COMPUTABILITY AND COMPLEXITY 24
SPACE COMPLEXITY

AMIR YEHUDAYOFF
DIKU

Remark. *Computational complexity studies the resources that are needed to achieve computational tasks. On a high-level, computational devices have costs (like time, memory size, energy, randomness, training data, etc.), and computational tasks have complexities (the minimum cost that is needed to achieve it). We now move to focussing on space.*

1. Space

Example 1. *What is $\sqrt{2}$? Can we write its digits? What is π?*

Example 2. *Sometimes programs run out of memory.*

Example 3. *What is the space complexity of "palinodromes"?*

We will focus on space in the TM model.

Definition 4. *The space that a TM M uses on $x \in \{0,1\}^*$ is the number of locations on the (working) tape M visits during the run on x. It is denote by $\mathsf{SPACE}(M,x)$, and it could be infinite. For $n \in \mathbb{N}$, define*

$$\mathsf{SPACE}(M,n) = \max_{x \in \{0,1\}^n} \mathsf{SPACE}(M,x).$$

Remark. *Memory locations can be used more than once for free. Only the first time a position on the tape is visited is counted. This is what distinguishes space from time.*

Remark. *We do not count the access to the input tape because we want to consider algorithms with sublinear space.*

Definition 5. *Let $S : \mathbb{N} \to \mathbb{N}$. We say that $L \subseteq \{0,1\}^*$ is in $\mathsf{SPACE}(S(n))$ if there is a TM M that decides L so that*

$$\mathsf{SPACE}(M,n) = O(S(n)).$$

Recall that we introduced a powerful computational resources: non determinism ("the power to guess correctly" or $\exists$).

Definition 6. *The space cost of a NTM M on x is the maximum space in the run of M on x over all (non-net.) choices. We similarly get $\mathsf{NSPACE}(S(n))$.*

Remark. *We only deal with $S(n)$ that are "space-constructible". That is, there is a TM M that given 1^n as input, computes $S(n)$ using space $O(S(n))$. All functions $S(n)$ from now on are assumed to be such (without explicitly stating it). All reasonable functions are such (but not all functions are such). We shall ignore this issue from now on.*

How is space related to time?

Theorem 7. *For every $S : \mathbb{N} \to \mathbb{N}$ so that $S(n) \geq \log n$,*

$$\mathsf{DTIME}(S(n)) \subseteq \mathsf{SPACE}(S(n)) \subseteq \mathsf{NSPACE}(S(n)) \subseteq \mathsf{DTIME}(2^{O(S(n))}).$$

The first inclusion holds because in times T, a TM can visit at most T locations. The second holds because non-determinism just adds power. The third shall follow from the following discussion.

1.1. **Configuration graphs.**

We can represent computations by graphs.

Definition 8. *The configuration graph of M on x is a directed graph $G_{M,x}$ that is defined as follows. The vertices are the "configuration" of the run (i.e., a vertex v fully encodes a possible state of the computation). There is an edge (v, u) in the graph if v is a vertex, u is a vertex and M moves from state v to state u in one time step.*

Remark. *If M is deterministic, all out-degrees are one.*

Remark. *If M is non-deterministic, all out-degrees are at most two.*

Remark. *Loops in the graph correspond to computations that do not terminate.*

Claim 9. *M accepts x iff there is a path from the initial state to an accepting state in $G_{M,x}$.*

Claim 10. *Each vertex in the graph can be described using $O(\mathsf{SPACE}(M, x))$ bits.*

Claim 11. *The number of vertices in the graph is at most $2^{O(\mathsf{SPACE}(M,x))}$.*

As was explained in the proof of the Cook-Levin theorem, the edges can be described effectively:

Claim 12. *For every TM or NTM M and $x \in \{0, 1\}^n$, there is a CNF formula $\varphi_{M,x}$ so that if v, u are two bit strings (encoding potential vertices in the graph) then $\varphi_{M,x}(v, u) = 1$ iff both v, u are vertices and (v, u) is an edge in $G_{M,x}$. In addition, the size of $\varphi_{M,x}$ is at most $O(\mathsf{SPACE}(M, x) + \log n)$.*

