Algorithmic Modeling and Complexity **Fall 2003**

Lecture 12: 9 October

Lecturer: J. van Leeuwen *Scribes: Shay Uzery & Kasper van den Berg*

12.1 Overview

We have seen many practical problems for which no polynomial-time algorithm is currently known to exist. Computational complexity theory has shown that there are many problems which indeed have no polynomial-time or even exponential-time solution. There exist problems of arbitrary complexity. The problems we have seen usually admit a good network- or integer LP model and the question arises why it is so hard to find a polynomial-time algorithm for them. In 1971, Cook showed that the essence of the question is a major open problem in complexity theory: *the P-versus-NP question.*

In this lecture a brief description of the theory of NP-completeness is given, and we will examine numerous NP-complete problems. We will also discuss the curious *threshold phenomenon* that some parameterized problems may be polynomial for small values of the parameters and NP-complete for values that are only slightly larger. We illustrate this for the Set Cover problem: the problem with 2 elements per sets has a polynomial-time computable solution, but the problem with 3 elements per sets is NP-complete.

The definition of polynomial-time computability and similar notions requires an underlying model of computation with a fair instruction set. These details are important but not treated in this lecture.

12.2 P versus NP

The algorithmic models we consider in this course are usually optimization problems. The theory in this lecture deals exclusively with the *decision version* of these problems. For example, the decision version of the Vertex Cover problem is: given a network G and integer k, does G has a vertex cover of size $\leq k$?

Definition 12.1 *The 'language' of a decision problem is the set of all instances I for which the answer to the problem is 'Yes'.*

Thus, $\langle G, k\rangle \in$ 'Vertex Cover' if G has a vertex cover of size $\leq k$.

Definition 12.2 *P is the class of all decision problems ('languages') solvable by a program with polynomial running time.*

Although Vertex Cover is not known to be in P, the following algorithm would 'solve' it apparently efficiently for a network $G = \langle V, E \rangle$:

> **Algorithm** VC-N
>
> **choose** any set of k vertices from V.
>
> **check** whether it is a vertex cover.
>
> If it is, return 'Yes', otherwise return 'No'.

This is an example of a non-deterministic algorithm. It makes use of a 'choose' operator which writes a sensible dataset, on which the algorithm continues. We are especially interested in whether a 'Yes' answer can be reached this way. A non-deterministic algorithm has the following properties:

Soundness: if the algorithm answers 'Yes' than there is a positive solution for the problem.

Completeness: if the problem has a positive solution than there is an option for the choice(s) in the algorithm that leads the algorithm to return 'Yes'.

Note: if a non-deterministic algorithm answers 'No', then there might still be an other option (or set of options) for the choice(s) in which the algorithm would return 'Yes'. Thus, a 'No'-answer has no definitive value.

In algorithm VC-N the 'choose' operator works in polynomial time, i.e. writes down a sensible dataset in polynomial time. Algorithm VC-N is said to run in 'nondeterministic polynomial time'. To define this more precisely, let's modify all nondeterministic algorithms such that they encode all choices that would have to be made during a run in a string y that is chosen at the start. The various choices to be made during the run can simply be made deterministically during the run by extracting the desired values from y.

Definition 12.3 *A decision problem A is NP-solvable ('in NP') if there are a program π and polynomial functions p and q such that : $x \in A \Leftrightarrow \exists_{y,|y| \leq q(x)} \pi(x, y) =$ 'Yes' and π runs in time complexity bounded by $p(|x|, |y|)$.*

Definition 12.4 *NP is the class of all problems that are NP-solvable.*

A major open problem since 1971 is: **prove or disprove that** $P = NP$. A considerable prize awaits the student or scientist who solves it ([1]).

12.3 Structure of NP

In figure 12-1 we see an illustration of the universe of problem classes according to the general belief. In this section the concepts shown in the figure will be defined more formally. We will not deal with the 'world' of problems outside NP.

