Time and Space

To begin with, we must define what it means for a computation to take time, and to take space. To do this we refer very much to the expected ideas corresponding to what a TM implemented in real-life would do, although this doesn't always carry over in how they go on to be used, and often it's usually better to consider them as abstract resources of a device capable of computation.

Definition 1 (Time and space) *Let $T, S : \mathbb{N} \to \mathbb{N}$ be functions. We say that a TM M runs in time T if for all $x \in \Sigma^*$, M halts within $T(|x|)$ steps. Further, we say that M runs in space S if for all $x \in \Sigma^*$, M accesses at most $S(|x|)$ cells (excluding the input tape).*

Robustness of Turing Machines

There is some initial work to be done here in terms of clarifying the robustness of this definition. Notably, while it was verified in Models of Computation that the TM model is robust to the introduction of multiple tapes, it's unclear how this changes things with respect to time bounds.

Consider a language $L \subseteq \{0,1\}^*$ decidable by a TM M with alphabet Γ in time T, and attempt to construct M' to decide L with alphabet $\{0,1\}$. We can encode any symbol of Γ with $\log|\Gamma|$ bits, and so for every transition $(q,a) \to (q',a',D)$ we introduce a sequence of states after identifying the $\log|\Gamma|$ block of bits so to change the value and move to the correct block. This gives us M' which decides L in time $4\log|\Gamma|T(n)$, space $\log|\Gamma|S(n)$.

Further, consider that L is decidable by a k-tape TM M with alphabet $\{0,1\}$. To simulate this with one tape, we introduce an alphabet $\{0,1\}^k$ (plus any additional necessary constants). Noting that one will never access more than $kT(n)$ cells, we can demonstrate that the single-tape machine can decide L in time $5kT(n)^2$, space $S(n)$. Note that the TM constructed here, where we go back and forth at each stage to perform each transition, is an oblivious TM, defined strictly below:

Definition 2 (Oblivious Turing machine) *An oblivious Turing machine M is one such that for any input $x \in \{0,1\}^*$, the location of the head of M at the ith step is a function of $|x|$ and i.*

Oblivious Turing machines feature again in later proofs, such as that for the Cook-Levin theorem. This is because it is notable that any TM can be simulated obliviously with only a quadratic time loss (and no space loss), and they make analysis far easier.

Formally, the claim that every reasonable model of computation can be simulated by a Turing machine is the Church-Turing thesis, and the additional view that every reasonable model of computation can be simulated by a TM with a polynomial overhead is the strong form of the Church-Turing thesis.

Theorem 1 (Efficient Universal Turing machine) *There exists a TM $\mathcal{U}$ which recognises $\{\langle M, x\rangle : M \text{ accepts } x\}$, and for each M, if M halts in time T, space S, then for any $x \in \{0,1\}^*$, $\mathcal{U}(M,x)$ halts in time $CT(n)\log T(n)$, space $CS(n)$ for C a function of M.*

The proof of this bound is somewhat involved, with the chief problem being the simulation of an arbitrary number of tapes using only a constant number of tapes. One can reach an $O(T(n)^2)$ bound straightforwardly by converting M to a single-tape machine, but to get tighter we need to construct a machine whereby the tape moves under the head, and we keep all filled cells at most $O(\log T(n))$ steps away from the head.

The importance of this claim is in allowing us to construct reprogrammable TMs with minimal loss.

A final important result for TMs is that we can comfortably omit coefficients from our analysis, given by the following theorem:

Theorem 2 (Speed-up theorem) *Let $T : \mathbb{N} \to \mathbb{N}$ be a time function. Let $n \in \mathbb{N}$ be a positive integer. If there is a TM M deciding $L \subseteq \Sigma^*$ in T steps, then there is a TM M' deciding L in $T/n + n + 3$ steps.*

To do this, we write each k symbols in the input as a single one, expanding Γ to Γ^k to do so, compressing an input length n to n/k. This requires $n + n/k + 2$ steps. We then simulate M on the compressed input by updating the cell and its two adjacent cells by the consequences of simulation of k steps, then doing an additional step to simulate the block we end up in. This takes 5 steps (lecture notes say 6???). Thus we simulate k steps in 6 steps, giving total time $6T(n)/k + n/k + n + 2$. Setting $k = 6r$ gives our result.