Remark. *Intuitively, the formula $\varphi_{M,x}$ checks that v encodes a proper state, u encodes a proper state, and the transition $v \to u$ agrees with M. The formula has the claimed size because of the local behavior of TMs.*

Proof of last part of Theorem 7. Given a NTM M and input x, we need to decide in time $2^{O(\mathsf{SPACE}(M,x))}$ if M accepts x or not. In other words, we need to decide if there is a path from the initial state to an accepting state in $G_{M,x}$. This can be done, e.g., using BSF on the graph $G_{M,x}$. □

Remark. *This is a powerful idea. We can think of a computation as a walk in a graph that is implicitly descried by M and x. Reachability problem are thus deeply related to computation.*

Remark. *We see that basic algorithms, like BFS, help to understand things concerning general TMs.*

1.2. **Important classes.** There are a few natural choices for classes to focus on:

Definition 13.

$$\mathsf{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathsf{SPACE}(n^k)$$
$$\mathsf{NPSPACE} = \bigcup_{k \in \mathbb{N}} \mathsf{NSPACE}(n^k)$$

Remark. $\mathsf{P} \subseteq \mathsf{PSPACE}$.

Remark. $\mathsf{NP} \subseteq \mathsf{PSPACE}$ *because e.g. we can check all assignments for a formula in polynomial space (and exponential time).*

Remark. *We do not know if* $\mathsf{P} = \mathsf{PSPACE}$ *or not. But if* $\mathsf{P} = \mathsf{PSPACE}$ *then* $\mathsf{P} = \mathsf{NP}$.

Remark. *We can represent* 2^n *objects in space* n*. So, allowing algorithms to run in space* n *may allow them to check* 2^n *options. This is often too costly. The following two classes are the space analogs of* P *and* NP.

Definition 14.

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

Remark (recap)**.** *Computations can be thought of as walks on huge digraphs. The edges of the graph are, however, easily described. A computational is "accepting" if two vertices are connected.*

1.3. PSPACE.

Remark. *One of the first things to do in order to understand a complexity class is find a "complete" problem for it.*

Remark. *A basic idea in this theory is that of "reductions". Intuitively,* $A \leq B$ *if "problem* A *is easier than* B*". We have seen* $A \leq_p B$ *where the reduction* $f : \{0,1\}^* \to \{0,1\}^*$ *so that* $x \in A$ *iff* $f(x) \in B$ *is poly-time. Later, we shall see other types of reductions. The reductions should fit the scenario we are thinking about.*

Definition 15. *A language* $C \subseteq \{0,1\}^*$ *is* PSPACE*-complete if* $C \in \mathsf{PSPACE}$ *and* $L \leq_p C$ *for every* $L \in \mathsf{PSPACE}$.

Remark. *There is a general way to construct complete problems. For* PSPACE*, it is something like*

$$\{\langle M, x, 1^s \rangle : M(x) = 1, \mathsf{SPACE}(M, x) \leq s\}.$$

But we typically want to identify more natural complete problems.

The complete problem we shall talk about is "totally quantified boolean formulas".

Definition 16. *A boolean formula over the variables* $x_1, \ldots, x_n$ *is an expression of the form*

$$(x_1 \wedge x_2) \vee (x_1 \vee (\neg x_2)).$$

It can be thought of as a tree (illustrate the example). Formulas can be inductively defined as follows. Each of the expression $x_1, \ldots, x_n$ *and* $0, 1$ *are formulas. If* f *is* *a formula then $(\neg f)$ is a formula. If f_1, f_2 are formulas then $(f_1 \wedge f_2)$ and $(f_1 \vee f_2)$ are formulas.*

Remark. *Formulas can be thought of as trees.*

Remark. *A formula computes a boolean function $\{0,1\}^n \to \{0,1\}$ in the obvious way.*

Remark. *Every boolean function $\{0,1\}^n \to \{0,1\}$ can be computed by some formula (in fact, by many formulas).*

