COMP3630 / COMP6363

week 10: **Various**

Most is not based on the book

slides created by: Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2025

Content of this Chapter

- On polytime-requirement for reductions
- Karp vs. Cook (reductions)
- Optimization problems

On polytime-requirement for reductions

On **P** membership vs. **P**-completeness

- For all classes (except **P**) we looked into completeness.
- So, why did we look into hardness (and thus, if matching bounds, completeness), rather than just membership?

On **P** membership vs. **P**-completeness

- For all classes (except **P**) we looked into completeness.
- So, why did we look into hardness (and thus, if matching bounds, completeness), rather than just membership?
  - Because membership is only an upper bound, not a lower one.
  - Hardness provides a lower bound.
  - E.g., providing an **EXPTIME** membership proof for a language doesn't prevent it from also being in **NP** or even **P**!

On **P** membership vs. **P**-completeness

- For all classes (except **P**) we looked into completeness.
- So, why did we look into hardness (and thus, if matching bounds, completeness), rather than just membership?
  - Because membership is only an upper bound, not a lower one.
  - Hardness provides a lower bound.
  - E.g., providing an **EXPTIME** membership proof for a language doesn't prevent it from also being in **NP** or even **P**!
- However, we never did that for **P**... We only showed membership! Why?

On **P** membership vs. **P**-completeness

- For all classes (except **P**) we looked into completeness.
- So, why did we look into hardness (and thus, if matching bounds, completeness), rather than just membership?
  - Because membership is only an upper bound, not a lower one.
  - Hardness provides a lower bound.
  - E.g., providing an **EXPTIME** membership proof for a language doesn't prevent it from also being in **NP** or even **P**!
- However, we never did that for **P**... We only showed membership! Why?
  - Because we need a refined definition of **P**-hardness, because . . .
  - otherwise, all problems in **P** are trivially **P**-hard and hence -complete!

P-completeness (Would–be)

Theorem w10.1

*If **P**-hardness were defined as for all other problems, then:*
*all non-trivial problems in **P** are **P**-complete.*

P-completeness (Would–be)

Theorem w10.1

If **P***-hardness were defined as for all other problems, then:*
all non-trivial problems in **P** *are* **P***-complete.*

Proof.

- Let $L \in \mathbf{P}$. Show that L is **P**-hard, i.e., for all $L' \in \mathbf{P}$, $L' \leq_P L$. Let $L' \in \mathbf{P}$.
- Show that there exists a reduction r, such that for any $w \in \Sigma^*$, $w \in L'$ iff $r(w) \in L$.

P-completeness (Would–be)

Theorem w10.1

*If **P**-hardness were defined as for all other problems, then:*
*all non-trivial problems in **P** are **P**-complete.*

Proof.

- Let $L \in \mathbf{P}$. Show that L is **P**-hard, i.e., for all $L' \in \mathbf{P}$, $L' \leq_P L$. Let $L' \in \mathbf{P}$.
- Show that there exists a reduction r, such that for any $w \in \Sigma^*$, $w \in L'$ iff $r(w) \in L$.
- Define r as follows:
  - Let $w_{yes} \in L$ and $w_{no} \notin L$. These are constant and thus independent of any w.
  - Decide $w \in L'$ in polynomial time (possible by assumption).
  - If $w \in L'$, define $r(w) = w_{yes}$, otherwise $r(w) = w_{no}$. □

Why does this break for trivial problems?

P-completeness (Would–be)

Theorem w10.1

If **P***-hardness were defined as for all other problems, then:*
all non-trivial problems in **P** *are* **P***-complete.*

Proof.

- Let $L \in \mathbf{P}$. Show that L is **P**-hard, i.e., for all $L' \in \mathbf{P}$, $L' \leq_P L$. Let $L' \in \mathbf{P}$.
- Show that there exists a reduction r, such that for any $w \in \Sigma^*$, $w \in L'$ iff $r(w) \in L$.
- Define r as follows:
  - Let $w_{yes} \in L$ and $w_{no} \notin L$. These are constant and thus independent of any w.
  - Decide $w \in L'$ in polynomial time (possible by assumption).
  - If $w \in L'$, define $r(w) = w_{yes}$, otherwise $r(w) = w_{no}$. □

Why does this break for trivial problems?

