CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2014**

Lalla Mouatadid

Introduction to Complexity Theory

Recall a boolean formula ϕ is a formula composed of boolean variables (i.e. variables that take 0 or 1) and the standard boolean operators $\wedge, \vee, \neg$. ϕ is in **conjunctive normal form** (CNF) if it is the conjunction ($\wedge$) of clauses where each clause is a disjunction ($\vee$) of the input variables. For instance:

$$\phi(x, y, z) = (x \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (y \vee \neg z)$$

We will encode ϕ as list of clauses $C = \{C_1, C_2, ..., C_m\}$ where each C_i is a list of variables drawn from the set $\{x_1, .., x_n\}$ where each x_i is either positive or negative: $C_i \subseteq \{x_1, ...x_n, \neg x_1, ..., \neg x_n\}$. Therefore we can express ϕ in a compact way as follows:

$$\phi(x_1, ..., x_n) = \bigwedge_{i=1}^{m} \bigvee_{w \in C_i} w$$

ϕ is **satisfiable** if there exists an assignment a to the variables of ϕ such that $\phi(a) = 1$.
The **satisfiability** problem, known as SAT, is a decision problem where we return 1 iff there is an assignment that satisfies ϕ:

SAT: Given a boolen formular ϕ in CNF, return 1 iff ϕ is satisfiable

In the last lecture, we stated the following theorem that we didn't prove:

Theorem 1. *Circuit-SAT is NP-complete*

We will use Theorem 1 to show that SAT is also NP-Complete. It is (clear (?) why?) that SAT is a restricted version of circuit-SAT. We will thus show that despite such a strong structure imposed by SAT, the problem remains NP-complete. The following theorem is known as the Cook-Levin Theorem:

Theorem 2. *SAT is NP-complete.*

Recall that to show a problem $\mathcal{L}$ is NP-complete, we need to show 2 things:

1. $\mathcal{L} \in$ NP.
2. $\mathcal{L}$ is NP-hard (i.e. $A \leq_P \mathcal{L}$ for all $A \in NP$).

Theorem 1 states that Circuit-SAT is NP-complete, and thus NP-hard, which means for all $A \in$ NP:

$$A \leq_P \text{ Circuit-SAT}$$

Therefore there exists a polytime reduction from any problem A in NP to Circuit-SAT. So if we can show that Circuit-SAT $\leq_P$ SAT (i.e. a polytime reduction from Circuit-SAT to SAT) then by transitivity[1]:
$A \leq_P$ SAT for all $A \in NP$! We prove this formally in the following proposition:

Proposition 1. *Let A and B be decision problems. If A is NP-hard and $A \leq_P B$ then B is NP-hard.*

[1] properties of reducibility!

Proof. A is NP-hard $\implies \mathcal{L} \leq_P A$ for all $\mathcal{L} \in$NP. Meaning, there is a polytime reduction T such that:

$$\mathcal{L}(x) = 1 \iff A(T(x)) = 1 \text{ (for all } x \in \{0,1\}^*) \tag{1}$$

Moreover:

$$A \leq_P B \implies A(y) = 1 \iff B(R(y)) = 1 \text{ (for all } y \in \{0,1\}^*) \tag{2}$$

Where R is another polytime reduction. Now putting (1) and (2) together we get:

$$\begin{aligned} \forall x \in \{0,1,\}^* \quad \mathcal{L}(x) = 1 &\iff A(T(x)) = 1 \\ &\iff A(y) = 1 \text{ (for } y = T(x)) \\ &\iff B(R(y)) = 1 \\ &\iff B(R(T(x))) = 1 \\ &\iff B(S(x)) = 1 \text{ (where } S(x) = R(T(x))). \end{aligned}$$

S is a polytime reduction, why? We conclude that $\mathcal{L} \leq_P B$ for all $\mathcal{L} \in$NP and thus B is NP-hard. $\square$

Before we start the proof of the Cook-Levin Theorem, we first formally describe the input to the Circuit-SAT problem. We consider every circuit $\mathcal{C}$ with m gates and n input variables. We label each gate g_i for $1 \leq i \leq m$ and each input variable x_j for $1 \leq j \leq n$. Each gate g_i is encoded as follows:

$$g_i(\lambda_i, I_i, O_i)$$

where λ_i denotes the type of gate: $\lambda_i \in \{\wedge, \vee, \neg, x_1, ..., x_n\}$. If $\lambda_i \in \{\wedge, \vee, \neg\}$ then it is an interior gate in $\mathcal{C}$, otherwise it is an input gate if $\lambda_i \in \{x_1, ..., x_n\}$.

I_i denotes the input gates to g_i and thus $I_i \subseteq \{1...m\}$. Notice that if $\lambda_i \in \{\wedge, \vee\}$ then $|I_i| = 2$ (e.g. if $\lambda_i = \{\wedge\}, I_i = \{j,k\}$ then g_i computes $(g_j \wedge g_k)$), if $\lambda_i = \{\neg\}$ then $|I_i| = 1$ and if $\lambda_i \in \{x_1, ..., x_n\}$ then $|I_i| = 0$.

O_i is the set of output gates of g_i and is unbounded; but we make the assumption that if $|O_i| = 0$ then g_i is the output gate of the circuit.

So why do we need this detailed description of the circuit? Well to perform a reduction from Circuit-SAT to SAT, we are given a Circuit $\mathcal{C}$ and need to somehow encode a SAT input out of $\mathcal{C}$. Recall that the input to SAT is a boolean formula ϕ in CNF:

$$\phi(x_1, ..., x_n) = \bigwedge_{i=1}^{m} \bigvee_{w \in C_i} w$$

We are now ready to proove the Cook-Levin Theorem. The outline of the proof is as follows: We first give a polytime verifier (show the problem is in NP), then given a circuit $\mathcal{C}$ as described above, we will construct a SAT instance ϕ in polytime from $\mathcal{C}$ such that $\phi(a) = 1$ iff $\mathcal{C}(a) = 1$ for some assignment a of the input variables.

