Complexity Theory
Lectures 7–12

Lecturer: Dr. Timothy G. Griffin

Slides by Anuj Dawar

Computer Laboratory
University of Cambridge
Easter Term 2009

http://www.cl.cam.ac.uk/teaching/0809/Complexity/

Hamiltonian Graphs

Recall the definition of HAM—the language of Hamiltonian graphs.

Given a graph $G = (V, E)$, a *Hamiltonian cycle* in G is a path in the graph, starting and ending at the same node, such that every node in V appears on the cycle *exactly once*.

A graph is called *Hamiltonian* if it contains a Hamiltonian cycle.

The language HAM is the set of encodings of Hamiltonian graphs.

Hamiltonian Cycle

We can construct a reduction from 3SAT to HAM

Essentially, this involves coding up a Boolean expression as a graph, so that every satisfying truth assignment to the expression corresponds to a Hamiltonian circuit of the graph.

This reduction is much more intricate than the one for IND.

Travelling Salesman

Recall the travelling salesman problem

Given

- V — a set of nodes.
- $c : V \times V \to \mathbb{N}$ — a cost matrix.

Find an ordering $v_1, \ldots, v_n$ of V for which the total cost:

$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$

is the smallest possible.

Travelling Salesman

As with other optimisation problems, we can make a decision problem version of the Travelling Salesman problem.

The problem TSP consists of the set of triples

$$(V, c : V \times V \to \mathbb{N}, t)$$

such that there is a tour of the set of vertices V, which under the cost matrix c, has cost t or less.

Reduction

There is a simple reduction from HAM to TSP, mapping a graph (V, E) to the triple $(V, c : V \times V \to \mathbb{N}, n)$, where

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ 2 & otherwise \end{cases}$$

and n is the size of V.

Sets, Numbers and Scheduling

It is not just problems about formulas and graphs that turn out to be NP-complete.

Literally hundreds of naturally arising problems have been proved NP-complete, in areas involving network design, scheduling, optimisation, data storage and retrieval, artificial intelligence and many others.

Such problems arise naturally whenever we have to construct a solution within constraints, and the most effective way appears to be an exhaustive search of an exponential solution space.

We now examine three more NP-complete problems, whose significance lies in that they have been used to prove a large number of other problems NP-complete, through reductions.

