Complexity Theory - NP Completeness

Daniel Vlasits

Abstract

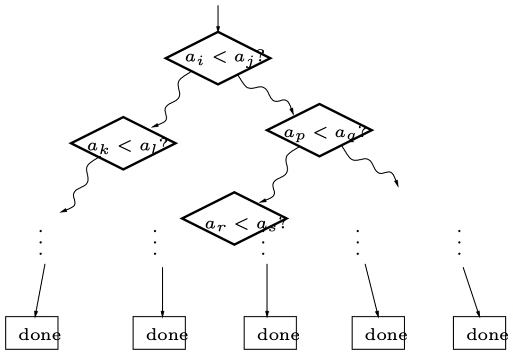
The aim of complexity theory is to understand what makes certain problems difficult to solve algorithmically. When we say that a problem is difficult, we mean not that it is hard to come up with an algorithm for solving a problem, but that any algorithm we can devise is inefficient, requiring inordinate amount of resources such as time and memory space. In this course, we aim at a theoretical understanding of why some problems appear to be inherently difficult.

1. Intro

Definition 1.1. Big O notation

- $f = O(g) \iff \exists n_0 \in \mathbb{N}, \exists$ **such that** $\forall n > n_0, f(n) \leq cg(n)$
- $f = \Omega(g) \iff \exists n_0 \in \mathbb{N}, \exists c > 0$ **such that** $\forall n > n_0, f(n) \geq cg(n)$
- $f = \Theta(g) \iff f = O(g)$ and $f = \Omega(g)$

Theorem 1.2. Comparison based sorting is $\Omega(n \log n)$. Assume n distinct numbers



n! different ways the initial collected of n numbers could be sorted. Therefore computation tree must have n! leaves. The tree has height $\log_2(n!) = \Theta(n \log n)$. Proof of this left as excercise.

Remark 1.3. Much easier to show upper bounds - just give an algorithm! Much harder to show lower bounds

Remark 1.4. Interesting to think how this works with bucket and radix sort?

Remark 1.5. Famous example is the Traveling Salsemen problem

Definition 1.6. Decidability is the problem of taking something and deciding true or false. So its not about optimizing. But acc or rej for example you take salsemen problem you say is there a solution less than a value?

Definition 1.7. Turing Machine

- Q - a finite set of states
- Σ - a finite set of symbols, disjoint from Q and including a distinguished blank symbol $\sqcup$ and a distinguished left-end marker $\triangleright$
- $s \in Q$ - an initial state
- $\delta : (Q \times \Sigma) \rightarrow Q \cup \{acc, rej\} \times \Sigma \times \{L, R, S\}$ - a transition function specifying all details

Definition 1.8. Some important language around turing machines:

- A *Configuration* is a triple (q, w, u), where $q \in Q$ and $w, u \in \Sigma^*$. Where the state is q. The string wu is on the tape and the head points to the last symbol in w.
- Given machine $M = (Q, \Sigma, s, \delta)$ we have a configuration yields one step written: $(q, w, u) \rightarrow_M (q', w', u')$ Where you can fill in the details of what is acceptable.
- $\rightarrow_M^*$ is the reflexive and transitive closure of $\rightarrow_M$
- Every machine M defines a language $L(M) \subseteq \Sigma^*$ which it accepts. This is defined by:

$$L(M) = \{x | (s, \triangleright, x) \rightarrow_M^* (acc, w, u) \text{ for some } w \text{ and } u\}$$

Definition 1.9. TIME and SPACE

- TIME($f(n)$) is the set of all languages accepted by some machine with running time $O(f(n))$
- SPACE($f(n)$) is the set of all languages accepted by a machine which uses at most $O(f(n))$ tape cells on inputs of length n

Definition 1.10. Nondeterminism

Non-deterministic Turing Machine

$$\delta \subseteq (Q \times \Sigma) \times (Q \cup \{acc, req\} \times \Sigma \times \{R, L, S\})$$

Now the language is such that we must have one path which leads to an accept.

$$\begin{tikzcd}
 & (s, \triangleright, x) \arrow[dl] \arrow[d] \arrow[dr] & \\
(q_0, u_0, w_0) \arrow[d] & (q_1, u_1, w_1) \arrow[dd] \arrow[ddr] & (q_2, u_2, w_2) \arrow[d] \\
(q_{00}, u_{00}, w_{00}) & & (\text{rej}, u_2, w_2) \\
\vdots & (q_{10}, u_{10}, w_{10}) & (q_{11}, u_{11}, w_{11}) \\
(\text{acc}, \ldots) & & \vdots
\end{tikzcd}$$