Going forward to consider the efficiency of language computation, we define the following sets for any $T, S : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{DTIME}(T) = \{L \subseteq \{0,1\}^* : \text{there exists a TM } M \text{ deciding } L \text{ in time } O(T)\}$$
$$\mathsf{DSPACE}(S) = \{L \subseteq \{0,1\}^* : \text{there exists a TM } M \text{ deciding } L \text{ in space } O(S)\}.$$

This allows us to begin to define some basic complexity classes. Note we extend the notation slightly here, but this isn't a problem as long as one recognises that $O(f)$ is a set, not a function.

$$\begin{aligned} \mathsf{P} &= \mathsf{DTIME}\left(n^{O(1)}\right) \\ \mathsf{E} &= \mathsf{DTIME}\left(2^{O(n)}\right) \\ \mathsf{EXP} &= \mathsf{DTIME}\left(2^{n^{O(1)}}\right) \\ \mathsf{L} &= \mathsf{DSPACE}(\log n) \\ \mathsf{PSPACE} &= \mathsf{DSPACE}\left(n^{O(1)}\right) \\ \mathsf{EXPSPACE} &= \mathsf{DSPACE}\left(2^{n^{O(1)}}\right) \end{aligned}$$

As one can very easily construct the complement of decidable languages for Turing machines, these are all closed under complementation (this becomes more difficult at later stages with non-determinism and randomness).

Note that we don't particularly care about $\mathsf{DTIME}(1)$ or $\mathsf{DSPACE}(1)$, as these contain only regular languages (for constant space, exactly the regular languages, and for constant time I don't know).

In order to relate time and space, note that we can never access more than $T(n)$ cells for a computation terminating in $T(n)$ steps, and if a procedure terminates using $S(n)$ space we can never take more than $2^{O(S)}$ steps. Thus we get the relation for a function $f : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{DTIME}(f) \subseteq \mathsf{DSPACE}(f) \subseteq \mathsf{DTIME}(2^{O(f)}),$$

demonstrating that $\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{EXPSPACE}$, $\mathsf{P} \subseteq \mathsf{E} \subseteq \mathsf{EXPSPACE}$.

One quite unintuitive result that's nevertheless important to note, is that it's possible to get some very weird time and space bounds (unlike the ones chosen above, which behave mostly quite nicely).

Theorem 3 (Gap theorem) *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. Then there exists a time bound T, space bound S such that $\mathsf{DTIME}(T) = \mathsf{DTIME}(f(T))$ and $\mathsf{DSPACE}(S) = \mathsf{DSPACE}(f(S))$.*

The proof of this is omitted within this course.

To get nicer looking bounds we use the following definition:

Definition 3 (Bound constructibility) *A time bound $T : \mathbb{N} \to \mathbb{N}$ is time-constructible if there is a TM M that computes $T(n)$ given 1^n as input in time $O(T)$. Similarly, a space bound $S : \mathbb{N} \to \mathbb{N}$ is space-constructible if there is a TM M that computes $S(n)$ given 1^n as input in space $O(S)$.*

Having this then allows us to develop stricter hierarchies, and indeed we get the following results:

Theorem 4 (Time hierarchy) *Let $T, T' : \mathbb{N} \to \mathbb{N}$ such that T, T' are time-constructible and $T \log T = o(T')$. Then $\mathsf{DTIME}(T) \subset \mathsf{DTIME}(T')$.*

Theorem 5 *Let $S, S' : \mathbb{N} \to \mathbb{N}$ such that S, S' are space-constructible and $S = o(S')$. Then $\mathsf{DSPACE}(S) \subset \mathsf{DSPACE}(S')$.*

Both of these use a diagonal contradiction with a universal Turing machine. Essentially we construct D to run $\mathcal{U}$ on input x and the TM M with $\langle M \rangle = x$, then return the opposite result if we halt before $T'(|x|)$ steps, and reject otherwise. This is clearly in $\mathsf{DTIME}(T')$. If we assume that $L(D)$ is decided in T time by some M', run $D(\langle M' \rangle)$, provided $|\langle M' \rangle|$ is large enough we will always terminate (as it's only a logarithmic slowdown), and thus give the wrong answer. In space we have almost exactly the same thing, except we have no 'slowdown'.

Consequently we conclude this section with the following known relations on the above classes:

$$\mathsf{P} \subset \mathsf{E} \subset \mathsf{EXP}$$
$$\mathsf{L} \subset \mathsf{PSPACE} \subset \mathsf{EXPSPACE}$$

Reducibility

One of the fundamental methods of proof in complexity theory is that of reductions. These refer to a general means of bounding the difficulty of a problem in terms of how it may be solved with use of solutions to other problems.

Definition 4 (Karp reductions) *A language $L_1 \subseteq \{0,1\}^*$ is polynomial-time Karp reducible (or "many-to-one reducible") to L_2, denoted $L_1 \le_P L_2$, if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$, $x \in L_1$ iff $f(x) \in L_2$ (alternatively written, $L_1 = f^{-1}(L_2)$).*

Definition 5 (Turing reductions) *A language $L_1 \subseteq \{0,1\}^*$ is polynomial-time Turing reducible (or "Cook reducible") to L_2, denoted $L_1 \le_T L_2$, if L_1 is decidable in polynomial time by a Turing machine equipped with an oracle for L_2 (an additional tape storing inputs to $x \in \{0,1\}^*$ to L_2, that verifies in one step whether $x \in L_2$). We write this as $L_1 \in \mathsf{P}^{L_2}$.*

While Turing reductions are useful, note that they behave strangely when reducing to undecidable languages, on account of being able to modify any oracle results after they have been provided meaning one can take an RE language as the oracle, then invert its result to show that the complement language is 'easier' (when of course it may not be). Thus in the context of decidability we almost always use Karp reductions. In this course that continues to complexity, and so our notions of hardness are usually specified in terms of Karp reductions.