3D Matching

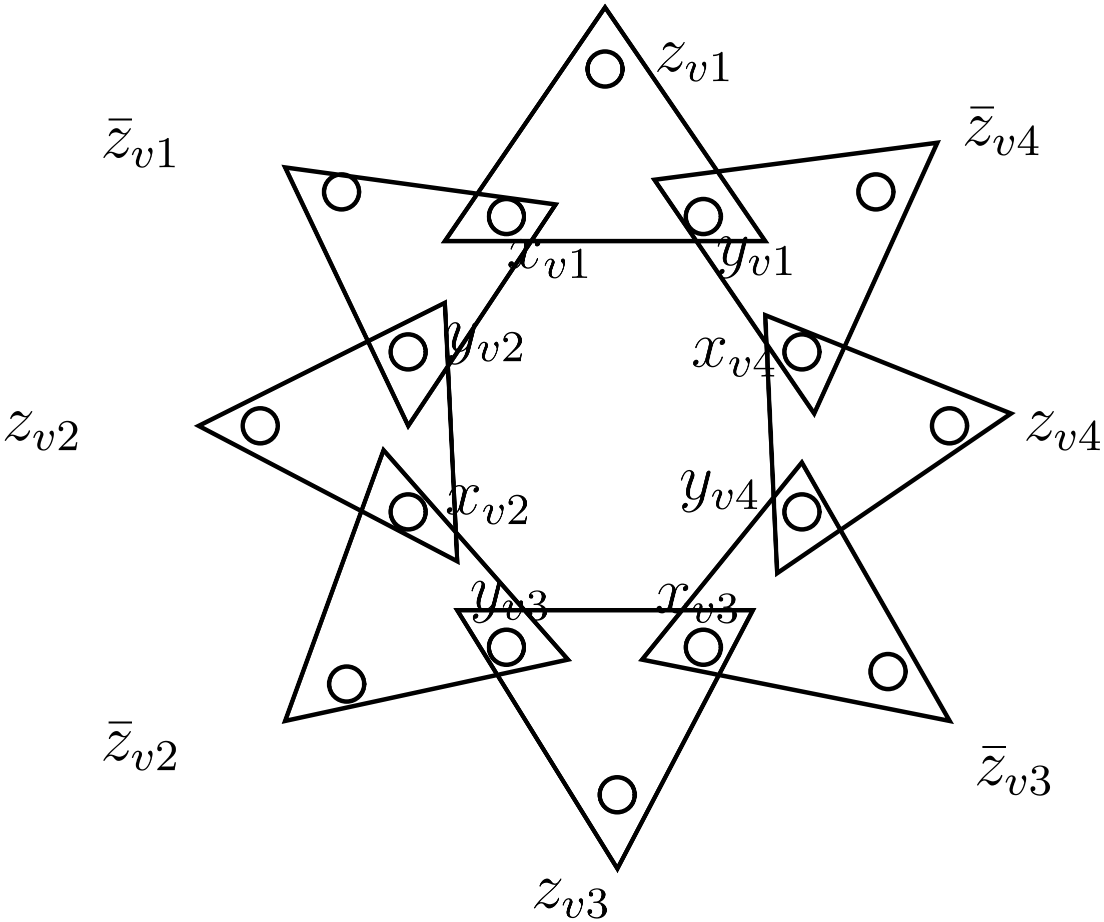
The decision problem of *3D Matching* is defined as:

> Given three disjoint sets X, Y and Z, and a set of triples $M \subseteq X \times Y \times Z$, does M contain a matching?
>
> I.e. is there a subset $M' \subseteq M$, such that each element of X, Y and Z appears in exactly one triple of M'?

We can show that 3DM is NP-complete by a reduction from 3SAT.

Reduction

If a Boolean expression ϕ in 3CNF has n variables, and m clauses, we construct for each variable v the following gadget.



In addition, for every clause c, we have two elements x_c and y_c.

If the literal v occurs in c, we include the triple

$$(x_c, y_c, z_{vc})$$

in M.

Similarly, if $\neg v$ occurs in c, we include the triple

$$(x_c, y_c, \bar{z}_{vc})$$

in M.

Finally, we include extra dummy elements in X and Y to make the numbers match up.

Exact Set Covering

Two other well known problems are proved NP-complete by immediate reduction from 3DM.

Exact Cover by 3-Sets is defined by:

> Given a set U with $3n$ elements, and a collection $S = \{S_1, \ldots, S_m\}$ of three-element subsets of U, is there a sub collection containing exactly n of these sets whose union is all of U?

The reduction from 3DM simply takes $U = X \cup Y \cup Z$, and S to be the collection of three-element subsets resulting from M.

Set Covering

More generally, we have the *Set Covering* problem:

> Given a set U, a collection of $S = \{S_1, \ldots, S_m\}$ subsets of U and an integer budget B, is there a collection of B sets in S whose union is U?

Knapsack

KNAPSACK is a problem which generalises many natural scheduling and optimisation problems, and through reductions has been used to show many such problems NP-complete.

In the problem, we are given n items, each with a positive integer value v_i and weight w_i.

We are also given a maximum total weight W, and a minimum total value V.

> Can we select a subset of the items whose total weight does not exceed W, and whose total value exceeds V?

Reduction

The proof that KNAPSACK is NP-complete is by a reduction from the problem of Exact Cover by 3-Sets.

Given a set $U = \{1, \ldots, 3n\}$ and a collection of 3-element subsets of U, $S = \{S_1, \ldots, S_m\}$.