Remark 1.11. A deterministic machine can always simulate a non-deterministic one as it can perform a breadth first search. We say that a non-deterministic machine runs in polynomial time says that the height of the computation tree on input string x is bounded by a polynomial $p(|x|)$, in $|x|$ - the length of x. However a breadth first search of a tree of height $p(|x|)$ is $O(2^{cp(|x|)})$ for some c

2. Complexity Class

Definition 2.1. The class of languages decidable in polynomial time

$$\boldsymbol{P} = \bigcup_{k=1}^{\infty} \text{TIME}(\boldsymbol{n^k})$$

We call a problem *tractable* if it is in this set. If you can prove it is not in this set we call it *intractable.* Up for debate if this really should be called feasable if n^{1000} but polynomials provide a very good basis for analysis for example we've found that taking this class remains the same over different models of computation which for example does not hold with linear class.

3. Example Problems

Definition 3.1. The Reachability problem is defined as the problem where, given a directed graph $G=(V,E)$, and two nodes $a,b \in V$ need to decide if there is a path from a to b in G. Simple algorithm for this:

1. mark node a, leaving other nodes unmarked, and initialise set S to {a}
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i,j) and j is unmarked, mark j and add j to S
3. if b is marked, accept else reject

Definition 3.2. Satisfiability

Given an Boolean expression $(x \wedge y) \wedge (\neg y)$

Remark 3.3. You can see this has a search space of possible solutions. You could view this as one way to find a solution is to generate all possible answers and then test if any of them work.

Definition 3.4. A *verifier* for a language

$$L = \{x | (x,c) \text{is accepted by V for some c}\}$$

If V runs in polynomial time in the length of x, then L is polynomialy verifiable.

Definition 3.5. NTIME and NP

NTIME(f) the set of languages f which are accepted by a *nondeterministic* Turing M, such that for every $x \in L$ of length n, there is an accepting computation of M on x of length at most f(n). Then

$$\boldsymbol{NP} = \bigcup_{k=1}^{\infty} \textbf{NTIME}(\boldsymbol{n^k})$$

THEOREM 3.6. L is polynomially verifiable if and only if it is in NP.

1. input x of length n
2. nondeterministically guess of length $\leq p(n)$
3. run V on (x,c)

DEFINITION 3.7. Reduction

We have a function f is a reduction from L_1 to L_2 if:

$$f(x) \in L_2 \iff x \in L_1$$

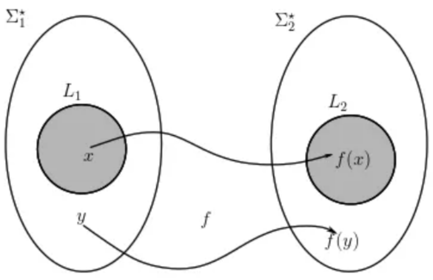


Figure 5: Reduction

These functions are vital as for example: if L_1 is undecidable (can't be solved) and we get a function from L_1 to L_2 then we have a proof that L_2 is also undecidable.

However, when we are talking about polynomial time computability, we consider reductions that can be computed within bounded resources. If f is a reduction from L_1 to L_2 and f is computable by an algorithm in poly time. we say that L_1 is *polynomial time reducible* to L_2, which we write as

$$L_1 \leq_P L_2$$

4. PROBLEM SHEET 1

4.1. Which of these is true? Prove it:

1. $|sin(n)| = O(1)$
2. $|sin(n)| = \Theta(1)$
3. $200 + sin(n) = \Theta(1)$
4. $123456n + 654321 = \Theta(n)$
5. $2n - 7 = O(17n^2)$
6. $\log(n) = \Theta(n)$
7. $n^{100} = O(2^n)$

8. $1+100/n = \Theta(1)$

4.2. This is an algorithm for picking the median of a list in $O(n)$

1. Split the list into groups of five
2. Sort each group of 5 and pick the median of each group
3. Recurse onto this list of medians to pick the median of these numbers
4. Use that median to partition the list in half and then recurse into the correct side

Show it is O(n)

4.3. Show: $\log_2(n!) = \Theta(n \log n)$

4.4. Show that travelling salseman is $\Omega(n \log n)$

Can you improve the bound?