Definition 6 (Log-space reductions) *A language $L_1 \subseteq \{0,1\}^*$ is logspace reducible to a language L_2, denoted $L_1 \le_l L_2$ if there is a function f with its length polynomially bounded in $|x|$ for input $x \in \{0,1\}^*$, and such that both whether index i is in $f(x)$, and $f(x)_i$ itself are computable in logarithmic space, and we have $x \in L_1$ iff $f(x) \in L_2$.*

The above is generally used for considering notions of NL-hardness and completeness.

Definition 7 (NP-hardness) *A language $L \subseteq \{0,1\}^*$ is NP-hard if for all $L' \in \mathsf{NP}$, $L' \le_P L$.*

Definition 8 (NP-completeness) *A language $L \subseteq \{0,1\}^*$ is NP-complete if it is both NP-hard and NP.*

Both of the above definitions are very generalisable to other complexity classes, although usually require different reductions. For example, we have a corresponding notion of -completeness with respect to log-space reductions.

Non-determinism

As seen in Models of Computation, in addition to standard Turing machines we have the notion of non-deterministic Turing machines. It was shown previously that one can simulate a NDTM with a TM, and therefore the notion of recognisability and decidability remains identical under NDTMs as it does TMs. The construction to prove this uses dovetailing in the computation paths however, inducing a heavy computational cost however, and so without further work we get that non-determinism constitutes an essentially exponential improvement in time.

Definition 9 (Non-deterministic time and space) *Let $T, S : \mathbb{N} \to \mathbb{N}$ be functions. We say that an NDTM M decides a language L in time T, space S if for all $x \in \Sigma^*$:*

- *If $x \in L$, there exists an accepting path in M's computation of x of height $\le T(|x|)$, along which every configuration uses $\le S(|x|)$ cells.*
- *If $x \notin L$, every path in M's computation of x rejects within $T(|x|)$ steps, and accesses $\le S(|x|)$ cells.*

In the same way as for standard Turing machines, we define our sets for $T, S : \mathbb{N} \to \mathbb{N}$:

$$\mathsf{NTIME}(T) = \{L \subseteq \{0,1\}^* : \text{ there exists an NDTM } M \text{ deciding } L \text{ in time } O(T)\}$$
$$\mathsf{NSPACE}(S) = \{L \subseteq \{0,1\}^* : \text{ there exists an NDTM } M \text{ deciding } L \text{ in space } O(S)\},$$

giving the following classes

$$\begin{aligned} \mathsf{NP} &= \mathsf{NTIME}\left(n^{O(1)}\right) \\ \mathsf{NE} &= \mathsf{NTIME}\left(2^{O(n)}\right) \\ \mathsf{NEXP} &= \mathsf{NTIME}\left(2^{n^{O(1)}}\right) \\ \mathsf{NL} &= \mathsf{NSPACE}(\log n) \end{aligned}$$

Importantly, as the definition of acceptance and rejection for NDTMs is asymmetric, we do not have necessarily that any of these classes are closed under complementation. Thus we may discuss coNP as a distinct class from NP (although as determined later, $\mathsf{coNL} = \mathsf{NL}$).

Immediately we have that for a time bound T, space bound S, we have $\mathsf{DTIME}(T) \subseteq \mathsf{NTIME}(T)$, $\mathsf{DSPACE}(S) \subseteq \mathsf{NSPACE}(S)$. Further, we see that if T is space-constructible, if $L \in \mathsf{NTIME}(T)$ then we can construct a TM to iterate over the possible computational paths of length $T(|x|)$, giving $L \in \mathsf{DSPACE}(T)$, so $\mathsf{NTIME}(T) \subseteq \mathsf{DSPACE}(T)$.

Theorem 6 (Alternative definition of NP) *For any $L \subseteq \{0,1\}^*$, $L \in \mathsf{NP}$ iff there is a language $R \in \mathsf{P}$ and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that*

$$L = \left\{x \in \{0,1\}^* : \exists w \in \{0,1\}^{\le p(|x|)} . \langle w, x\rangle \in R\right\}$$

More intuitively: this theorem gives that NP is the set of languages stating an existence problem that is efficiently verifiable once a certificate is provided.