Remark. *Formulas can be thought of as computational devices. As such, a formula has a cost. The size of f is inductively defined: the size of the base case is 1, and $size(\neg f) = size(f)$ and $size(f_1 * f_2) = size(f_1) + size(f_2)$ for $* \in \{\wedge, \vee\}$.*

Remark. *If f has size s then it can be described by $O(s)$ bits.*

Remark. *Formulas lead to a different model of computational complexity theory; more on this later on. In it, the devices are formulas and the complexity of a function is the size of the minimum formula computing it.*

Remark. *A totally quantified boolean formula is an expression of the form*

$$\forall x_1 \in \{0,1\} \exists x_2 \in \{0,1\} \ (x_1 \vee x_2) \wedge x_1.$$

The quantifiers Q can be either $\forall$ and $\exists$. A totally quantified boolean formula (TQBF) is an expression of the form

$$E = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \ \varphi$$

where φ is boolean formula over $x_1, \dots, x_n$. It is understood that each x_i takes values in $\{0,1\}$. Every TQBF has a truth value in $\{0,1\}$. For example, the truth value of the above expression is 1.

Example 17.

$$\phi(x_1, \dots, x_n) \in \mathsf{SAT} \iff 1 = \exists x_1 \dots \exists x_n \phi$$

Remark. *TQBF capture properties that are deeply related to game theory. Let us consider chess for example. The expression*

$$\exists w_1 \forall b_1 \exists w_2 \forall b_2 \dots \exists w_{100} \ \textit{white wins}$$

says that there is a first move for white, so that for every move of black, there is a move of white, ... so that white wins in 100 moves.

Definition 18.

$$\mathsf{TQBF} = \{\langle E \rangle : E \textit{ is a true TQBF}\}.$$

Theorem 19. TQBF *is* PSPACE*-complete.*

Proof. We start by sketching why TQBF is in PSPACE. This is not entirely trivial because we need to reuse space (there are exponentially many things to check). The algorithm A is recursive. The base case is boolean formulas over $0, 1$. The algorithm A output the truth value of the input formula. The inductive step is as follows. If the input to A is

$$\forall x_1 \psi(x_1)$$

for some (partially quantified) formula ψ, the algorithms first substitute $x_1 = 0$ in ψ, runs the recursion and get $y_0 = A(\psi(0))$. It writes y_0 down and deletes the working memory. The algorithms then computes $y_1 = A(\psi(1))$ and outputs $y_0 \wedge y_1$. The case of $\exists x_1 \psi(x_1)$ is similar. The memory size is

$$O(\text{the number of quantifiers plus the size of the inner formula}).$$

It remains to prove that it is PSPACE-hard. Let $L \in \mathsf{PSPACE}$ and let M be a TM with poly-space deciding M and fix an input x. Let

$$s = \mathsf{SPACE}(M, x).$$

We wish to construct a TQBF ψ so that

$$M(x) = 1 \iff \psi = 1.$$

We translate $M(x) = 1$ to the reachability problem in $G_{M,x}$ from the initial state to an accepting state. We want ψ to capture this reachability.

> How can we do that? We want to capture "there is a path". We have $\exists$ at our disposal. But how can $\forall$ help?

We discussed a formula $\varphi = \varphi_{M,x}$ that computes if v, u is an edge in $G_{M,x}$. In other words, the formula

$$\varphi(v, u)$$

checks if there is a path of length one from v to u. The formula

$$\exists w \ \varphi(v, w) \wedge \varphi(w, u)$$

checks if there is a path of length at most two from v to u.

> Can we keep on going?

The answer is, on the face of it, no because the length of the desired path could be exponential in s and so will the size of the overall formula we get. And we also did not use the $\forall$ quantifier.

> The main observation is that if $\phi_i(v, u)$ computes if there is a path of length at most 2^i between v, u, then
>
> $$\phi_{i+1}(v, u) = \exists w \ \forall a, b \ ((a = v) \wedge (b = w)) \vee ((a = w) \wedge (b = u)) \to \phi_i(a, b)$$
>
> computes if there is a path of length at most 2^{i+1} between v, u.