4.5. We say that a propositional formula ϕ is in 2CNF if it is a conjunction of clauses, each of which contains exactly 2 literals. The point of this problem is to show that the satisfiability problem for formulas in 2CNF can be solved by a polynomial time algorithm.

First note that any clause with 2 literals can be written as an implication in exactly two ways. For example p or (not q) is equivalent to $q \to p$ and not $p \to$ not q. And similar for the rest.

For any formula ϕ, define the directed graph G_ϕ, to be the graph whose set of vertices is he set of all literals that occur in ϕ and in which there is an edge from literal x to literal y if, and only if, the implication $(x \to y)$ is equivalent to one of the clauses in ϕ.

1. If ϕ has n variables and m clauses, give an upper bound on the number of vertices and edges in G_ϕ
2. Show that ϕ is *unsatisfiable* if, and only if, there is a literal x such that there is a path in in G_ϕ from x to (not x) *and* a path from (not x) to x.
3. Give an algorithm for verifying that a graph G_ϕ satisfies the property stated in (b) above. What is the complexity of the algorithm?
4. From (3) deduce that there is a polynomial time algorithm for testing whether or not a 2CNF propositional formula is satisfiable.
5. Why does this no longer work with 3 literals?

4.6. A clause (i.e. a disjunction of literals) is called a Horn clause, if it contains *at most one* positive litera. Such a clause can be written as an implication: (x or (not y)) or (not (w) or (not z)). is equivalent to ((y and w and z) $\to$ x. HORNSTAT is the problem of deciding whether a given Boolean expression that is a conjunction of Horn clauses is satisfiable.

1. Show that there is a polynomial time algorithm for solving HORNSAT.

4.7. In general, k-colourability is the problem of deciding, given a graph $G = (V,E)$, whether there is a colouring $\chi : V \to \{1,\ldots,k\}$ of the vertices such that if $(u,v) \in E$, then $\chi(u) \neq \chi(v)$. That is, adjacent vertices do not have the same colour.

1. Show that there is a polynomial time algorithm for solving 2-colourability
2. Show that, for each k, k-colourability is reducible to k + 1-colourability.

5. NP-Completeness

Reductions give us the tools to make statements about the *relative complexity* of problems. Even if we don't know whether or not there is a polynomial time algorithm for L_2, if $L_1 \leq_P L_2$, we can say that if there were one, there would also be one for L_1

Definition 5.1.
- A language L is NP-hard if for every language $A \in NP$, $A \leq_P L$
- A language L is NP-Complete if it is in NP and it is NP-Hard

Theorem 5.2. Proof that SAT is NP-Complete Firstly SAT must be in NP as it is easily verifiable if I provide the value of each variable for satisfiability

Proof it is NP-Hard:

Consider a language in NP. It has an associated Non-determinstic Machine which must accept everything in the language in time $p(|x|)$. For each $x \in \Sigma^*$ we must provide a boolean expression f(x) which is satisfiable if, and only if, there is an accepting computation of M on x. We construct f(x) using the following variables:

$$S_{i,q} \text{ for each } i \leq n^k \text{ and } q \in Q$$
$$T_{i,j,\sigma} \text{ for each } i,j \leq n^k \text{ and } \sigma \in \Sigma$$
$$H_{i,j} \text{ for each } i,j \leq n^k$$

The total number of variables is $|Q|n^k + |\Sigma|n^{2k} + n^{2k}$. The intended reading of these variables is that $S_{i,q}$ will be true if the machine at time i is in state q; $T_{i,j,\sigma}$ will be set to true if at time i

6. NP-Complete Free for all

Lets talk about problems and their reducibleness!

Above can be easily adjusted to be in CNF (Conjunctive Normal Form). Call this CNF-SAT

Theorem 6.1. 3SAT is NP-Complete 3CNF is in CNF form, and each clause is the disjunction of no more than 3 literals. 3SAT is the set of expressions in 3CNF that are satisfiable.

It is *not* the case that every Boolean expression is equivalent to one in 3CNF. However, we can say that for every CNF expression ϕ, there is an expression ϕ' in 3CNF so that ϕ' is satisfiable if, and only if, ϕ is. Moreover, there is an algorithm, running in polynomial time, which will convert ϕ to ϕ'.