We map this to an instance of KNAPSACK with m elements each corresponding to one of the S_i, and having weight and value

$$\Sigma_{j \in S_i} (m+1)^{j-1}$$

and set the target weight and value both to

$$\Sigma_{j=0}^{3n-1} (m+1)^j$$

Scheduling

Some examples of the kinds of scheduling tasks that have been proved NP-complete include:

Timetable Design

> Given a set H of *work periods*, a set W of *workers* each with an associated subset of H (available periods), a set T of *tasks* and an assignment $r : W \times T \rightarrow \mathbb{N}$ of *required work*, is there a mapping $f : W \times T \times H \rightarrow \{0, 1\}$ which completes all tasks?

Scheduling

Sequencing with Deadlines

Given a set T of *tasks* and for each task a *length* $l \in \mathbb{N}$, a release time $r \in \mathbb{N}$ and a deadline $d \in \mathbb{N}$, is there a work schedule which completes each task between its release time and its deadline?

Job Scheduling

Given a set T of *tasks*, a number $m \in \mathbb{N}$ of processors a length $l \in \mathbb{N}$ for each task, and an overall deadline $D \in \mathbb{N}$, is there a multi-processor schedule which completes all tasks by the deadline?

Responses to NP-Completeness

Confronted by an NP*-complete problem, say constructing a timetable, what can one do?*

- It's a single instance, does asymptotic complexity matter?
- What's the critical size? Is scalability important?
- Are there guaranteed restrictions on the input? Will a special purpose algorithm suffice?
- Will an approximate solution suffice? Are performance guarantees required?
- Are there useful heuristics that can constrain a search? Ways of ordering choices to control backtracking?

Validity

We define VAL—the set of *valid* Boolean expressions—to be those Boolean expressions for which every assignment of truth values to variables yields an expression equivalent to `true`.

$$\phi \in \mathsf{VAL} \quad \Leftrightarrow \quad \neg\phi \notin \mathsf{SAT}$$

By an exhaustive search algorithm similar to the one for SAT, VAL is in $\mathsf{TIME}(n^2 2^n)$.

Is $\mathsf{VAL} \in \mathsf{NP}$?

Validity

$\overline{\mathsf{VAL}} = \{\phi \mid \phi \notin \mathsf{VAL}\}$—the *complement* of VAL is in NP.

Guess a a *falsifying* truth assignment and verify it.

Such an algorithm does not work for VAL.

In this case, we have to determine whether *every* truth assignment results in `true`—a requirement that does not sit as well with the definition of acceptance by a nondeterministic machine.

Complementation

If we interchange accepting and rejecting states in a deterministic machine that accepts the language L, we get one that accepts $\overline{L}$.

If a language $L \in \mathsf{P}$, then also $\overline{L} \in \mathsf{P}$.

Complexity classes defined in terms of nondeterministic machine models are not necessarily closed under complementation of languages.

Define,

co-NP – the languages whose complements are in NP.

Succinct Certificates

The complexity class NP can be characterised as the collection of languages of the form:

$$L = \{x \mid \exists y R(x, y)\}$$

Where R is a relation on strings satisfying two key conditions

1. R is decidable in polynomial time.
2. R is *polynomially balanced*. That is, there is a polynomial p such that if $R(x, y)$ and the length of x is n, then the length of y is no more than $p(n)$.

Succinct Certificates

y is a *certificate* for the membership of x in L.

Example: If L is SAT, then for a satisfiable expression x, a certificate would be a satisfying truth assignment.