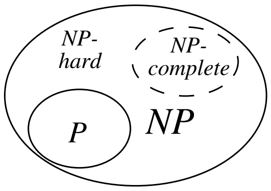

Figure 12-1: P and NP in the world of all problems

Definition 12.5 *$B \preccurlyeq_p A$ ('B P-reduces to A') if there exists a function f from instances of B to instances of A, such that:* (i) *f is computable in a polynomial time, and* (ii) *$x \in B \Leftrightarrow f(x) \in A$.*

$B \preccurlyeq_p A$ means that A is at least as hard as B. The notion of reducibility as defined is also called *Karp-reducibility*. It is slightly stronger than the notion of *Turing reducibility*: $B \preccurlyeq_T A$ if B can be solved by means of a (polynomial-time) algorithm that can call A as a subroutine, with each call counting as one step.

Lemma 12.6 (i) *$C \preccurlyeq_p B$ and $B \preccurlyeq_p A \Rightarrow C \preccurlyeq_p A$ (transitivity).*
(ii) *$B \preccurlyeq_p A$ and $A \in P \Rightarrow B \in P$.*
(iii) *$B \preccurlyeq_p A$ and $A \in NP \Rightarrow B \in NP$.*

12.3.1 NP-completeness

Definition 12.7 (i) *A is NP-hard if $B \preccurlyeq_p A$ for any $B \in NP$.*
(ii) *A is NP-complete if $A \in NP$ and A is NP-hard, i.e. $B \preccurlyeq_p A$ for any $B \in NP$.*

Theorem 12.8 *For any NP-complete problem A: $A \in P \Leftrightarrow P = NP$.*

Theorem 12.9 (Cook, 1971) *NP-complete problems exist. In particular, 3SAT is an NP-complete problem.*

Cook's theorem heavily relies on a precise definition of the underlying model of computation and on capturing the computational steps of an algorithm by stating what happens with every 'bit', in a long but polynomially bounded series of logical implications.

Theorem 12.10 (Ladner, 1976) *If $P \neq NP$, then there exist infinitely many problems in NP that do not belong to P and that are not NP-complete (see figure 12-1).*

Proving that problems are NP-complete can be done by either doing Cook's proof in each separate case or using the following essential Lemma. Suppose we want to prove that problem $A \in NP$ is NP-complete.

Lemma 12.11 *Suppose $A \in NP$ and B is NP-complete. If $B \preccurlyeq_p A$ then A is also NP-complete.*

Proof: Let B be NP-complete. According to definition 12.7, we have for any $C \in NP$ that $C \preccurlyeq_p B$. As $B \preccurlyeq_p A$, it follows by transitivity that for every $C \in NP$ also $C \preccurlyeq_p A$. From definition 12.7 it follows A is NP-complete. ■

Theorem 12.12 (Karp, 1972) *The following decision problems are NP-complete:*
(i) *Vertex Cover,*
(ii) *Independent Set,*
(iii) *Clique ('Given a graph G, determine whether there exists a clique of size $\geq k$'),*
(iv) *Set Cover,*
(v) *Dominating Set,*
(vi) *0-1 Linear Programming,*
(vii) *Integer Linear Programming.*

Proof: All problems are easily seen to be in NP, except Integer Linear Programming. However, even this problem can be shown to be in NP. The NP-completeness of all problems follows by transitivity from the NP-completeness of 3SAT, using reductions proved in earlier lectures (which can be verified to be polynomial time reductions). ■

12.3.2 Consequences of NP-completeness

We now see that the acclaimed computational hardness of these (and many other) problems is related in a surprising manner: if any of the problems listed in Theorem 12.12 has a polynomial-time solution, then all of them have and $P = NP$.

The theory of NP-completeness became widespread with the book of Garey & Johnson. It is estimated that currently 'thousands' of problems in algorithmic modeling have been identified as being NP-complete or NP-hard.