- A language L is called trivial iff $L = \emptyset$ or $L = \Sigma^*$.
- In both cases, there does not exist both an w_{yes} instance and w_{no} instance.
- So, no reduction is possible for trivial problems.

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

- The question doesn't even make sense... polynomial in what? The language? What does that mean? How is it presented? What "does change"? (Any complexity analysis requires "an input that changes".)

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

- The question doesn't even make sense... polynomial in what? The language? What does that mean? How is it presented? What "does change"? (Any complexity analysis requires "an input that changes".) Let's look into the question anyway! :)
- The proof also assumes that we "have" the polytime procedure that decides L'. But do we really know it, just because we "know" $L' \in \mathbf{P}$?

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

- The question doesn't even make sense... polynomial in what? The language? What does that mean? How is it presented? What "does change"? (Any complexity analysis requires "an input that changes".) Let's look into the question anyway! :)
- The proof also assumes that we "have" the polytime procedure that decides L'. But do we really know it, just because we "know" $L' \in$ **P**? Probably? For a problem to be in P, we either know that there exists an algorithm that proves **P**-membership or there is a series of reductions to a problem in **P** (which is again a poly algorithm, since polytime functions compose). However, I'm not sure whether for every problem in **P**, we are give a concrete decision procedure that runs in polytime, or whether for some problem we just "concluded" that there must be some...

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

- The question doesn't even make sense... polynomial in what? The language? What does that mean? How is it presented? What "does change"? (Any complexity analysis requires "an input that changes".) Let's look into the question anyway! :)
- The proof also assumes that we "have" the polytime procedure that decides L'. But do we really know it, just because we "know" $L' \in \mathbf{P}$? Probably? For a problem to be in P, we either know that there exists an algorithm that proves **P**-membership or there is a series of reductions to a problem in **P** (which is again a poly algorithm, since polytime functions compose). However, I'm not sure whether for every problem in **P**, we are give a concrete decision procedure that runs in polytime, or whether for some problem we just "concluded" that there must be some...
- The proof assumes that we "know" a constant $w_{yes} \in L'$ and $w_{no} \notin L'$. But do we?

On finding reductions

Maybe a philosophical question: Can we always find a reduction in polynomial time?

- The question doesn't even make sense... polynomial in what? The language? What does that mean? How is it presented? What "does change"? (Any complexity analysis requires "an input that changes".) Let's look into the question anyway! :)
- The proof also assumes that we "have" the polytime procedure that decides L'. But do we really know it, just because we "know" $L' \in \mathbf{P}$? Probably? For a problem to be in P, we either know that there exists an algorithm that proves **P**-membership or there is a series of reductions to a problem in **P** (which is again a poly algorithm, since polytime functions compose). However, I'm not sure whether for every problem in **P**, we are give a concrete decision procedure that runs in polytime, or whether for some problem we just "concluded" that there must be some...
- The proof assumes that we "know" a constant $w_{yes} \in L'$ and $w_{no} \notin L'$. But do we?
  - One of them: yes, in polytime. Just take any word $w \in \Sigma^*$ and decide it in polytime. Then, either declare $w_{yes} := w$ (if $w \in L'$) or $w_{no} := w$ (if $w \notin L'$).
  - But how to find "the other answer"? Even if such a witness has polylength based on ... (what? the representation of the language?), there would still be exponentially many words to try.

This question relates to those "more philosophical ones" in the tutorial of week 9.

Why polytime reductions? Why not more?

Theorem w10.2

Under poly-space reductions, all non-trivial problems in **PSPACE** *are* **PSPACE**-*complete.*

Comment.

Recall that this includes all problems in **P** and **NP**.

Why polytime reductions? Why not more?

Theorem w10.2

Under poly-space reductions, all non-trivial problems in **PSPACE** *are* **PSPACE**-*complete.*

Comment.
Recall that this includes all problems in **P** and **NP**.

Proof.