Suppose we have a clause C:

$$C = (l_1 \vee l_2 \vee l_3 \vee l_4))$$

Introducing new variables n_1 and n_2, we can write down an expression in 3CNF

$$3C = (l_1 \vee l_2 \vee n_1) \wedge (\neg n_1 \vee l_3 \vee n_2) \wedge (\neg n_2 \vee l_4)$$

These expressions are not equivalent, but 3C is satisfiable if, and only if, C is satisfiable. also NP-complete.

We have poly time reduction from CNF-SAT to 3SAT. as CNF-SAT is NP-Complete 3SAT is

Theorem 6.2. Independent Set: Let $G=(V,E)$ be an undirected graph with a set V of vertices, and E of edges. We say that $X \subseteq V$ is an *independent set* if there are no edges $u,$ v in E, for any $u,v \in X$. Given a graph G, find the lagest independent set. Instead of this optimisation problem, consider a decision problem which we call IND, which is defined as: The set of pairs (G,K), where G is a graph, and K is an integer, such that G contains an independent set with K or more vertices.

We turn the question into a yes/no question.

Clearly in NP as can non-deterministically try.

Need to create a reduction from 3SAT to IND.

Create triangles labelled with the literals. And connect any edges where one represents the complement of the other. Can then feed this to IND with pair (G,m) where m is the number of triangles.

Need to show poly.

Need to show G contains independent set of m vertices if and only if equation satisfiable.

Theorem 6.3. Clique is NP-complete.

Clique is the problem of finding the largest subset of G such that all vertices are connected to all others in the Clique. But we will look at the decision problem (G, K) check if there is a Clique greater or equal to K.

Theorem 6.4. Graph Colourability. Given a graph G = (V, E), call a colouring of G an assignment of colours to the vertices of G such that no edge connects two vertices of the same colour. Say that G is k-colourable if there is a colouring of G which uses no more than k colours.

Decision problem is given (V, E

7. Problem Sheet 2

7.1. Given a graph $G=(V,E)$, a set $U \subseteq V$ of vertices is called a *vertex cover* of G if, for each edge $(u,v) \in E$, either $u \in U$ or $v \in U$. That is, each edge has at least one end point in U. The decision problem V-COVER is defined: given a graph G = (V,E), and an integer K, does G contain a vertex cover with K or fewer elements?

1. Show a polynomial time reduction from IND to V-COVER.
2. Use (1) to argue that V-COVER is NP-complete.

7.2. The problem of four dimensional matching, 4DM, is defined analogously with 3DM:

TODO

7.3. Given a graph $G=(V,E)$, a *source vertex* $s \in V$ and a *target vertex* $t \in V$, a *Hamilitonian Path* from s to t in G is a path that begins at s, ends at t and visits every vertex in V exactly once. We define the decision problem HamPath as: Given a graph G = (V,E) and vertices $s,t \in V$, does G contain a Hamiltonian path from s to t?

1. Give a polynomial time reduction from the Hamiltonian cycle problem to HamPath

2. Give a polynomial time reduction fom HamPath to the problem of determining whether a graph has a hamiltonian cycle.

7.4. Every problem in NP is reducible to SAT. Sometimes it is easy to give an explicit reduction. In this exercisyou are asked to give such explicit reductions for two graph problems: 3-Col and HAM. That is,

1. describe how to obtain, for any graph G = (V, E), a Boolean expression ϕ_G so that ϕ_G is satisfiable if, and only if, G contains a Hamiltonian cycle.

7.5. An instance of a linear programming problem consists of a set $X = \{x_1, \ldots, x_n\}$ of variables and a set of constraints, each of the form

$$\sum_{1 \leq i \leq n} c_i x_i \leq b$$

where each c_i and b is an integer. The 0-1 Integer Linear Programming Feasability problem is, to determine, given such a linear progrmming problem, whether there is an assignment of values from the set {0, 1} to the variables in X so that substituting these values in the constrains leads to all constraints being simultaneously satisfied. Prove that this problem is NP-Complete

7.6. *Self-reducibility* refers to the property of some problems in $L \in NP$, where the problem of finding a *witness* for the membership of an input x in L can be reduced to the decision problem for L. This question asks you to give such arguments in two specific instances.