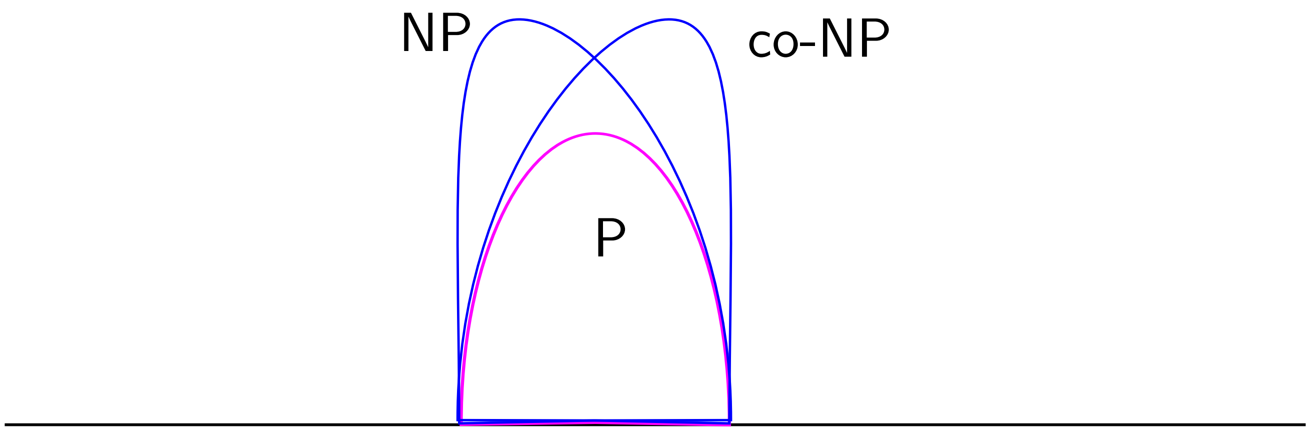
co-NP

As co-NP is the collection of complements of languages in NP, and P is closed under complementation, co-NP can also be characterised as the collection of languages of the form:

$$L = \{x \mid \forall y \, |y| < p(|x|) \rightarrow R'(x, y)\}$$

NP – the collection of languages with succinct certificates of membership.

co-NP – the collection of languages with succinct certificates of disqualification.



Any of the situations is consistent with our present state of knowledge:

- $\mathsf{P} = \mathsf{NP} = \text{co-}\mathsf{NP}$
- $\mathsf{P} = \mathsf{NP} \cap \text{co-}\mathsf{NP} \neq \mathsf{NP} \neq \text{co-}\mathsf{NP}$
- $\mathsf{P} \neq \mathsf{NP} \cap \text{co-}\mathsf{NP} = \mathsf{NP} = \text{co-}\mathsf{NP}$
- $\mathsf{P} \neq \mathsf{NP} \cap \text{co-}\mathsf{NP} \neq \mathsf{NP} \neq \text{co-}\mathsf{NP}$

co-NP-complete

VAL – the collection of Boolean expressions that are *valid* is *co-NP-complete*.

Any language L that is the complement of an NP-complete language is *co-NP-complete*.

Any reduction of a language L_1 to L_2 is also a reduction of $\bar{L}_1$–the complement of L_1–to $\bar{L}_2$–the complement of L_2.

There is an easy reduction from the complement of SAT to VAL, namely the map that takes an expression to its negation.

$$\mathsf{VAL} \in \mathsf{P} \Rightarrow \mathsf{P} = \mathsf{NP} = \text{co-}\mathsf{NP}$$

$$\mathsf{VAL} \in \mathsf{NP} \Rightarrow \mathsf{NP} = \text{co-}\mathsf{NP}$$

Prime Numbers

Consider the decision problem PRIME:

Given a number x, is it prime?

This problem is in co-NP.

$$\forall y(y < x \to (y = 1 \lor \neg(\mathrm{div}(y,x))))$$

Note again, the algorithm that checks for all numbers up to $\sqrt{n}$ whether any of them divides n, is not polynomial, as $\sqrt{n}$ is not polynomial in the size of the input string, which is $\log n$.

Primality

Another way of putting this is that Composite is in NP.

Pratt (1976) showed that PRIME is in NP, by exhibiting succinct certificates of primality based on:

> A number $p > 2$ is *prime* if, and only if, there is a number r, $1 < r < p$, such that $r^{p-1} = 1 \bmod p$ and $r^{\frac{p-1}{q}} \neq 1 \bmod p$ for all *prime divisors* q of $p - 1$.