In words, there is some w so that $\phi_i(v, w)$ and $\phi_i(w, u)$.

> We doubled the distance covered and increased the formula size by an additive factor.

Recall that v, u, w are $O(s)$-bit strings. So, the size of ϕ_{i+1} is at most $O(s)$ plus the size of ϕ_i. The final formula is chosen to be $\psi = \phi_{O(s)}$ and its size is $O(s^2)$. It holds that

$$\psi(v_0, v_{accept}) = 1 \iff M(x) = 1.$$

$\square$

Remark. *The theorem can be thought of saying that TQBF is "extremely hard". In other words, in general deciding if a player in a two-player game (like chess) has a winning strategy is hard.*

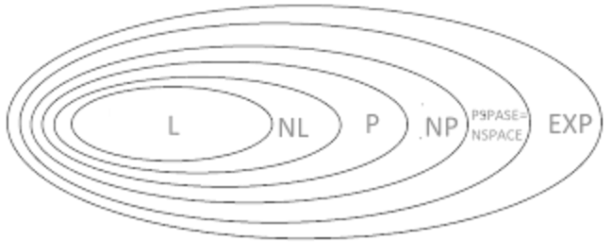
Remark. *What properties of $G = G_{M,x}$ did the proof use? It has size $2^{poly(n)}$. Deciding the edges of G can be done in time $poly(n)$ and space $poly(n)$. We never used the fact that the out-degrees are one. So it applies both to deterministic as well as non-deterministic computations!*

Theorem 20. $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

The ideas we have developed, in fact, allow to prove the following general result:

Theorem 21 (Savitch 1970)**.** *If $S(n) \geq \log n$ then*
$$\mathsf{NSPACE}(S(n)) \subseteq \mathsf{SPACE}((S(n))^2).$$

Remark. *A standard picture in computational complexity looks like:*



1.4. **NL.**

Remark. *Again, we try to identify complete problems for* NL*. What is the suitable notion of reductions? Allowing reductions to run in poly-time is not suitable because* $\mathsf{NL} \subseteq \mathsf{P}$*.*

Definition 22. *A function $f : \{0,1\}^* \to \{0,1\}^*$ is implicitly log-space computable (ILC) if*

(1) there is $k > 0$ so that for all x,
$$|f(x)| \leq |x|^k.$$
(2)
$$\{\langle x,i\rangle : f(x)_i = 1\} \in \mathsf{L}.$$
(3)
$$\{\langle x,i\rangle : i \leq |f(x)|\} \in \mathsf{L}.$$

Remark. *The first item says that f does not output very long strings. The other two items say that we can compute in log-space the i'th bit of $f(x)$ as long as it makes sense.*

Definition 23. *We write $A \leq_\ell B$ if there is an ILC f so that for all x, we have $x \in A$ iff $f(x) \in B$.*

Remark. *This type of reductions satisfy the following two natural properties:*

(1) If $A \leq_\ell B$ and $B \in \mathsf{L}$ then
$$A \in \mathsf{L}.$$
(2) If $A \leq_\ell B$ and $B \leq_\ell C$ then
$$A \leq_\ell C.$$

This is intuitive but not obvious; space must be reused here. The details are left as an exercise.

Definition 24. *A language $C \subseteq \{0,1\}^*$ is* NL*-complete if it is in* NL *and for every $L \in \mathsf{NL}$ we have $L \leq_\ell C$.*

Theorem 25. *The language*
$$\mathsf{PATH} = \{\langle G,s,t\rangle : G \textit{ is a digraph, } s,t \in V(G)\textit{, there is a path from } s \textit{ to } t\}$$
is NL*-complete.*

Remark. *We shall not fully prove but we have seen all ideas. First,* $\mathsf{PATH} \in \mathsf{NL}$ *because we can guess the path from s and accept only if we reach t. In this process, we "forget the history". Second, if $L \in \mathsf{NL}$ then there is a NTM N that decides it that uses log-space, so we can define a reduction:*

$$f(x) = \langle G_{N,x}, v_0, v_{accept} \rangle.$$