NP versus Co-NP
COMS20010 (Algorithms II)

John Lapinskas, University of Bristol

The asymmetry of NP

Given a decision problem X, we write $\overline{X}$ for its **complement**:
`Yes` instances of X are `No` instances of $\overline{X}$ and vice versa.

For example, $\overline{\mathrm{SAT}}$ asks: Is the input CNF formula **un**satisfiable?

X and $\overline{X}$ are always equivalent under Cook reductions; we just apply our oracle and invert the answer. So they're equally hard to solve.

But despite this, $X \in \mathrm{NP}$ does not imply $\overline{X} \in \mathrm{NP}$, because membership of NP requires that `Yes` instances have witnesses.

We can verify that a CNF formula is satisfiable by checking a satisfying assignment, so SAT $\in$ NP. But how do we verify that it's **un**satisfiable? We'd need a short logical proof (i.e. one of polynomial length).

Conjecture: In general, this isn't possible. If it were, it wouldn't mean much for algorithms, but it would be a revolution in mathematics.

Given a decision problem X, we write $\overline{X}$ for its **complement**:

Yes instances of X are No instances of $\overline{X}$ and vice versa.

Having an algorithm for X gives you an algorithm for $\overline{X}$ and vice versa.

We define **Co-NP** to be the set of decision problems whose complements are in NP, such as $\overline{\text{SAT}}$.

Conjecture: NP $\neq$ Co-NP.
P $=$ NP would imply P $=$ Co-NP, so this conjecture implies P $\neq$ NP.

Cook reductions are useless here, since every problem in NP reduces to $\overline{\mathrm{SAT}}$ and every problem in Co-NP reduces to SAT.

We need a notion of reduction that can make finer distinctions and tell the two complexity classes apart...

Given a decision problem X, we write $\overline{X}$ for its **complement**:

Yes instances of X are No instances of $\overline{X}$ and vice versa.

Having an algorithm for X gives you an algorithm for $\overline{X}$ and vice versa.

Conjecture: NP $\neq$ Co-NP.
Cook reductions don't help with this, so we need something new...

If X and Y are decision problems, a **Karp reduction** from X to Y is a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

We write $X \leq_K Y$.

Intuitively: $X \leq_C Y$ means "X is no harder than Y".
$X \leq_K Y$ means "X is a special case of Y."

Karp reductions are **stronger** than Cook reductions; $X \leq_K Y \Rightarrow X \leq_c Y$, since we can apply our oracle to $f(x)$, but the reverse doesn't hold.

Given a decision problem X, we write $\overline{X}$ for its **complement**:
Yes instances of X are No instances of $\overline{X}$ and vice versa.

We define **Co-NP** $= \{X : \overline{X} \in \mathrm{NP}\}$.
Conjecture: NP $\neq$ Co-NP. This would imply P $\neq$ NP.

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a Yes instance of Y if and only if x is a Yes instance of X.

As with Cook reductions, we say a decision problem Y is **NP-hard under Karp reductions** if $X \leq_K Y$ for all $X \in \mathrm{NP}$.

Y is **NP-complete under Karp reductions** if it is also in NP.

The same definitions work for Co-NP...

Given a decision problem X, we write $\overline{X}$ for its **complement**:

Yes instances of X are No instances of $\overline{X}$ and vice versa.

We define **Co-NP** $= \{X : \overline{X} \in \text{NP}\}$.
Conjecture: NP $\neq$ Co-NP. This would imply P $\neq$ NP.

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a Yes instance of Y if and only if x is a Yes instance of X.

As with Cook reductions, we say a decision problem Y is **Co-NP-hard under Karp reductions** if $X \leq_K Y$ for all $X \in$ Co-NP.

Y is **Co-NP-complete under Karp reductions** if it is also in Co-NP.

The proof of Cook-Levin implies SAT is NP-complete under Karp reductions, and $\overline{\text{SAT}}$ is Co-NP-complete under Karp reductions.

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The good news: Every Cook reduction we've seen has also been Karp!