Proving that a problem is NP-complete or NP-hard does *not* mean that it can't be solved efficiently in practice. Perhaps we only need to solve 'small' instances and never run into the bad asymptotic behaviour that non-polynomial time solutions invariably have. Perhaps good heuristics can be found that do very well on larger problem instances if one is satisfied with answers that are only approximately optimal ('approximation algorithms').

In particular, the NP-completeness of a problem means no more and no less than that the problem belongs to an entire class of 'equivalent problems' for which other people haven't found polynomial-time algorithms until now either.

12.4 Threshold phenomena

In algorithmic modeling one frequently runs into problem where a small change in parameters can mean the difference between polynomial-time solvability and NP-hardness. We discuss 4 examples.

12.4.1 Satisfiability

In the previous lecture we saw that $2SAT \in P$. On the other hand $3SAT$ is NP-complete.

12.4.2 Graph Coloring

A k-coloring of a graph G is an assignment of colors to nodes such that (i) each node gets precisely one color, (ii) adjacent nodes get different colors, and (iii) no more than k distinct colors are used. In algorithmic modeling the colors could e.g. represent timeslots for activities or real colors (in cartographic contexts). Graph Coloring is the following problem: given a graph G and an integer k, can G be k-colored. Karp (1972) proved: *Graph Coloring is NP-complete.*

The case of planar graphs is especially interesting and shows that the dividing line between known complexities can be quite 'thin'.

- By the solution of the '4-color problem' by Appel and Haken in 1977, every planar graph is 4-colorable. Robertson *et al.* proved in 1996 that there is a polynomial-time algorithm for 4-coloring every planar graph.
- The problem of deciding whether a planar graph can be 3-colored is NP-complete (Garey, Johnson and Stockmeyer, 1976).
- A (planar) graph is 2 colorable if and only if it is bipartite. It can be decided in easy polynomial time whether a graph is bipartite or not.

12.4.3 Multi-partite matching

In a previous lecture we saw the *(Perfect) Bipartite Matching* problem and proved it to be polynomially solvable by results from the theory of maximum flows. Now consider the *(Perfect) Tripartite Matching* problem, also known as '3-Dimensional Matching':

> Given
>
> > sets X, Y, Z with $|X| = |Y| = |Z| = q$, and a set S of triples $\subseteq X \times Y \times Z$.
>
> decide whether there exists a 'perfect matching' $M \subseteq S$ such that:
>
> > $|M| = q$, and every element of X, Y and Z occurs exactly once in a triple $\in M$.

Tripartite Matching has interpretations in algorithmic modeling. For example, let X be a set of q boys, Y a set of q girls and Z a set of q appartments. Let S be a set of triples indicating preferences of the sort: boy x and girl y would like to share appartment z. We leave the further interpretation to the reader.

Theorem 12.13 (Karp,1972) *Tripartite Matching is NP-complete.*

12.4.4 Set Cover

Consider the case of the Set Cover Problem in which all subsets have exactly k elements. This problem is called the k-Set Cover problem. The variant in which every element should be covered only *once* is called the k-Set Packing problem. For $k = 3$ both problems are already computationally hard.

Theorem 12.14 *The following problems are NP-complete:*
(i) *3-Set Cover,*
(ii) *3-Set Packing.*

Proof: Both decision problems are easily seen to be $\in NP$. To prove NP-completeness we show that Tripartite Matching $\preccurlyeq_p$ 3-Set Cover. By theorem 12.13, Tripartite Matching is NP-complete and thus this reduction would indeed prove the desired result.

Consider any instance X, Y, Z and $S \subseteq X \times Y \times Z$ of Tripartite Matching. Let W = 'the disjoint union $X \cup Y \cup Z$' (this universe has $= 3q$ elements). Create for every $(x, y, z) \in S$ a subset $\{x, y, z\} \subseteq W$. A perfect tripartite match exists if and only if there is a set cover of W with $\leq q$ [1] subsets. This is also a reduction to 3-Set Packing. The reduction is clearly computable in polynomial time. ■

Exercise. Prove from theorem 12.14 that the general Set Cover and Set Packing problems are NP-complete.