Primality

In 2002, Agrawal, Kayal and Saxena showed that PRIME is in P.

If a is co-prime to p,

$$(x-a)^p \equiv (x^p - a) \pmod{p}$$

if, and only if, p is a prime.

Checking this equivalence would take to long. Instead, the equivalence is checked *modulo* a polynomial $x^r - 1$, for "suitable" r.

The existence of suitable small r relies on deep results in number theory.

Factors

Consider the language Factor

$$\{(x, k) \mid x \text{ has a factor } y \text{ with } 1 < y < k\}$$

Factor $\in$ NP $\cap$ co-NP

Certificate of membership—a factor of x less than k.

Certificate of disqualification—the prime factorisation of x.

Optimisation

The Travelling Salesman Problem was originally conceived of as an optimisation problem

to find a minimum cost tour.

We forced it into the mould of a decision problem – TSP – in order to fit it into our theory of NP-completeness.

Similar arguments can be made about the problems CLIQUE and IND.

This is still reasonable, as we are establishing the *difficulty* of the problems.

A polynomial time solution to the optimisation version would give a polynomial time solution to the decision problem.

Also, a polynomial time solution to the decision problem would allow a polynomial time algorithm for *finding the optimal value*, using binary search, if necessary.

Function Problems

Still, there is something interesting to be said for *function problems* arising from NP problems.

Suppose

$$L = \{x \mid \exists y R(x, y)\}$$

where R is a polynomially-balanced, polynomial time decidable relation.

A *witness function* for L is any function f such that:

- if $x \in L$, then $f(x) = y$ for some y such that $R(x, y)$;
- $f(x) =$ “no” otherwise.

The class FNP is the collection of all witness functions for languages in NP.

FNP and FP

A function which, for any given Boolean expression ϕ, gives a satisfying truth assignment if ϕ is satisfiable, and returns "no" otherwise, is a witness function for SAT.

If any witness function for SAT is computable in polynomial time, then $\mathsf{P} = \mathsf{NP}$.

If $\mathsf{P} = \mathsf{NP}$, then for every language in NP, some witness function is computable in polynomial time, by a binary search algorithm.

$\mathsf{P} = \mathsf{NP}$ if, and only if, $\mathsf{FNP} = \mathsf{FP}$

Under a suitable definition of reduction, the witness functions for SAT are FNP-complete.

Factorisation

The *factorisation* function maps a number n to its prime factorisation:

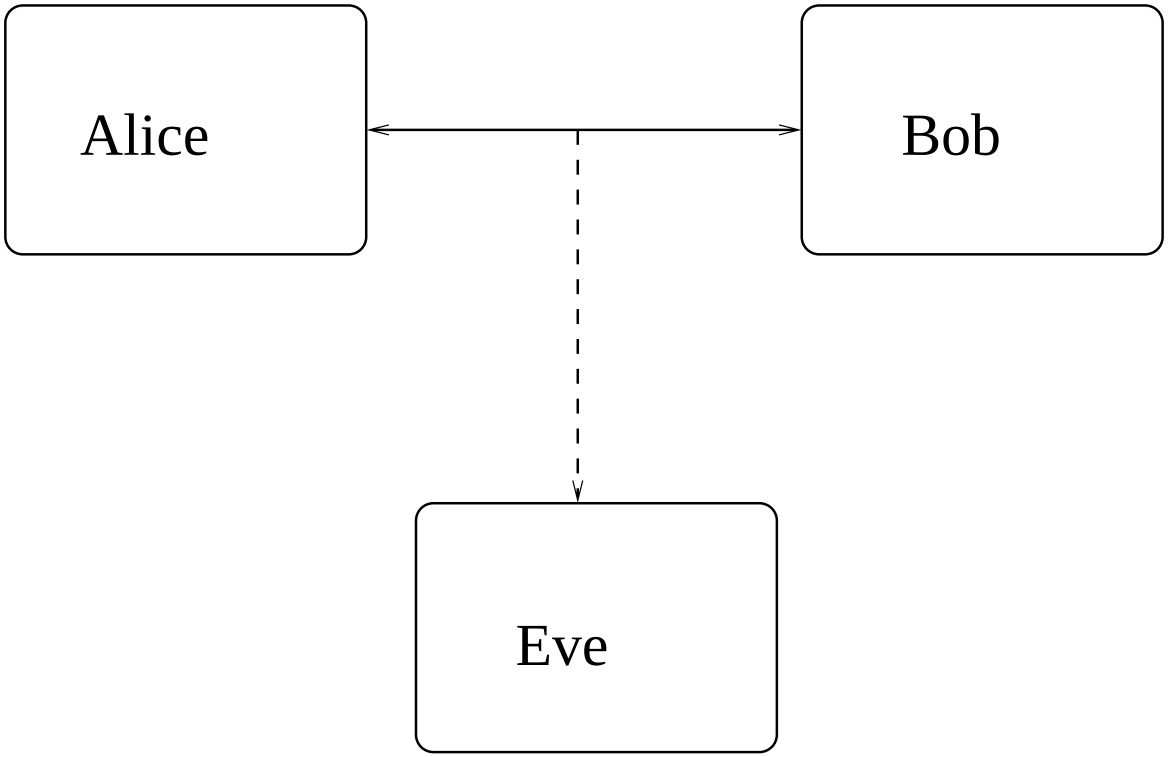
$$2^{k_1} 3^{k_2} \cdots p_m^{k_m}.$$

This function is in FNP.

The corresponding decision problem (for which it is a witness function) is trivial - it is the set of all numbers.

Still, it is not known whether this function can be computed in polynomial time.

Cryptography



Alice wishes to communicate with Bob without Eve eavesdropping.

Private Key

In a private key system, there are two secret keys

e – the encryption key

d – the decryption key

and two functions D and E such that:

for any x,

$$D(E(x, e), d) = x$$

For instance, taking $d = e$ and both D and E as *exclusive or*, we have the *one time pad*:

$$(x \oplus e) \oplus e = x$$

One Time Pad

The one time pad is provably secure, in that the only way Eve can decode a message is by knowing the key.

If the original message x and the encrypted message y are known, then so is the key:

$$e = x \oplus y$$

Public Key

In public key cryptography, the encryption key e is public, and the decryption key d is private.

We still have,

for any x,

$$D(E(x, e), d) = x$$

If E is polynomial time computable (and it must be if communication is not to be painfully slow), then the function that takes $y = E(x, e)$ to x (without knowing d), must be in FNP.

Thus, public key cryptography is not *provably secure* in the way that the one time pad is. It relies on the existence of functions in FNP − FP.

One Way Functions

A function f is called a *one way function* if it satisfies the following conditions:

1. f is one-to-one.
2. for each x, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some k.
3. $f \in \mathsf{FP}$.
4. $f^{-1} \notin \mathsf{FP}$.

We cannot hope to prove the existence of one-way functions without at the same time proving $\mathsf{P} \neq \mathsf{NP}$.

It is strongly believed that the RSA function:

$$f(x, e, p, q) = (x^e \bmod pq, pq, e)$$

is a one-way function.

UP

Though one cannot hope to prove that the RSA function is one-way without separating P and NP, we might hope to make it as secure as a proof of NP-completeness.

Definition

A nondeterministic machine is *unambiguous* if, for any input x, there is at most one accepting computation of the machine.

UP is the class of languages accepted by unambiguous machines in polynomial time.

UP

Equivalently, UP is the class of languages of the form

$$\{x \mid \exists y R(x, y)\}$$

Where R is polynomial time computable, polynomially balanced, *and* for each x, there is *at most one* y such that $R(x, y)$.

UP One-way Functions

We have

$$\mathsf{P} \subseteq \mathsf{UP} \subseteq \mathsf{NP}$$

It seems unlikely that there are any NP-complete problems in UP.

