CS 341 - Algorithms, Spring 2017. University of Waterloo

Lecture 17: Polynomial time reductions

We will begin our study of polynomial time reductions, which allow us to compare the difficulty of different problems in terms of polynomial time solvability.

Polynomial time reductions

Once we have learned more and more algorithms, they become our building blocks and we may not need to design algorithms for a new problem from scratch.

It becomes more and more important to be able to use existing algorithms to solve new problems.

We have seen some reductions already.

For example, if we have an algorithm for longest common subsequence, then we can use it to solve the longest increasing subsequence problem. We reduce LIS to LCS.

If we have an algorithm for maximum matching in bipartite graphs, then we can use it to solve the baseball / basketball league problem. We reduce baseball league to matching.

In general, if there is an efficient reduction from problem A to problem B and there is efficient algorithm to solve problem B, then we have an efficient algorithm to solve problem A.

Decision problems

To formalize the notion of a reduction, it is more convenient to restrict our attention to decision problems, for which we just need to answer YES or NO.

For example, instead of finding a maximum matching, the decision version of the problem is "Does G have a matching of size k?".

As we will discuss later, we won't lose much by considering decision problems.

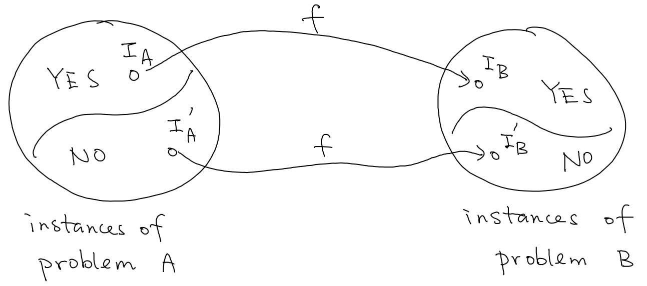
If we know how to solve decision problems, then we can use the algorithm to find a solution (e.g. a matching of size k).

Definition (polynomial time reductions)

We say a decision problem A is polynomial time reducible to a decision problem B if for every instance I_A of A there is a polynomial time algorithm f that maps/transforms

I_A into an instance I_B of B (that is, $f(I_A)=I_B$) such that I_A is a YES instance for problem A if and only if I_B is a YES instance for problem B.

We use the notation $A \le_p B$ to denote that such a reduction exists, meaning that A is not more difficult than B in terms of polynomial time solvability.



If such a polynomial time reduction algorithm f exists, then given an instance I_A of problem A we can apply f to get an instance $I_B=f(I_A)$ of problem B.

The reduction algorithm has the property that I_A is a YES-instance iff I_B is a YES-instance.

So, if we have an efficient algorithm for problem B so that we can determine if I_B is a YES-instance efficiently, then this determines whether I_A is a YES-instance efficiently.

More precisely, if f has time complexity $p(n)$ for an instance I_A of size n, and if there is an algorithm ALG_B to solve an instance I_B of size m in time complexity $q(m)$, then we can solve problem A in time $O(q(p(n)))$.

So, if both p and q are polynomials, then we can solve problem A in polynomial time by the following algorithm.

<u>Algorithm</u> (solving problem A by reduction)

Input: an instance I_A of problem A

Output: whether I_A is a YES-instance

- Use the reduction algorithm f to map/transform I_A into $I_B=f(I_A)$ of problem B.
- Return $ALG_B(I_B)$

We will soon give many examples of reduction algorithms.

<u>Proving hardness using reductions</u>

So far it is all familiar: We reduce problem A to problem B efficiently, and use an efficient algorithm for problem B to solve problem A.

Now, we explore the other implication of the inequality $A \le_p B$.

Suppose problem A is known to be impossible to solve in polynomial time.

Then $A \le_p B$ implies that B cannot be solved in polynomial time either, as otherwise we could solve problem A in polynomial time by the previous discussion.

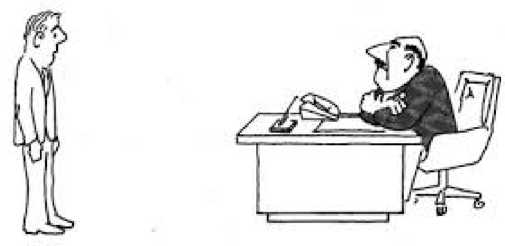
Therefore, if A is computationally hard and $A \le_p B$, then B is also computationally hard.

In reality, we know very little about proving that a problem cannot be solved in polynomial time, and so we couldn't draw such a strong conclusion from $A \le_p B$.