Identical to the one before. Just replace **P** by **PSPACE** and poly-time by poly-space. □

Karp vs. Cook (reductions)

Trivia

Stephen Cook

- Formalized the notion of polytime-reductions in 1971, in his paper: "The Complexity of Theorem Proving Procedures"
- Such reductions are also called "Cook-reductions" (see next slide)
- Remember "Cook's Theorem"? He proved SAT **NP**-complete.
- He won the Turing Award ("Nobel Prize for Computer Science") in 1982

Richard Karp

- Proved 21 important problems **NP**-complete in his 1972 paper: "Reducibility Among Combinatorial Problems"
- Provided an alternative definition of reductions, "Karp-reduction" (see next slide)
- He won the Turing Award in 1985

Karp- and Cook-reductions

Definition w10.1 (Cook-reduction)

Let A and B be decision problems. We say that A Cook-reduces to B (written $A \leq_P^C B$) if there exists a deterministic Turing machine M that decides A in polynomial time with access to an oracle for B (arbitrarily often).

Karp- and Cook-reductions

Definition w10.1 (Cook-reduction)

Let A and B be decision problems. We say that A Cook-reduces to B (written $A \leq_P^C B$) if there exists a deterministic Turing machine M that decides A in polynomial time with access to an oracle for B (arbitrarily often).

Definition w10.2 (Karp-reduction)

Let A and B be decision problems. We say that A Karp-reduces to B (written $A \leq_P^K B$) if there exists a function $f : \Sigma^* \to \Sigma^*$ such that:

- f is computable in polynomial time, and
- for all $w \in \Sigma^*$: $w \in A$ if and only if $f(w) \in B$.

Note:

- We use . . .

Karp- and Cook-reductions

Definition w10.1 (Cook-reduction)

Let A and B be decision problems. We say that A Cook-reduces to B (written $A \leq_P^C B$) if there exists a deterministic Turing machine M that decides A in polynomial time with access to an oracle for B (arbitrarily often).

Definition w10.2 (Karp-reduction)

Let A and B be decision problems. We say that A Karp-reduces to B (written $A \leq_P^K B$) if there exists a function $f : \Sigma^* \to \Sigma^*$ such that:

- f is computable in polynomial time, and
- for all $w \in \Sigma^*$: $w \in A$ if and only if $f(w) \in B$.

Note:

- We use Karp-reductions!
- Every Karp-reduction is also a Cook-reduction (trivially: the Cook oracle just calls the result of the Karp reduction).
- Not every Cook-reduction is a Karp-reduction (non-trivial, skipped here).

Karp- and Cook-reductions

Definition w10.1 (Cook-reduction)

Let A and B be decision problems. We say that A Cook-reduces to B (written $A \leq_P^C B$) if there exists a deterministic Turing machine M that decides A in polynomial time with access to an oracle for B (arbitrarily often).

Definition w10.2 (Karp-reduction)

Let A and B be decision problems. We say that A Karp-reduces to B (written $A \leq_P^K B$) if there exists a function $f : \Sigma^* \to \Sigma^*$ such that:

- f is computable in polynomial time, and
- for all $w \in \Sigma^*$: $w \in A$ if and only if $f(w) \in B$.

Note:

- We use Karp-reductions!
- Every Karp-reduction is also a Cook-reduction (trivially: the Cook oracle just calls the result of the Karp reduction).
- Not every Cook-reduction is a Karp-reduction (non-trivial, skipped here).
- We get different theoretical results for those reductions (see next slide).

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Further notes on Karp vs. Cook:

- Karp-reductions are transitive, but Cook-reductions are not (because we would need Oracles from multiple problems).

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Further notes on Karp vs. Cook:

- Karp-reductions are transitive, but Cook-reductions are not (because we would need Oracles from multiple problems).
- We don't know yet whether the set of **NP**-complete problems under Cook- and Karp-reductions are the same.

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Further notes on Karp vs. Cook:

- Karp-reductions are transitive, but Cook-reductions are not (because we would need Oracles from multiple problems).
- We don't know yet whether the set of **NP**-complete problems under Cook- and Karp-reductions are the same.
- Cook lets us flip the answer after a polytime reduction, Karp doesn't.

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Further notes on Karp vs. Cook:

- Karp-reductions are transitive, but Cook-reductions are not (because we would need Oracles from multiple problems).
- We don't know yet whether the set of **NP**-complete problems under Cook- and Karp-reductions are the same.
- Cook lets us flip the answer after a polytime reduction, Karp doesn't.
- We don't know whether the set of **NP**-complete problems under Cook reductions are the same as those under Karp reductions.