One-way functions exist *if, and only if,* $\mathsf{P} \neq \mathsf{UP}$.

Space Complexity

We've already seen the definition $\mathsf{SPACE}(f(n))$: the languages accepted by a machine which uses $O(f(n))$ tape cells on inputs of length n. *Counting only work space*

$\mathsf{NSPACE}(f(n))$ is the class of languages accepted by a *nondeterministic* Turing machine using at most $f(n)$ work space.

As we are only counting work space, it makes sense to consider bounding functions f that are less than linear.

Classes

$\mathsf{L} = \mathsf{SPACE}(\log n)$

$\mathsf{NL} = \mathsf{NSPACE}(\log n)$

$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{SPACE}(n^k)$

The class of languages decidable in polynomial space.

$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$

Also, define

co-NL – the languages whose complements are in NL.

co-NPSPACE – the languages whose complements are in NPSPACE.

Inclusions

We have the following inclusions:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}$$

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \mathsf{co\text{-}NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{co\text{-}NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \mathsf{co\text{-}NPSPACE}$$

Establishing Inclusions

To establish the known inclusions between the main complexity classes, we prove the following.

- $\mathsf{SPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$;
- $\mathsf{TIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$;
- $\mathsf{NTIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$;

The first two are straightforward from definitions.

The third is an easy simulation.

The last requires some more work.

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

A simple search algorithm solves it:

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.

NL Reachability

We can construct an algorithm to show that the Reachability problem is in NL:

1. write the index of node a in the work space;
2. if i is the index currently written on the work space:
   (a) if $i = b$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   (b) if (i, j) is not an edge, reject, else replace i by j and return to (2).

We can use the $O(n^2)$ algorithm for Reachability to show that:

$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$

for some constant k.

Let M be a nondeterministic machine working in space bounds $f(n)$.

For any input x of length n, there is a constant c (depending on the number of states and alphabet of M) such that the total number of possible configurations of M within space bounds $f(n)$ is bounded by $n \cdot c^{f(n)}$.

> Here, $c^{f(n)}$ represents the number of different possible contents of the work space, and n different head positions on the input.

Configuration Graph

Define the *configuration graph* of M, x to be the graph whose nodes are the possible configurations, and there is an edge from i to j if, and only if, $i \rightarrow_M j$.

Then, M accepts x if, and only if, some accepting configuration is reachable from the starting configuration $(s, \triangleright, x, \triangleright, \varepsilon)$ in the configuration graph of M, x.

Using the $O(n^2)$ algorithm for Reachability, we get that M can be simulated by a deterministic machine operating in time

$$c'(nc^{f(n)})^2 \sim c'c^{2(\log n + f(n))} \sim k^{(\log n + f(n))}$$

In particular, this establishes that NL $\subseteq$ P and NPSPACE $\subseteq$ EXP.

Savitch's Theorem

Further simulation results for nondeterministic space are obtained by other algorithms for Reachability.

We can show that Reachability can be solved by a *deterministic* algorithm in $O((\log n)^2)$ space.

Consider the following recursive algorithm for determining whether there is a path from a to b of length at most n (for n a power of 2):

$O((\log n)^2)$ space Reachability algorithm:

Path(a, b, i)

if $i = 1$ and (a, b) is not an edge reject
else if (a, b) is an edge or $a = b$ accept
else, for each node x, check:

1. is there a path $a - x$ of length $i/2$; and
2. is there a path $x - b$ of length $i/2$?

if such an x is found, then accept, else reject.

The maximum depth of recursion is $\log n$, and the number of bits of information kept at each stage is $3 \log n$.

Savitch's Theorem - 2

The space efficient algorithm for reachability used on the configuration graph of a nondeterministic machine shows:

$$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$$

for $f(n) \geq \log n$.