But suppose there is a problem, say the traveling salesman problem in L14, which is very famous and has attracted many brilliant researchers to solve it in polynomial time but to no avail.

Your boss gives you a problem C, and you couldn't solve it in polynomial time.

Instead of just saying that you couldn't solve it, it would be much more convincing if you could show that $TSP \le_p C$. Here is the NP-completeness cartoon from Garey and Johnson.


"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, but neither can all these famous people."

This is what we will be doing for the next few lectures!

Why polynomial time?

Because it tells us that the problems are fundamentally easier that it doesn't require exhaustive search.

Simple reductions

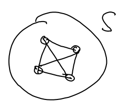
We will show that the following problems are equivalent in terms of polynomial-time solvability, either they all can be solved in polytime or they all cannot be solved in polytime.

Maximum Clique (Clique) A subset $S \subseteq V$ is a clique if $uv \in E \ \forall u,v \in S$.

Maximum Clique (Clique) A subset $S \subseteq V$ is a clique if $uv \in E \quad \forall u,v \in S$.

Input: Graph $G=(V,E)$, an integer k

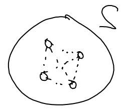
Output: Is there a clique in G having $\geq k$ vertices?



Maximum Independent set (IS) A subset $S \subseteq V$ is an independent set if $uv \notin E \quad \forall u,v \in S$.

Input: Graph $G=(V,E)$, an integer k

Output: Is there an independent set in G having $\geq k$ vertices?



Minimum vertex cover (VC) A subset $S \subseteq V$ is a vertex cover if $uv \cap S \neq \phi \quad \forall uv \in E$.

Input: Graph $G=(V,E)$, an integer k.

Output: Is there a vertex cover in G having $\leq k$ vertices?

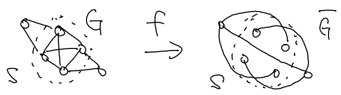
Claim Clique $\leq_p$ IS and IS $\leq_p$ clique.

proof This is easy to see.

Clique and IS are very similar, one wants to find $\geq k$ vertices with all edges in between, and one wants to find $\geq k$ vertices with no edges in between.

So, to reduce Clique to IS, we just need to change edges to non-edges and vice versa.

More formally, to solve Clique on $G=(V,E)$, the reduction algorithm f would construct a graph $\bar{G}=(V,\bar{E})$ where $uv \in E$ if and only if $uv \notin \bar{E}$.



Clearly, the reduction algorithm runs in polynomial time, actually in $O(n+m)$ time.

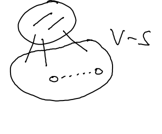
Now, it is also easy to see that S is a clique in G iff S is an independent set in $\bar{G}$.

So, $\{G,k\}$ is an YES-instance for clique iff $\{\bar{G},k\}$ is an YES-instance for IS. □

To see the connection between independent sets and vertex covers, we need the following observation.

Observation In $G=(V,E)$, $S \subseteq V$ is a vertex cover of G iff $V-S$ is an independent set in G.

proof If there is an edge e between two vertices in $V-S$, then S is not a vertex cover, as $S \cap e = \phi$.



So, if S is a vertex cover, then $V-S$ is an independent set.

Similarly, if $V-S$ is an independent set, then all the edges in G have at least one vertex in S, and thus S is a vertex cover. □

With this observation, it follows that whether G has a vertex cover of size $\le k$ if and only if G has an independent set of size $\ge n-k$.

So, the reduction algorithm simply maps $\{G,k\}$ for VC to $\{G, n-k\}$ for IS, and clearly it runs in polynomial time (in fact constant time!) and satisfies the iff property.

Therefore, $VC \le_p IS$ and $IS \le_p VC$.

<u>Claim</u> If $A \le_p B$ and $B \le_p C$, then $A \le_p C$.

<u>proof</u> If there is a polytime algorithm f that maps A to B, and a polytime algorithm g that maps B to C, then $g \circ f$ maps A to C and still runs in polynomial time. And also $g \circ f$ has the property that it maps a YES-instance of A to a YES-instance of C, and maps a No-instance of A to a No-instance of C. □

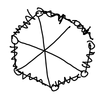
So, Clique, IS, and VC are equivalent in polynomial time solvability.

<u>More simple reductions</u>

<u>Hamiltonian cycle (HC)</u> A cycle is a Hamiltonian cycle if it touches every vertex exactly once.

Input: $G=(V,E)$

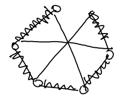
Output: Does G have a Hamiltonian cycle?