In the forward direction, we have when $L \in \mathsf{NP}$ that there is an NDTM N such that for each $x \in L$ there is an accepting polynomial-length computation of x via N. Thus we construct

$$R = \{\langle w, x\rangle : w \text{ is an accepting computation of } x \text{ via } N\},$$

and by N a polynomial-time NDTM for L we immediately have the desired equality. The backwards direction then follows immediately by just constructing an NDTM N to guess a w then run R.

With this alternative definition we can see some of the importance of the $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$ question. If $\mathsf{P} = \mathsf{NP}$, then it is as easy to verify a certificate to a problem instance's truth as it is to determine the problem instance's truth by any other means. Further, this allows us to construct certificates in polynomial time, with only a small bit of extra work.

Additionally, we have a clearer idea from this as to the construction of coNP. It is precisely the set of $x \in \{0,1\}^*$ such that for any $w \in \{0,1\}^{\le p(|x|)}$, $\langle w, x\rangle \notin R$, so a universally quantified statement.

Theorem 7 (Non-deterministic time hierarchy theorem) *Let T, T' be time-constructible functions with $T(n+1) = o(T'(n))$, then*

$$\mathsf{NTIME}(T) \subset \mathsf{NTIME}(T')$$

Cook-Levin theorem

A very important result for the purpose of considering NP-hardness is the Cook-Levin theorem, which allows us to characterise NP by SAT:

Theorem 8 (Cook-Levin) *Both SAT and 3SAT and NP-complete.*

Immediately we can construct an NDTM for SAT by guessing a variable assignment, which will be of length polynomial in the input, and then validating whether it satisfies the formula which can also be done in polynomial time.

Given an arbitrary $L \in \mathsf{NP}$, we have a corresponding M deciding some $L' \in \mathsf{P}$ where there is a $u \in \{0,1\}^{p(|x|)}$ such that $M(x,u) = 1$ iff $x \in L$. We want to construct φ_x with $\varphi_x(u) = M(x,u)$ for all u, so φ_x is satisfiable iff $x \in L$. Note that this only works if we can get $|\varphi_x|$ polynomial.

To do this, the goal is to be able to specify each stage of the execution of M by a snapshot that can be encoded in a binary string, then construct a formula which verifies whether a sequence of snapshots is a valid accepting computation.

To do this we need to assume that M is a 2-tape oblivious TM (i.e. the head position at step i is dependent only on the input size and i), which is fine as this only corresponds to a quadratic increase in complexity, and then specify each stage by $(a, b, q) \in \Gamma \times \Gamma \times Q$. To determine if a snapshot z_i is valid, for y the input, at each i we only need to consider snapshots z_{i-1}, $y_{\text{inputpos}(i)}$, and $z_{\text{prev}(i)}$, where $\text{prev}(i)$ is the last time the current cell was accessed. This tells us what state we were in, as well as the current cell values, allowing us to verify the computation.

From here we then just need to verify that the inputs are valid, the first step is valid, and the sequences are valid, which produces a CNF polynomial in $|x|$, as the computation is polynomial in $|x|$.

From here reduction to 3SAT is relatively straightforward.

Space Complexity

Non-determinism opens up a wealth of routes through which to analyse space complexity, by introducing a construction that while likely stronger than a deterministic one in time efficiency, nevertheless doesn't immediately appear to give significant space advantages (and indeed, the bounds we have are a lot tighter here).

Firstly, we just remark briefly on the notion of a *configuration graph*. For any NDTM M, input x, we construct this graph $G_{M,x}$ with each configuration represented as a vertex, and each edge corresponding to a transition between two configurations. Further, by clearing the state before accepting as a form of normalisation, we maintain that there is only ever one accepting state. Thus $M(x) = 1$ iff there is a path from the start vertex to the accept vertex in $G_{M,x}$.

It is clear that for computational purposes that if M has space-complexity S, the graph can be constructed in $2^{O(S)}$ time, so we get that $\mathsf{NSPACE}(S) \subseteq \mathsf{DTIME}(2^{O(S)})$.

One of the most important space complexity classes is that of PSPACE. Additionally, we refer to the notion of PSPACE-hardness and completeness, with respect to polynomial-time Karp reductions. To consider it further, take the language of valid fully quantified boolean formulas. An instance of the language takes the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n . \varphi(x_1, x_2, \dots, x_n).$$

We can solve this recursively by having $A(\exists x\, \psi) = A(\psi[x/0]) \vee A(\psi[x/1])$, $A(\forall x\, \psi) = A(\psi[x/0]) \wedge A(\psi[x/1])$. Note however that we can avoid doubling the space requirement each time by having the first call be processed, then returning the single required bit and making the second call, using their independence. This gives us polynomial space.

Further, we show PSPACE-hardness by taking a language decided by M in space S, and then noting we can construct the configuration graph of M on x in space $O(S)$, before then constructing a quantified boolean formula that is true iff there is a path in the configuration graph from the start vertex to the accepting vertex. This can be done in $O(S^2)$ space (essentially we just need to use quantifiers to avoid doubling the formula size for reducing the path by a factor of 2).