1. Show that, given an oracle (i.e. a black box) for deciding whether a given graph G = (V, E) is Hamiltonian, there is a polynomial-time algorithm that, on input G, outputs a Hamiltonian cycle in G if one exists.
2. Show that, given an oracle for deciding whether a given graph G is 3-colourable, there is a polynomial-time algorthm that, on input G, produces a valid 3-colouring of G in one exists.

8. Space Complexity

Definition 8.1. NSPACE($f(n)$) is the set of languages accepted by a *nondeterministic* machine which uses at most $O(f(n))$ tape cells on inputs of length n.

Note this can be smaller than the input as that is stored on a seperate read only tape.

Definition 8.2. Space Complexity Classes

- $L = SPACE(\log n)$ The class of languages decidable using logarithmic workspace.
- $NL = NSPACE(\log n)$ The class of languages recognisable by a nondeterministic machine using logarithmic workspace
- $PSPACE = \cup_{k=1}^{\infty} SPACE(n^k)$ The class of languages decidable in polynomial space.
- $NPSPACE = \cup_{k=1}^{\infty} NSPACE(n^k)$ The class of langugages recognisable by a nondeterministic machine using polynomial space.

Can show:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE$$

Remark 8.3. We assume that whatever function is within the big-O notation that we can calculate the f(n) reasonably where that means: There is a deterministic machine M which, on any input of length n, replaces the input with the string $0^{f(n)}$, and M runs in time $O(n+f(n)$ and uses $O(f(n))$ *work space*

Intuition is that computing the function f shouldn't require more resources than the limit imposed by f itself. I.e. so we can use it in our machines.

Theorem 8.4.

- $SPACE(f(n)) \subseteq NSPACE(f(n))$
- $TIME(f(n)) \subseteq NTIME(f(n)$

The two easy inclusions

Theorem 8.5. $NTIME(f(n)) \subseteq SPACE(f(n))$

Can do this with DFS using f(n) space.

Theorem 8.6. $NPSACE(\log n) \subseteq TIME(k^{\log n+f(n)}$

Suppose M is a nondeterministic machine working with workspace bounded by $f(n)$ for inputs of length n. For a given input string x of length n there is a fixed finite number of configurations of M that are possible.

- The finite state control can be in any of q states. (q does not depend on x)
- The work tape can have one of $s^{f(n)}$ strings on it, where s is the number of distinct symbols on the tape alphabet
- The head can be in one of n positions
- The head on the work tape can be in one of f(n) positions

Therefore! We have a total number of configurations of no more than:

$$qnf(n)s^{f(n)}$$

. For some constant c, we have $nc^{f(n)}$

Then we have that M accepts x if, and only if, there is a path from the starting configuration $(s,\triangleright,x,\triangleright,\epsilon)$ to an accepting configuration.

We have deterministic reachability in $O(g^2)$ and therefore we have that time is at most $c(n^{f(n)})^2 = k^{\log n+f(n)}$ So we get: $NL \subseteq P$ AND $NPSPACE \subseteq EXP$

Theorem 8.7. Savitch's Theorem: Reachability in $O((\log n)^2)$

Code for this:

```
Path(a,b,i).

 if i=1 and a =/= b and there is no edge (a,b)
    then reject
 else if there is an edge (a,b) or a=b
    then accept
 else for each vertex x
       if Path(a,x,floor(i/2)) and Path(x,b,ceil(i/2)) then accept
```

Theorem 8.8. NSPACE($f(n)$) $\subseteq$ SPACE($f(n)^2$).

Once again we have that the configuration graph has $g = c^{\log n + f(n)}$ nodes.

Therefore for reachability:

So we have that:

$$O((\log g)^2) = O((\log n + f(n))^2) = O((f(n))^2)$$

Whenever we need to see if a pair we look at the machine.

And from this we get that PSPACE = NPSPACE

9. Problem Sheet 3

9.1. Reachability $\in$ NL

Can you come up with an algorithm that shows reachability is in NL:

9.2. We not only have polynomial time reductions but logspace reductions. Which is what we use to get P-Complete problems. Show that lopspace reductions are closed under complementation. I.e. taking two logspace reductions p and g. Show you can create a new turing machine which composes these two machines but remains in logspace.

9.3. Prove $P \neq NP$