<u>Hamiltonian path (HP)</u> A path is a Hamiltonian path if it touches every vertex exactly once.

Input: $G=(V,E)$

Output: Does G have a Hamiltonian path?

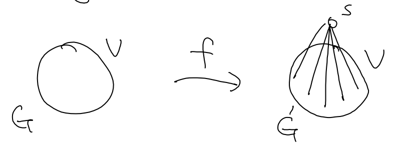


It is not surprising that these two problems are equivalent in terms of polytime computation, but it is a good exercise to work out the reductions.

First, we show $HP \le_p HC$.

Given $G=(V,E)$ for HP, we construct $G'=(V+s, E')$ by copying G and add a new vertex s that connects to every vertex in V.

Clearly, the reduction runs in polynomial time.



We claim that G has a Hamiltonian path iff G' has a Hamiltonian cycle.

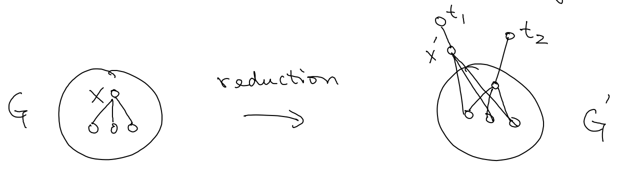
($\Rightarrow$) If G has a Hamiltonian path P with endpoints a and b, then $P+sa+sb$ is a Hamiltonian cycle in G'.

($\Leftarrow$) If G' has a Hamiltonian cycle C, then there are two edges incident on s, call them sa and sb, then $C-sa-sb$ is a Hamiltonian path in G.

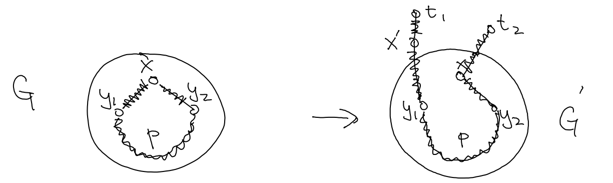
Therefore, we conclude that $HP \leq_p HC$.

What about the other direction? We would like to show $HC \leq_p HP$.

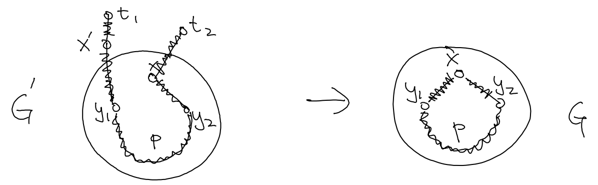
Given $G=(V,E)$ for HC, we pick an arbitrary vertex $x \in V$, and construct G' by adding a duplicate x' of x and two new degree one vertices t_1 and t_2 as shown in the picture.



($\Rightarrow$) If there is a Hamiltonian cycle in G, then there is a Hamiltonian path in G' by replacing xy_1 by $x'y_1$ and t_1x', and also add xt_2.



($\Leftarrow$) If there is a Hamiltonian path in G', then the two endpoints must be t_1 and t_2, since they are degree one vertices.



Then t_1x' must be in the path, and call the other neighbor of x' be y_1.

Since x' is a duplicate of x, we know that x has an edge to y_1.

We also know that t_2x must be in the Hamiltonian path, since t_2 is a degree one vertex.

So, by replacing t_1x', $x'y_1$ and t_2x by xy_1, we have a Hamiltonian cycle in G.

Therefore, $HC \leq_p HP$.

A common technique to do reduction is to show that one problem is a special case of another problem. We call this <u>specialization</u>.

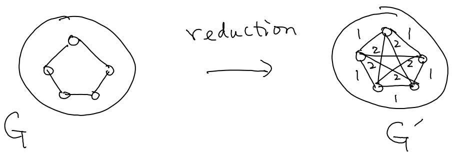
<u>Claim</u> $HC \leq_p TSP$.

<u>Proof</u> Given $G=(V,E)$ for HC, we construct $G'=(V,E')$ for TSP such that if $uv \in E$

Claim HC $\le_p$ TSP.

proof Given $G=(V,E)$ for HC, we construct $G'=(V,E')$ for TSP such that if $uv \in E$, then we set its weight in G' to be 1, and if $uv \notin E$, we set the weight of uv in G' to be 2 (so that G' is a complete weighted graph).

Then G has a Hamiltonian cycle if and only if G' has a TSP tour with total cost n. □



A nontrivial reduction

First we introduce an important problem.

We are given n boolean variables $x_1, \dots, x_n$. each can either be set to True or False.