Karp vs. Cook reductions

Why Karp? If we have a deterministic algorithm for an **NP**-complete problem that runs in time worse than poly, but not yet exponential, e.g., $\mathcal{O}(n^{\log n})$, then

- with Karp, we can solve any problem in **NP** in that time (because Karp is transitive and polytime composes)
- with Cook, we cannot conclude anything (because it's not transitive, see below)

Further notes on Karp vs. Cook:

- Karp-reductions are transitive, but Cook-reductions are not (because we would need Oracles from multiple problems).
- We don't know yet whether the set of **NP**-complete problems under Cook- and Karp-reductions are the same.
- Cook lets us flip the answer after a polytime reduction, Karp doesn't.
- We don't know whether the set of **NP**-complete problems under Cook reductions are the same as those under Karp reductions.
- If **P** = **NP** (note that the definition of **NP** does not depend on reductions), the two kinds of reductions are equally expressive.

Optimization Problems

Optimization Problems

So far:

- We have just considered yes/no problems
- E.g., "Does problem P possess 'a solution'?"

In Practice:

- We want to *obtain* a solution! And maybe even the best!
- For example, a satisfying assignment, or the size of the smallest vertex cover.

Optimization Problems

So far:

- We have just considered yes/no problems
- E.g., "Does problem P possess 'a solution'?"

In Practice:

- We want to *obtain* a solution! And maybe even the best!
- For example, a satisfying assignment, or the size of the smallest vertex cover.

Example w10.1

- Yes/No problem: Does G have a vertex cover of size $\leq k$?
- Optimization problem:
  - What is the size of the smallest vertex cover for G? Or:
  - What is a vertex cover for G with the smallest size?

Optimization Problems

So far:

- We have just considered yes/no problems
- E.g., "Does problem P possess 'a solution'?"

In Practice:

- We want to *obtain* a solution! And maybe even the best!
- For example, a satisfying assignment, or the size of the smallest vertex cover.

Example w10.1

- Yes/No problem: Does G have a vertex cover of size $\leq k$?
- Optimization problem:
  - What is the size of the smallest vertex cover for G? Or:
  - What is a vertex cover for G with the smallest size?

Observation:

- If we can solve the optimization problem, we can solve the yes/no problem.

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.
- Since this comparison can be done in **P**, we also solved our decision problem in **P**.

Completeness for Optimisation Problems

Optimisation Problems

- Cannot be in **NP**, as they are not yes/no problems.
- In fact, cannot be in any complexity class, since it's not a decision problem.

Theorem w10.2

If the a decision problem is **NP***-complete and* $\mathbf{P} \neq \mathbf{NP}$*, then we cannot solve the optimization version of it in polytime.*

Proof.

- We know: yes/no version is **NP**-complete and $\mathbf{P} \neq \mathbf{NP}$ (as assumed).
- Now assume we can solve the optimization version in **P**.
- Solve this problem in **P**. Compare solution size s with k of the decision variant. Return yes iff $s \leq k$.
- Since this comparison can be done in **P**, we also solved our decision problem in **P**.
- This is a contradiction to $\mathbf{P} \neq \mathbf{NP}$, so the optimization problem is not in **P**.

□

Examples

Example w10.3

- "Does ϕ have a satisfying valuation?" can be expressed as a decision problem.
- "Find a satisfying valuation of ϕ." is <u>not</u> a decision problem.

Examples

Example w10.3

- "Does ϕ have a satisfying valuation?" can be expressed as a decision problem.
- "Find a satisfying valuation of ϕ." is not a decision problem.

From week 11:

Example w10.4

- "Does the classical planning problem P have a solution?" can be expressed as a decision problem.
- "Find a solution to the classical planning problem P." is not a decision problem.

Examples

Example w10.3

- "Does ϕ have a satisfying valuation?" can be expressed as a decision problem.
- "Find a satisfying valuation of ϕ." is not a decision problem.

From week 11:

Example w10.4

- "Does the classical planning problem P have a solution?" can be expressed as a decision problem.
- "Find a solution to the classical planning problem P." is not a decision problem.

Related to week 12: (too complex, not covered)

Example w10.5

- "Does the delete-relaxed HTN planning problem P have a solution?" can be expressed as a decision problem. (It is **NP**-complete, proved in 2014 by Alford et al.)
- "Find a solution to the delete-relaxed HTN planning problem P." is not a decision problem. Interestingly, even shortest solutions can be exponentially long in $|P|$! (Think of what that implies for certificates! How is that possible?!)