What about k-Set Cover and k-Set Packing for $k = 2$?

Theorem 12.15 *The following problem are in P:*
(i) *2-Set Cover,*
(ii) *2-Set Packing.*

Proof: Consider an instance of the 2-Set Packing problem. Create a network G as follows. The nodes are the elements of the set universe. For every 2 elements

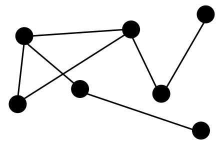

Figure 12-2: Reduction of Set Packing to Perfect Matching

$\{i, j\}$ that belong to the same set we add an edge (i, j) to the graph. The 2-Set Packing Problem is equal to the Perfect Matching Problem for G, which is known to be in P.

Consider the same construction. The 2-Set Cover problem is equivalent to the problem of finding a minimum set of edges that covers the nodes of G. This is the classical *Edge Cover* problem for G, which is known to be in P also by a result of Gallai (1959). ■

[1] Because $|universe| = 3q$ and each subset covers 3 elements this is $= q$.

We examine Gallai's result in more detail. Consider the following algorithm.

> **Algorithm** EC
>
> Let G be a network.
>
> **compute** a maximum matching M in G.
>
> *Note: the nodes that are still free form an independent set in G.*
>
> **return** the edges of M, and for every node u not covered by the matching one edge incident to u.

Theorem 12.16 *Algorithm EC computes a minimum edge cover, and runs in easy polynomial time.*

Proof: Let m be the size of the maximum matching M that was found, and q be the size of a minimum edge cover. The cover that was constructed has the size: $m + (n - 2m) = n - m$.

Consider a minimum edge cover, say of size q. The cover cannot have chains of length 3 and thus is the union of, say, s stars that do not touch. (A 'star' is a node together with a number of incident edges.) Choosing an edge from every star will give a matching, thus $s \leq m$. On the other hand, by counting nodes as covered by the stars:

$$n = s + q \Rightarrow q = n - s \geq n - m.$$

Thus, the edge cover constructed is in fact minimum. The computation is polynomial, because maximum matchings in general networks can be computed in polynomial time. ■

Corollary 12.17 (Gallai identities) *For every undirected n-node graph one has:*
(i) $n = |maximum\_independent\_set| + |minimum\_vertex\_cover|$.
(ii) $n = |maximum\_matching| + |minimum\_edge\_cover|$.

Proof: *(i)* follows by the complementarity of independent sets and vertex covers in networks. *(ii)* follows from the proof of Theorem 12.16. ■

12.5 NP-completeness of the Traveling Salesman Problem

We show that the essence of most vehicle routing problems is NP-hard, by proving that the TSP is NP-complete. The NP-completeness proof will be done using two reductions. The first reduction will be from Vertex Cover, which we already saw to be NP-complete, to Hamiltonian Cycle (from Garey and Johnson [4], also Skiena [9]), the second reduction will be from Hamiltonian Cycle to TSP.

The Hamiltonian Cycle problem is the problem of deciding whether there exists a cycle in a given graph that visits each node exactly once. The TSP is defined for weighted complete graphs and asks whether there exists a 'weighted Hamiltonian Cycle' of weight $\leq K$. Both problems are easily seen to be $\in NP$.