In fact from this we get the stronger statement that the language is $\mathsf{NPSPACE}$-hard, and in doing so one proves that $\mathsf{PSPACE} = \mathsf{NPSPACE}$. Further, this leads to Savitch's theorem, by using reachability within a configuration graph to get only a polynomial space deficit when losing non-determinism:

TQBF (the language of true quantified boolean formulas) in fact gives a good description of what PSPACE represents. Noting that we can represent the statement of the existence of a winning strategy in a turn-based two-player perfect information game as a quantified boolean formula, PSPACE is the set of problems equivalent to the existence of a winning strategy to some turn-based two-player game.

Theorem 9 (Savitch's theorem) *For any space-constructible $S : \mathbb{N} \to \mathbb{N}$ with $S(n) \ge \log n$, $\mathsf{NSPACE}(S) \subseteq \mathsf{DSPACE}(S^2)$.*

Concretely, with M an NDTM using $O(S)$ space, for an input x of length n take $G_{M,x}$ the configuration graph with $2^{O(S(n))}$ vertices, and determine $\mathrm{REACH}(C_{\text{start}}, C_{\text{acc}}, 2^{O(S(n))})$ (need to calculate exact coefficients, but this is straightforward enough). To do this, note $\mathrm{REACH}(u, v, 2^i) = \bigvee_z (\mathrm{REACH}(u, z, 2^{i-1}) \wedge \mathrm{REACH}(z, v, 2^{i-1}))$, and enumerating over all z takes only $O(S)$ space. This allows us to write $s_i = s_{i-1} + O(S)$, resulting in $O(S^2)$ complexity.

Considering now the smaller class of NL, note that PATH, the problem of deciding whether one can reach t from s in the graph G, can be determined in logarithmic space non-deterministically by at any point storing the current vertex in $\log n$ space, and accepting if the current vertex ever becomes t by only ever moving over edges.

Then note that if $L \in \mathsf{NL}$, taking the configuration graph for its NDTM M on x as $G_{M,x}$, $|G_{M,x}|$ is polynomial in n, and we can determine whether there is an edge from u to v in $G_{M,x}$ in $O(\log n)$ space to store both and check the transition function of M. Thus we have a logspace reduction to PATH so it is NL-complete.

This is in fact quite useful, giving us a similar alternative definition as for NP:

Theorem 10 (Alternative definition of NL) *For any $L \subseteq \{0,1\}^*$, $L \in \mathsf{NL}$ iff there is a polynomial $p : \mathbb{N} \to \mathbb{N}$, and a language $R \in L$ for which there is a logspace TM M for R where to compute $\langle u, x\rangle$, each element of u is accessed only once, and*

$$L = \left\{x \in \{0,1\}^* : \exists u \in \{0,1\}^{\le p(|x|)} . \langle u, x\rangle \in R\right\}.$$

While this seems slightly insane, note that we can consider u as, for example, a description of a path in a graph of polynomially many vertices. Having this definition we can then see that $\overline{\text{PATH}} \in \mathsf{NL}$, as we can construct a polynomial certificate to demonstrate that searching from the start vertex does not retrieve the end vertex. Consequently we get the following theorem:

Theorem 11 (Immerman-Szelepcsényi Theorem) $\mathsf{NL} = \mathsf{coNL}$.

Furthermore, in fact we have that for space constructible $S(n) > \log n$, $\mathsf{coNSPACE}(S) = \mathsf{NSPACE}(S)$. This is by generalising the proof for the above, saying that if we have that there are N configurations reachable by a TM M from the starting configuration, we consider each configuration in lexicographical order, guessing a path from the start until it doesn't reach the intended configuration. Each time we find a path to the desired configuration, we increment the number of found reachable configurations, until this reaches N. If the configuration is the accepting one, and we find a path to it, immediately we must reject. Otherwise if we can guess N configurations reachable and have them not include the accepting one, we have that M must reject, so we can accept. Note that each guessing process requires only $S(n)$ cells.

By a similar process of doing very many guesses of graph traversals then eventually rejecting if the correct targets were not reached, we can calculate N in space $S(n)$.