Proof. Our polytime verifier V will take the list of clauses C in addition to a certificate y; where y is an encoding of an assignment a to the variables of the clauses. Clearly we can verify if a is an accepting or rejecting instance in polynomial time by just plugging the assignment a into the clauses of C and checking if every C_i is satisfied. The verifier returns 1 iff all the clauses are satisified and 0 otherwise. Therefore SAT $\in$ NP.

To show SAT is NP-hard, we give a reduction from Circuit-SAT to SAT: let $\mathcal{C} = \{g_1, g_2, ..., g_m\}$ be the desciption of the circuit (input to Circuit-SAT) and $\{x_1, ..., x_n\}$ the input variables.

We first define $m+n$ variables ϕ a SAT instance as follows $\{x_1, x_2, .., x_n, y_1, y_2, ..., y_m\}$ where $\{x_1, ..., x_n\}$ are the input variables to $\mathcal{C}$ and $\{y_1, ..., y_m\}$ correspond to the values of $\{g_1, ..., g_m\}$ respectively given the assignment of $\{x_1, ..., x_n\}$ (in other words, y_i is the value of the ouput coming out g_i).

Next, we need to define the clauses $C = \{C_1, C_2, ..., C_m\}$ of our CNF. To do so, we consider each gate g_i for $1 \leq i \leq m$:

- If g_i is an input gate, then we introduce the constraint $y_i = x_j$ where x_j is the input to g_i. We represent $(y_i = x_j)$ in CNF as follows:
$$(y_i = x_j) \equiv (y_i \vee \neg x_j) \wedge (\neg y_i \vee x_j)$$
- If g_i is an $\neg$ gate and g_j is the input to g_i, then we introduce the constraint $y_i = \neg y_j$, equivalently:
$$(y_i = \neg y_j) \equiv (y_i \vee y_j) \wedge (\neg y_i \vee \neg y_j)$$
- If g_i is an $\wedge$ gate and g_j, g_k the two gates that connect to g_i, then $y_i = y_j \wedge y_k$:
$$(y_i = y_j \wedge y_k) \equiv (y_i \vee \neg y_j \vee \neg y_k) \wedge (\neg y_i \vee y_j) \wedge (\neg y_i \vee y_k)$$
- If g_i is an $\vee$ then $y_i = y_j \vee y_k$ and
$$(y_i = y_j \vee y_k) \equiv (\neg y_i \vee y_j \vee y_k) \wedge (y_i \vee \neg y_j) \wedge (y_i \vee \neg y_k)$$
- Finally if g_i is the output gate, we add the constraint $y_i \equiv 1$, which is equivalent to adding the clause (y_i).

Before showing that ϕ outputs 1 iff $\mathcal{C}$ outputs 1, we first need to show that this construction can be done in polynomial time given the circuit $\mathcal{C}$ as described above. To construct the clauses above, we iterate through all the gates in $\mathcal{C}$, check their type λ_i and record the corresponding clauses as constructed above. How many clauses do we record per gate? At most 3 clauses per gate (3 for $\wedge$ and $\vee$ gates). So the algorithm takes at most $\mathcal{O}(|\mathcal{C}|)$ time to construct $C = \{C_1, ..., C_m\}$ the list of clauses for ϕ, where ϕ is the conjunction of the clauses described above.

Next we show that $\mathcal{C}$ is satisfiable iff ϕ is satisfiable!

($\Rightarrow$) Let $(x_1^*, ..., x_n^*)$ be a satisfying assignment to $\mathcal{C}$. Then for every gate g_i in $\mathcal{C}$, we denote by y_i^* the value outputted by g_i given $(x_1^*, ..., x_n^*)$ and we claim that the assignment $(x_1^*, ..., x_n^*, y_1^*, ..., y_m^*)$ is a satisfying assignment to ϕ. Why? Because every clause in ϕ is satisfied **iff** each y_j has the value outputted by g_j given $(x_1^*, ..., x_n^*)$! But recall we added the final clause to encode the output gate which is satisfied iff the circuit output 1, which is the case given our assumption. So $(x_1^*, ..., x_n^*, y_1^*, ..., y_m^*)$ is a satisfying assignment to ϕ.

($\Leftarrow$) conversely, suppose $(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_{n+m}) = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n, \tilde{y}_1, ..., \tilde{y}_m)$ is a satisfying assignment to the constructed SAT instance. Then we claim that the values that $(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ receive will cause $\mathcal{C}$ to output 1. Why? Because the values of the $\tilde{y}_i$'s variables correspond precisely to the values of the gates in $\mathcal{C}$ given $(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$.

Conclusion: Given this Karp-rduction, it follows from the NP-hardness of Circuit-SAT that SAT is NP-hard, and thus SAT is NP-complete. □

So even if we restrict our Circuit to a CNF, and thus imposing lots of structure on our input, the problem still remains NP-complete. In the tutorial, you were introduced to 3-SAT and k-SAT in general: SAT instances where each clause in ϕ has most 3 (resp. k) variables. You were able to show that even if we restrict SAT even further to 3-SAT, the problem still remains NP-complete! This is quite surprising, especially when we consider the Interval Scheduling problem, a restricted instance of Independent Set, but we were able to solve it efficiently by having the structure imposed by the intervals.