Lemma 12.18 *Vertex Cover $\preccurlyeq_p$ Hamiltonian Cycle.*

Proof: Let $G = \langle V, E\rangle$ be the graph for which we need to find a vertex cover. Let k be the size of the vertex cover. Create a graph $G' = \langle V', E'\rangle$ with the following properties:

- For each edge $(u, v) \in E$ add the vertices $\{u_1, \ldots, u_6, v_1, \ldots, v_6\}$ to V'. Connect these vertices by adding edges $\{(u_1, v_3), (u_3, v_1), (u_4, v_6), (u_6, v_4)\}$ and adding edges $\{(u_1, u_2), (u_2, u_3), (u_3, u_4), (u_4, u_5), (u_5, u_6), (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_5, v_6)\}$. The vertices $\{u_1, \ldots, u_6, v_1, \ldots, v_6\}$ and the edges between them are called a "gadget". Nodes u_1, u_6, v_1 and v_6 are called "endpoints" of the gadget. Nodes $\{u_1, \ldots, u_6\}$ are associated with vertex $u \in V$ and nodes $\{v_1, \ldots, v_6\}$ are associated with $v \in V$. Figure 12-3 shows this graphically.

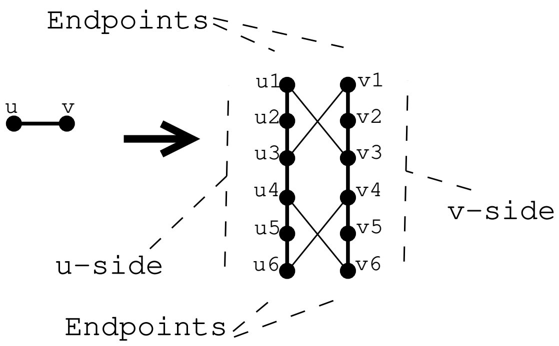

Figure 12-3: (u, v) is translated to a gadget

- Each side of each gadget of each edge is associated with a vertex $v \in V$. For every vertex, link all the gadgets sides associated with this vertex into a chain by adding edges between the endpoints. Figure 12-4 shows how this is done.

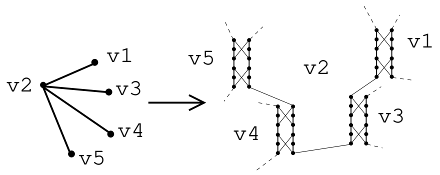

Figure 12-4: a chain of gadgets from edges adjacent to v

- Add k "selector" vertices to V'. Connect both endpoints of all chains to each of the k selector vertices. Figure 12-5 shows the graph resulting from translating a simple Vertex Cover problem with $k = 2$.

G' has $12|E|+k$ vertices and $14|E|+2k|V|$ edges. This graph can be constructed in polynomial bounded time.

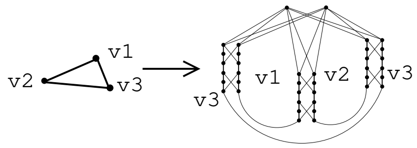

Figure 12-5: translating a Vertex Cover with $k = 2$

Observation 12.19 *Given a gadget with vertices* $\{u_1, \ldots, u_6, v_1, \ldots, v_6\}$*, edges* $\{(u_1, v_3), (u_3, v_1), (u_4, v_6), (u_6, v_4), (u_1, u_2), \ldots, (u_5, u_6), (v_1, v_2), \ldots (v_5, v_6)\}$ *and entering and exiting only at the endpoints* u_1*,* u_6*,* v_1 *and* v_6*, there are only 3 possible ways to visit all the vertices:*

- $\{u_1; u_2; \ldots; u_6$ *and* $v_1; v_2; \ldots; v_6\}$
- $\{u_1; u_2; u_3; v_1; v_2; \ldots; v_6; u_4; u_5; u_6\}$
- $\{v_1; v_2; v_3; u_1; u_2; \ldots; u_6; v_4; v_5; v_6\}$

From this we can conclude:

- when entering a gadget from an endpoint associated with a vertex $u \in V$ and visiting all vertices, the only way to leave the gadget is via the other endpoint associated with u.
- when entering and leaving a gadget, representing $(u, v) \in E$ via the endpoints associated with u we can choose whether we want to visit the vertices $\{v_1, \ldots, v_6\}$ immediately or visit these vertices from the endpoints associated with v.