Ultimately from the results as given above we can get the understanding:

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

Polynomial Hierarchies

With NP and coNP we can capture a general class of problems that appear very difficult to solve efficiently. This is not the most general notion however, and it is possible to extend this to a wider hierarchy of classes. With a class $\mathcal{C}$, we define the classes $\exists^{\mathsf{P}}\mathcal{C}$ and $\forall^{\mathsf{P}}\mathcal{C}$ as follows:

$$\exists^{\mathsf{P}}\mathcal{C} := \left\{\{x \in \{0,1\}^* : \exists w \in \{0,1\}^{\le p(|x|)} . \langle x, w\rangle \in R\} : p \in \mathbb{N}[x], R \in \mathcal{C}\right\}$$
$$\forall^{\mathsf{P}}\mathcal{C} := \left\{\{x \in \{0,1\}^* : \forall w \in \{0,1\}^{\le p(|x|)} . \langle x, w\rangle \in R\} : p \in \mathbb{N}[x], R \in \mathcal{C}\right\}.$$

Further, we have that

$$\begin{aligned} \mathsf{co}\,\exists^{\mathsf{P}}\mathcal{C} &= \{\overline{L} : L \in \exists^{\mathsf{P}}\mathcal{C}\} \\ &= \left\{\{x \in \{0,1\}^* : \forall w \in \{0,1\}^{\le p(|x|)} . \langle x, w\rangle \notin R\} : p \in \mathbb{N}[x], R \in \mathcal{C}\right\} \\ &= \left\{\{x \in \{0,1\}^* : \forall w \in \{0,1\}^{\le p(|x|)} . \langle x, w\rangle \in R'\} : p \in \mathbb{N}[x], R' \in \mathsf{co}\,\mathcal{C}\right\} \\ &= \forall^{\mathsf{P}}\mathsf{co}\,\mathcal{C} \end{aligned}$$

allowing us to write, for example, $\mathsf{coNP} = \forall^{\mathsf{P}}\mathsf{coP} = \forall^{\mathsf{P}}\mathsf{P}$. From here we write inductively:

$$\begin{aligned} \Sigma_0^p &= \mathsf{P} \\ \Pi_i^p &= \mathsf{co}\,\Sigma_i^p \\ \Sigma_{i+1}^p &= \exists^{\mathsf{P}}\Pi_i^p, \end{aligned}$$

and define $\mathsf{PH} = \bigcup_i \Sigma_i^p$. Note we can also, by the above, write $\Pi_{i+1}^p = \forall^{\mathsf{P}}\Sigma_i^p$. Note that if we find that $\Sigma_i^p = \Sigma_{i+1}^p$, then $\mathsf{PH} = \Sigma_i^p$ (it collapses).

More intuitively, given a language $L \in \Sigma_{2i}^p$, we know that it solves a problem of the form $\exists x_1 \forall x_2 \dots \exists x_{2i-1} \forall x_{2i} \langle x, x_1, x_2, \dots, x_{2i}\rangle \in R$ for $R \in \mathsf{P}$, each x_j of polynomial size in $|x|$.

An alternative means of viewing the hierarchy is via oracle machines. Writing $\Sigma_i\mathrm{SAT}$ as the language of true boolean formulas quantified alternatively i times (beginning with $\exists$, with each variable a boolean vector), note that $\Sigma_i\mathrm{SAT}$ is Σ_i^p-complete, applying an almost identical proof as for the Cook-Levin theorem, appending an $\exists z$ to the front that allows $\varphi_x(z, u)$ to be true iff z simulates an accepting computation of $\langle x, u\rangle$ by a TM for R.

Using this we can then write without loss of generality $\mathcal{C}^{\Sigma_i^p}$ for $\mathcal{C}^{\Sigma_i\mathrm{SAT}}$. Further, we then get that $\Sigma_i^p = \mathsf{NP}^{\Sigma_{i-1}^p}$ as an alternative definition.

Circuits and Non-uniformity

One might imagine that for certain languages, while the language itself might be particularly difficult, it behaves rather differently for fixed input sizes. This is known as nonuniform computation, where, unlike for TMs, we deal with inputs of different classes with possibly entirely different constructions.

Definition 10 (Boolean circuits) *An n-input, single-output Boolean circuit is a directed acyclic graph with n sources (vertices with no incoming edges) and one sink (vertex with no outgoing edges). Any nonsource vertex is a gate, labelled by $\neg$, $\wedge$, or $\vee$.*

We refer to the fan-in of a gate as the number of incoming edges (often restricted to 2 for $\wedge$ and $\vee$, 1 for $\neg$), and the fan-out is unrestricted (unlike for normal Boolean formulas).

Note that we can compute the output of a Boolean circuit in linear time by computing its vertices one-by-one.