For SAT $\leq_C$ 3-SAT, we built a 3-SAT instance with the same answer as the SAT instance...

$$F = u \wedge (\neg u \vee \neg v) \wedge (v \vee \neg w \vee x \vee \neg y \vee \neg z) \wedge (y \vee z) \wedge (\neg v \vee w \vee z)$$

$$\big\downarrow$$

$$\begin{aligned} F' = {} & (u \vee f_1 \vee f_2) \wedge (\neg u \vee \neg v \vee f_1) \wedge (e_1 \vee \neg v \vee f_1) \wedge (e_1 \vee w \vee f_1) \wedge (\neg e_1 \vee v \vee \neg w) \\ & \wedge (e_2 \vee \neg e_1 \vee f_1) \wedge (e_2 \vee \neg x) \wedge (\neg e_2 \vee e_1 \vee x) \wedge (e_3 \vee \neg e_2 \vee f_1) \wedge (e_3 \vee y \vee f_1) \vee (\neg e_3 \vee e_2 \vee \neg y) \\ & \wedge (e_3 \vee \neg z \vee f_1) \wedge (y \vee z \vee f_1) \wedge (\neg v \vee w \vee z) \wedge (\neg f_1 \vee a_1 \vee a_2) \wedge (\neg f_1 \vee a_1 \vee \neg a_2) \wedge (\neg f_1 \vee \neg a_1 \vee a_2) \\ & \wedge (\neg f_1 \vee \neg a_1 \vee \neg a_2) \wedge (\neg f_2 \vee a_1 \vee a_2) \wedge (\neg f_2 \vee a_1 \vee \neg a_2) \wedge (\neg f_2 \vee \neg a_1 \vee a_2) \wedge (\neg f_2 \vee \neg a_1 \vee \neg a_2). \end{aligned}$$

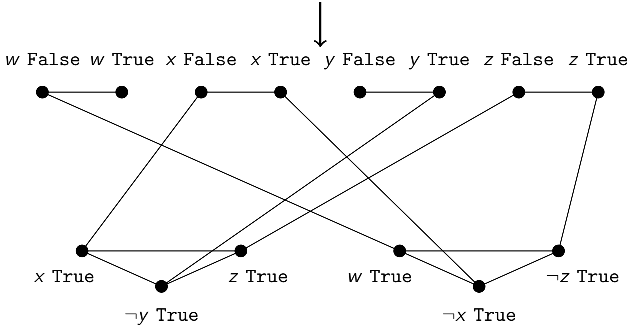
We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The good news: Every Cook reduction we've seen has also been Karp!

For 3-SAT $\leq_C$ IS, we built an independent set instance with the same answer as our 3-SAT instance...

$$F = (x \vee \neg y \vee z) \wedge (w \vee \neg x \vee \neg z)$$



We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The good news: Every Cook reduction we've seen has also been Karp!

For IS $\leq_C$ VC, we built a vertex cover instance with the same answer as our independent set instance...

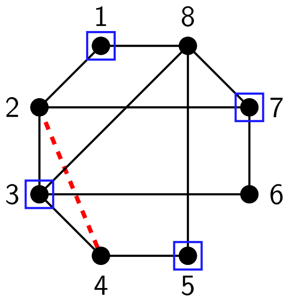
$$(G, k) \longrightarrow (G, |V| - k).$$

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The good news: Every Cook reduction we've seen has also been Karp!

And for VC $\leq_C$ ILP, we built an integer linear programming instance with the same answer as our vertex cover instance.



$$\longrightarrow$$

$$\begin{aligned} \textstyle\sum_v x_v &\to \min \text{ subject to} \\ x_u + x_v &\geq 1 \text{ for all } \{u,v\} \in E; \\ x_v &\leq 1 \text{ for all } v \in V; \\ x_v &\geq 0 \text{ for all } v \in V; \\ x_v &\in \mathbb{N} \text{ for all } v \in V. \end{aligned}$$

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The good news: Every Cook reduction we've seen has also been Karp!

The bad news: NP-completeness and NP-hardness are **different** concepts under Cook reductions than under Karp reductions.

Only decision problems can be NP-hard under Karp reductions, but all problems can be NP-hard under Cook reductions.

And we believe there **are** problems which are NP-complete under Cook reductions but not under Karp reductions.

The worse news: Different people use different definitions. Complexity theorists use Karp reductions, programmers use Cook reductions. And both groups usually just say "NP-hard" or "NP-complete".

In this course: If I don't give more detail, "NP-complete" means under Karp reductions, and "NP-hard" means under Cook reductions.

We write $X \leq_K Y$ if there is a **Karp reduction** from X to Y, i.e. a map f from instances of X to instances of Y such that:

- we can compute $f(x)$ in time polynomial in $|x|$;
- $f(x)$ is a `Yes` instance of Y if and only if x is a `Yes` instance of X.

The slightly better news: *Almost* every NP-complete problem is NP-complete under both Cook and Karp reductions. So thinking only in terms of Karp reductions still saves effort without really sacrificing power.

You can skip trying to come up with an algorithm and move straight to coming up with gadgets, trying to simulate aspects of one problem using the other.

And in areas where Karp-unfriendly reduction techniques are more common (e.g. counting problems), everyone just uses Cook reductions, even the pure theorists.