This yields

$$\mathsf{PSPACE} = \mathsf{NPSPACE} = \mathsf{co\text{-}NPSPACE}.$$

Complementation

A still more clever algorithm for Reachability has been used to show that nondeterministic space classes are closed under complementation:

If $f(n) \geq \log n$, then

$$\mathsf{NSPACE}(f(n)) = \mathsf{co\text{-}NSPACE}(f(n))$$

In particular

$$\mathsf{NL} = \mathsf{co\text{-}NL}.$$

Complexity Classes

We have established the following inclusions among complexity classes:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

Showing that a problem is NP-complete or PSPACE-complete, we often say that we have proved it intractable.

While this is not strictly correct, a proof of completeness for these classes does tell us that the problem is structurally difficult.

Similarly, we say that PSPACE-complete problems are harder than NP-complete ones, even if the running time is not higher.

Provable Intractability

Our aim now is to show that there are languages (*or, equivalently, decision problems*) that we can prove are not in P.

This is done by showing that, for every *reasonable* function f, there is a language that is not in $\mathsf{TIME}(f(n))$.

The proof is based on the diagonal method, as in the proof of the undecidability of the halting problem.

Constructible Functions

A complexity class such as $\mathsf{TIME}(f(n))$ can be very unnatural, if $f(n)$ is.

We restrict our bounding functions $f(n)$ to be proper functions:

Definition

A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and
- 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*.

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

If f and g are constructible functions, then so are $f + g$, $f \cdot g$, 2^f and $f(g)$ (this last, provided that $f(n) > n$).

Using Constructible Functions

Recall $\mathsf{NTIME}(f(n))$ is defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every $x \in L$, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in $\mathsf{NTIME}(f(n))$ is accepted by a machine for which all computations are of length at most $O(f(n))$.

Also, given a Turing machine M and a constructible function f, we can define a machine that simulates M for $f(n)$ steps.

Inclusions

The inclusions we proved between complexity classes:

- $\mathsf{NTIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$

really only work for *constructible* functions f.

The inclusions are established by showing that a deterministic machine can simulate a nondeterministic machine M for $f(n)$ steps.

For this, we have to be able to compute f within the required bounds.

Time Hierarchy Theorem

For any constructible function f, with $f(n) \geq n$, define the f-bounded *halting language* to be:

$$H_f = \{[M], x \mid M \text{ accepts } x \text{ in } f(|x|) \text{ steps}\}$$

where $[M]$ is a description of M in some fixed encoding scheme.

Then, we can show

$H_f \in \mathsf{TIME}(f(n)^3)$ and $H_f \notin \mathsf{TIME}(f(\lfloor n/2 \rfloor))$

Time Hierarchy Theorem

For any constructible function $f(n) \geq n$, $\mathsf{TIME}(f(n))$ is properly contained in $\mathsf{TIME}(f(2n+1)^3)$.

Strong Hierarchy Theorems

For any constructible function $f(n) \geq n$, $\mathsf{TIME}(f(n))$ is properly contained in $\mathsf{TIME}(f(n)(\log f(n)))$.

Space Hierarchy Theorem
For any pair of constructible functions f and g, with $f = O(g)$ and $g \neq O(f)$, there is a language in $\mathsf{SPACE}(g(n))$ that is not in $\mathsf{SPACE}(f(n))$.

Similar results can be established for nondeterministic time and space classes.

Consequences

- For each k, $\mathsf{TIME}(n^k) \neq \mathsf{TIME}(n^{k+1})$.
- $\mathsf{P} \neq \mathsf{EXP}$.
- $\mathsf{L} \neq \mathsf{PSPACE}$.
- Any language that is EXP-complete is not in P.
- There are no problems in P that are complete under linear time reductions.

P-complete Problems

It makes little sense to talk of complete problems for the class P with respect to polynomial time reducibility $\leq_P$.

There are problems that are complete for P with respect to *logarithmic space* reductions $\leq_L$.

One example is CVP—the circuit value problem.

- If CVP $\in$ L then L $=$ P.
- If CVP $\in$ NL then NL $=$ P.