Definition 11 (Circuit families) *Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A T-size circuit family is a sequence (C_n) of Boolean circuits, where C_n has n inputs and a single output, with $|C_n| \le T(n)$ for every n.*

Using this, we consider a new class:

$$\mathsf{SIZE}(T) = \left\{\{x \in \{0,1\}^* : C_{|x|}(x) = 1\} : (C_n) \text{ is an } O(T)\text{-size circuit family}\right\}.$$

This gives us the immediate definition

$$\mathsf{P/poly} = \left(n^{O(1)}\right),$$

with some interesting consequences. Firstly, note that $\mathsf{P} \subset \mathsf{P/poly}$, because not only can we construct a circuit for any language by considering its computation by an oblivious TM, and then constructing a circuit based on snapshots (giving a polynomial-size circuit family), but we can also decide far more difficult languages. Indeed, consider the tally languages, which we can decide all of immediately using a linear-size circuit family. To distinguish slightly here, we use the following definition:

Definition 12 (P-uniform circuit families) *A circuit family (C_n) is P-uniform if there is a polynomial-time TM that on input 1^n outputs the description of the circuit C_n.*

Note that restricting circuits to be P-uniform collapses $\mathsf{P/poly}$ to P (in fact, the set of languages decided by P-uniform circuit families is precisely P). Another similar notion is that of logspace-uniform families, where we require that there is a logspace-computable function mapping 1^n to a description of C_n.

We would like to be able to relate circuits to Turing machines in some more concrete way, and indeed we can do by adding an additional notion of 'advice' to a Turing machine. That is, for an input x of size n, the TM takes as input both the x itself as normal, as well as an advice string α_n. We can then analyse the complexity using the following definition:

$$\mathsf{DTIME}(T)/\alpha = \{L \subseteq \{0,1\}^* : L \text{ decidable in time } T(n) \text{ by a TM with } \alpha(n) \text{ bits of advice}\}.$$

This yields the characterisation

$$\mathsf{P/poly} = \mathsf{DTIME}\left(n^{O(1)}\right)/n^{O(1)},$$

formed by providing the circuit description as advice, for the forward direction. For the reverse direction, we use the existing construction that was used to show $\mathsf{P} \subseteq \mathsf{P/poly}$, hardwiring in the advice to the circuit so to ensure it functions with polynomial size.

Even despite $\mathsf{P/poly}$ appearing particularly powerful in certain instances, nevertheless it is unlikely that $\mathsf{NP} \subseteq \mathsf{P/poly}$. By the Karp-Lipton theorem we have that if this were the case, $\mathsf{PH} = \Sigma_2^p$. To show this we need to demonstrate that $\Sigma_2^p = \Sigma_3^p$ (or, alternatively, $\Pi_2^p \subseteq \Sigma_2^p$). **Continue proof**.

Theorem 12 *For every $n > 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit C of size $2^n/10n$.*

This can be seen from a counting argument. **How many circuits are there of size S?**

Theorem 13 (Nonuniform hierarchy theorem) *For $T, T' : \mathbb{N} \to \mathbb{N}$ with $2^n/n > T'(n) > 10T(n) > n$,*

$$\mathsf{SIZE}(T) \subset \mathsf{SIZE}(T').$$

We can do this via a similar counting argument.

Finally, we consider the additional complexity classes:

$$\mathsf{NC}^d = \{L \subseteq \{0,1\}^* : L \text{ decided by a family } (C_n) \text{ of polynomial size, depth } O(\log^d n)\}$$
$$\mathsf{NC} = \bigcup_{d \ge 0} \mathsf{NC}^d.$$

We also define AC^i and AC in a similar way, with the exception that we allow circuits with unbounded fan-in. Note NC^0 is extremely limited by this definition, only being able to use a bounded number of bits.

The above complexity classes essentially represent whether there exist efficient algorithms for problems. Thus it is of major concern whether $\mathsf{P} = \mathsf{NC}$.

Randomness

Randomness can often be useful when writing programs, for a few reasons. Firstly, it is useful to break symmetries, because in certain instances it can become impossible to make a choice if all is equal between two objects. Secondly, it can be useful for hiding or obfuscating information. Thirdly, it can be useful for making computation more efficient.

As an example, while it takes deterministically superquadratic time to verify that $AB = C$, one way to randomly verify this is to select a random vector $x \in \{0,1\}^n$, then compute $ABx = A(Bx)$ in quadratic time, and Cx in quadratic time, then if the results are equal we have that $AB = C$ with probability $1/2$. Iterate this k times and we get probability $1 - 1/2^k$.

Alternatively we might want to solve the connectivity problem in an undirected graph. To do this given trying to check if s and t are in the same connected component, take a random walk from s and terminate either after a fixed number of steps to reject, or after the walk reaches t.

A probabilistic turing machine (PTM) is an NTM where there are at most two choices at any stage in the computation. Given a computation $C_0, C_1, \dots, C_t$ where for all $i \in \{1, \dots, t-1\}$, $C_i \vdash C_{i+1}$, we say that C_t has probability $1/2^k$ in the computation if there are exactly k probabilistic transitions. We say that a PTM accepts x with probability p in the expected way (sum of probabilities of accepting over possible computations).

