), and M is not a maximum matching. Thus, S is a vertex cover.

Now, we will claim that $|S| 2 \cdot \text{Optimal Soln Size}$). The optimal solution has to cover every edge of M. Hence, the optimal solution size is at least the size of M. Finally, $2 \cdot \text{Optimal Soln Size} \geq 2|M| = |S|$, and we are done.

Thus, we have shown an algorithm that has a 2-approximate solution for Vertex-Cover.

As a final remark for Vertex-Cover, we do not yet know if that there exists a polynomial time algorithm that is α-approximate for $\alpha < 2$.