We are also given a formula in conjunctive normal form (CNF), where it is an AND of the clauses, and each clause is an OR of some literals, where a literal is either x_i or $\bar{x}_i$, which is the negation of x_i.

For example, in $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3)$, there are four clauses, each has at most three literals, and the formula has only three variables.

3-Satisfiability (3SAT)

Input: A CNF-formula in which each clause has at most three literals.

Output: A truth assignment to the variables that satisfies all the clauses.

In the above example. setting $x_1=T$ $x_2=F$ $x_3=T$ will satisfy all the clauses, and hence the formula.

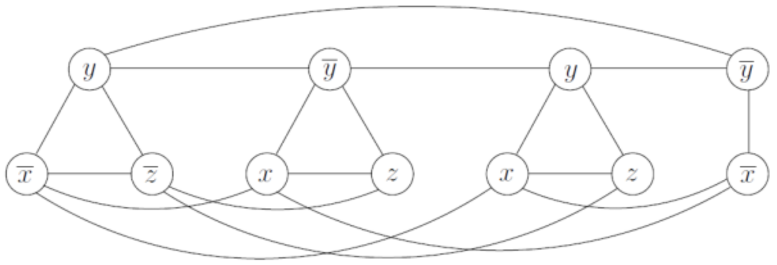
This problem looks quite different than other problems that we have seen.

Doing a reduction between two seemingly different problems often requires some new ideas.

Theorem 3SAT $\le_p$ IS.

Given a 3SAT formula, we would like to construct a graph such that the formula is satisfiable if and only if the graph has an independent set of certain size.

Figure 8.8 The graph corresponding to $(\overline{x} \vee y \vee \overline{z})\ (x \vee \overline{y} \vee z)\ (x \vee y \vee z)\ (\overline{x} \vee \overline{y})$.



[DPV]

The reduction is to create one vertex for each literal.

To satisfy the formula, we need to set at least one literal for each clause to be True.

It is easy to do that if we satisfy each clause separately, but the important point is to be consistent over the choices, i.e. if we set x to be True to satisfy some clause, then we can't set x to be False to satisfy other clauses.

We will use the graph edges to enforce consistency, so that x is connected to every $\overline{x}$ and vice versa, so that they won't both belong to an independent set.

We also add edges between literals of the same clause, to make sure that we only choose one literal in each clause to satisfy.

So, there are two types of edges, one within clauses, and another to enforce consistency.

Clearly, the reduction can be done in polynomial time; a formula with N literals is mapped to a graph with N vertices.

The following claim will imply the theorem.

Claim Suppose the formula has k clauses. Then the formula is satisfiable if and only if there is an independent set of size k in the graph.

proof ($\Rightarrow$) If there is a satisfying assignment, then we choose one literal that is set to True in each clause (e.g. if $x_2 = F$ in the satisfying assignment, then the literal $\overline{x_2}$ is True), and put the corresponding vertex to be in the independent set.

Since the assignment is satisfiable, there is at least one true literal in each clause, and so the set has at least k vertices.

These k vertices form an independent set because we won't choose x and $\overline{x}$ for some variable x, as the satisfying assignment is consistent.

($\Leftarrow$) Suppose there is an independent set of size k.

Since there must be edges between literals in each clause, any independent set can only pick at most one vertex in each clause.

Since there are only k clauses, an independent set of size k must choose exactly one vertex from each clause.

Also, because of the consistency edges, each variable we choose at most one literal (we could choose none of the literals).

So, if we choose x_i, then we set x_i to be true, and if we choose $\bar{x}_i$ in the independent set, then we set x_i to be false

By the consistency, this assignment is well defined, and since there is a vertex in each clause, this assignment satisfies every clause. $\square$

Concluding remarks

We have introduced the notion of a polynomial time reduction, and we used it to prove the relations between different problems, and so far we have

$$\begin{tikzcd}
& IS \arrow[r] \arrow[dl, shift left] \arrow[dr, shift left] & 3SAT & HC \arrow[r, bend left] & HP \arrow[l, bend left] \\
Clique \arrow[ur, shift left] & & VC \arrow[ul, shift left] & & TSP \arrow[ul]
\end{tikzcd}$$

an edge $B \leftarrow A$ means $B \leq_p A$.

In principle, we can add new problems and relate to these problems, and slowly develop a big web of all computational problems.

Because $\leq_p$ is transitive, any strongly connected component in this web forms an equivalent class of problems, in terms of polytime solvability.

Is there a better way to do it than to consider the problem one by one?

References: [KT 8.1, 8.2]