Definition 13 *Given PTM M, time function $t : \mathbb{N} \to \mathbb{N}$, error bound $e : \mathbb{N} \to [0,1]$, we say that M decides L in time t with bounded error ε if M always halts in time $t(|x|)$ and $x \in L$ implies that M accepts x with probability $\ge 1 - \varepsilon(|x|)$, and if $x \notin L$ then M accepts x with probability $\le \varepsilon(|x|)$.*

We say that an algorithm has one-sided error where $x \in L$ means that M accepts with probability $\ge 1 - \varepsilon(|x|)$, and if $x \notin L$ means M accepts with probability 0, and it has zero error if it always outputs the correct answer when it halts, but halts in time $t(|x|)$ with probability $\ge 1 - \varepsilon(|x|)$.

Definition 14

$$\begin{aligned} \mathsf{BPP} &= \{L : \exists k \text{ s.t. } L \text{ is decided by some PTM with error } 1/3 \text{ in } O(n^k)\} \\ \mathsf{RP} &= \{L : \exists k \text{ s.t. } L \text{ is decided by some PTM with one-sided error } 1/3 \text{ in } O(n^k)\} \\ \mathsf{ZPP} &= \{L : \exists k \text{ s.t. } L \text{ is decided by some PTM with zero error } 1/3 \text{ in } O(n^k)\} \end{aligned}$$

Between these classes we have the following relations:

$$\begin{aligned} \mathsf{ZPP} &= \mathsf{RP} \cap \mathsf{coRP} \\ \mathsf{RP} &\subseteq \mathsf{BPP} \\ \mathsf{coRP} &\subseteq \mathsf{BPP}. \end{aligned}$$

The intuition one should have for the above is that ZPP and RP are analogous to P and NP. Whereas we don't even have a proof that $\mathsf{BPP} \subseteq \mathsf{NEXP}$ currently (despite believing that $\mathsf{BPP} = \mathsf{P}$), it is clearer that ZPP and RP lie firmly in NP.

Theorem 14 (Alternative definition of BPP) *A language $L \subseteq \{0,1\}^*$ is in BPP iff there is a polynomial-time TM M, polynomial p such that*

$$L = \left\{x \in \{0,1\}^* : \mathbb{P}(M(x,R) = 1) \ge 2/3,\ R \sim U(\{0,1\}^{p(|x|)})\right\}.$$

Theorem 15 (Error reduction for BPP) *Let $L \subseteq \{0,1\}^*$ be a language and suppose there is a polynomial-time PTM M such that for $x \in \{0,1\}^*$, $\mathbb{P}(M(x) = \mathbb{1}(x \in L)) \ge 1/2 + |x|^{-c}$. Then for every constant $d > 0$ there is a PTM M' such that $\mathbb{P}(M'(x) = \mathbb{1}(x \in L)) \ge 1 - 2^{-|x|^d}$.*

To see this, for each $x \in \{0,1\}^*$ run M for $k = 8|x|^{2c+d}$ times, and take the majority answer.

For BPP we have the following two results:

$$\mathsf{BPP} \subseteq \mathsf{P/poly}$$
$$\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

The former follows by using Chernoff bounds to obtain error reductions that via counting give that there is a randomly generated string for each n that gives exactly the intended language, allowing for a hardwiring in building a polynomial circuit. The latter follows by considering the size of the set of random strings for which an input is accepted, then using the probabilistic method on discrete distributions to turn this into an existential quantification.

One of the major open questions regarding the ordering is whether $\mathsf{BPP} = \mathsf{P}$, as well as whether $\mathsf{BPP} \subseteq \mathsf{SUBEXP}$ (the set of languages in $O(2^{n^\varepsilon})$ for all $\varepsilon > 0$).

Theorem 16 *Let $p : \Sigma^* \to [0,1]$ be a function. M is a PTM running in polynomial time iff there is a polynomial time bounded deterministic TM N and a constant k such that for any $x \in \Sigma^*$, if M accepts x with probability $p(x)$ then the probability that $N(x, \pi)$ accepts with $\pi \sim U(\{0,1\}^{n^k})$ is $p(x)$.*

The proof by means of constructing N is relatively straightforward. π just decides the entire sequence of transitions, and it is clear how to construct N polynomial to do this.

Theorem 17 *Suppose E does not have circuits of size $2^{\varepsilon n}$ for almost all n with $\varepsilon > 0$. Then $\mathsf{BPP} = \mathsf{P}$.*

Theorem 18 *The following are equivalent:*

- *$L \in \mathsf{RP}$*
- *For all $k \ge 0$ there is a PTM with one-sided error $\le 1/2^{n^k}$ deciding L.*
- *There is a $k \ge 0$ such that there is a PTM with one-sided error $\le 1 - 1/n^k$ deciding L.*

Lemma 19 *Let $x_1, \dots, x_m$ be IID random variables in $[0,1]$. Suppose $\mu = \mathbb{E}[\sum x_i]$. Then $\mathbb{P}(\sum x_i > \mu + \varepsilon) \le 2e^{-\frac{t^2}{2m}}$.*