Claim 12.20 *G' has a Hamiltonian Cycle iff G has a Vertex Cover of size k.*

Proof:($\Rightarrow$) Suppose $\{v_1; v_2; \ldots; v_n\}$ is a Hamiltonian Cycle. Assume that it starts at one of the k selector vertices. Since the tour is a Hamiltonian Cycle all gadgets are traversed. The k-chains correspond to the k vertices in the cover. To avoid visiting a vertex more than once, each chain is associated with a selector vertex.

($\Leftarrow$) Suppose we have a vertex cover of size $\leq k$. We can alway add more vertices to the cover to bring it up to size k. For each vertex in the cover, start traversing the chain. At each entry point to a gadget, check if the other vertex is in the cover and traverse the gadget accordingly. ■

Thus Vertex Cover can be reduced to a corresponding Hamiltonian Cycle, and the reduction can be done in polynomial time. Thus Vertex Cover $\preccurlyeq_p$ Hamiltonian Cycle. ■

Lemma 12.21 *Hamiltonian Cycle $\preccurlyeq_p$ TSP.*

Proof: Given a graph $G = \langle V, E\rangle$ over the set of nodes $V = \{1, 2, \ldots, n\}$, we define a complete graph $G' = \langle V, E'\rangle$ with the following cost function for the edges:

$$C(i,j) = \begin{cases} 1 & (i,j) \in E \\ 2 & (i,j) \notin E \end{cases} \tag{1}$$

We show that a Hamiltonian Cycle exists in G iff there exists a TSP tour in G' with cost $\leq n$. ($\Rightarrow$) If there is a Hamiltonian cycle in G, then the same cycle exists in G' and its price is n. Hence, there exists a TSP tour in G' with cost $\leq n$. ($\Leftarrow$) If there is a TSP tour in G' with cost $\leq n$, its length is necessarily n and therefore the price of every edge in this tour is 1. According to the reduction, these edges exist also in G, and therefore this Hamiltonian cycle exists also in G.

Building G' and defining the cost function C can be done in polynomial time, and therefore the reduction is polynomial. ∎

We conclude:

Theorem 12.22 *The Traveling Salesman Problem is NP-complete.*

As the TSP is a subproblem of many vehicle routing problems, it follows that most vehicle routing problems are at least NP-hard.

References

[1] Clay Mathematics Institute. P vs NP, *Millennium prize problems*, *www.claymath.org/Millennium_Prize_Problems/P_vs_NP*.

[2] S.A. Cook. The complexity of theorem proving procedures. In: *Proc. 3rd Annual ACM Symposium on Theory of Computing*, ACM Press, New York, 1971, pp 151-156.

[3] T. Gallai. Über extreme Punkt- und Kantenmengen. *Annales Univ. Sci. Budapestinensis de Rolando Eotvos, Secto Math.* 2 (1959) 133-138.

[4] M.R. Garey, D.S. Johnson. *Computers and intractability - A guide to the theory of NP-completeness*, W.H. Freeman and Company, San Francisco, 1979.

[5] M.R. Garey, D.S. Johnson and L. Stockmeyer. Some simplified NP-complete graph problems, *Theoretical Computer Science* 1 (1976) 237-267.

[6] R.M. Karp. Reducibility among combinatorial problems. In: R.E. Miller and J.W. Thatcher (Eds.), *Complexity of computer computations*, Plenum Press, New York, 1972, pp. 85-103.

[7] R.E. Ladner. On the structure of polynomial time reducibility, *Journal of the ACM* 22 (1975) 155-171.

[8] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas. The four color theorem, see: *http://www.math.gatech.edu/~thomas/FC/fourcolor.html#Algorithm*.

[9] S. Skiena. The Stony Brook Algorithm Repository, Lecture 22 - techniques for proving hardness, 2001, *www.cs.sunysb.edu/~algorith/lectures-good/node22.html*.