A Crash Course in Complexity Theory with a view towards `ustcon` $\in L$

Anurag Sahay

University of Rochester

asahay@ur.rochester.edu

25th February, 2020

Overview of the Talk

1. Introduction
   - What is Computation?

2. Formalizing *Efficient* Computation
   - Time Complexity: P
   - Space Complexity: L
   - The Relationship between Time and Space Complexity

3. Reingold's Theorem: $\texttt{ustcon} \in L$

4. Nondeterminism in Space Complexity

References

Sanjeev Arora and Boaz Barak.
Computational complexity: a modern approach.
Cambridge University Press, 2009.

Omer Reingold.
Undirected connectivity in log-space.
Journal of the ACM (JACM), 55(4):1–24, 2008.

Decision Problems

We will restrict ourselves to decision problems for this talk. A decision problem is a function $f : \{0,1\}^* \to \{0,1\}$.

Equivalently, a decision problem is a subset $\mathtt{L} \subseteq \{0,1\}^*$.

Decision Problems

We will restrict ourselves to decision problems for this talk. A decision problem is a function $f : \{0,1\}^* \to \{0,1\}$.

Equivalently, a decision problem is a subset $\mathtt{L} \subseteq \{0,1\}^*$.

When asking yes-or-no questions about countable collections of discrete objects (like graphs, integers, binary strings) it is usually possible to encode the question as a decision problem. For a toy example, consider the `parity` problem: $n \in \mathbb{N}$, is the number of 1 in the binary expansion of n odd?

Turing Machine

Definition

A deterministic k-tape Turing machine M is a tuple (Q, Γ, δ) where

- Q is a finite set of states.
- Γ is a finite set of symbols, called an alphabet. These are the symbols that can be written on any of the tapes of the Turing machine.
- $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$ is called the transition function.
- There are two distinguished states $q_{\mathtt{start}}, q_{\mathtt{halt}} \in Q$. $q_{\mathtt{halt}}$ has the property that $\delta(q_{\mathtt{halt}}, \cdot)$ does not change any tapes, or move any of the heads.
- There is a distinguished symbol $\square \in \Gamma$, representing a blank space.

In particular, we can choose $\Gamma = \{0, 1, \square\}$, and set $k = 3$. As long as Γ has at least two non-blank elements, it doesn't practically change anything we will discuss.

What is an Efficient Computation?

There are two basic resources a computer uses: time and space.

What is an Efficient Computation?

There are two basic resources a computer uses: time and space.

Informally, we define these resources as follows:

Definition

For a given deterministic Turing machine M, and a given input $x \in \{0,1\}^*$, the time $T_M(x)$ on that input is the number of transitions M makes on input x before it halts.

What is an Efficient Computation?

There are two basic resources a computer uses: time and space.

Informally, we define these resources as follows:

Definition

For a given deterministic Turing machine M, and a given input $x \in \{0,1\}^*$, the time $T_M(x)$ on that input is the number of transitions M makes on input x before it halts.

The space $S_M(x)$ is the largest distance travelled by the head of any work tape before M halts on input x.

What is an Efficient Computation?

There are two basic resources a computer uses: time and space.

Informally, we define these resources as follows:

Definition

For a given deterministic Turing machine M, and a given input $x \in \{0,1\}^*$, the time $T_M(x)$ on that input is the number of transitions M makes on input x before it halts.

The space $S_M(x)$ is the largest distance travelled by the head of any work tape before M halts on input x. Further, for $n \in \mathbb{N}$,

$$T_M(n) = \max_{|x| \leq n} T_M(x)$$

$$S_M(n) = \max_{|x| \leq n} S_M(x)$$

What is an Efficient Computation?

This leads us to the following definitions:

Definition (*DTIME*)

For $\mathtt{L} \subseteq \{0,1\}^*$, we say $\mathtt{L} \in DTIME(f(n))$, if there is a Turing machine M that decides $\mathtt{L}$ such that

$$T_M(n) = \mathcal{O}(f(n))$$

What is an Efficient Computation?

This leads us to the following definitions:

Definition (*DTIME*)

For $\mathtt{L} \subseteq \{0,1\}^*$, we say $\mathtt{L} \in \mathit{DTIME}(f(n))$, if there is a Turing machine M that decides $\mathtt{L}$ such that

$$T_M(n) = \mathcal{O}(f(n))$$

Definition (*DSPACE*)

Similarly, we say $\mathtt{L} \in \mathit{DSPACE}(f(n))$, if there is a Turing machine M that decides $\mathtt{L}$ such that

$$S_M(n) = \mathcal{O}(f(n))$$

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

- On the work tape, write 0.

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

- On the work tape, write 0.
- Read the input tape from left to right, and do the following:
  - If the symbol read is a 0, the simply keep reading the input (i.e., move right) without changing anything else.

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

- On the work tape, write 0.
- Read the input tape from left to right, and do the following:
  - If the symbol read is a 0, the simply keep reading the input (i.e., move right) without changing anything else.
  - If the symbol read is a 1, then switch the symbol on the work tape (from 0 to 1 and vice-versa). Continue reading the input after.

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

- On the work tape, write 0.
- Read the input tape from left to right, and do the following:
  - If the symbol read is a 0, the simply keep reading the input (i.e., move right) without changing anything else.
  - If the symbol read is a 1, then switch the symbol on the work tape (from 0 to 1 and vice-versa). Continue reading the input after.
  - If the symbol read is a $\square$, copy the symbol from the work tape to the output and halt.

Toy example: `parity`

We now describe a Turing machine that decides `parity`, as follows:

- On the work tape, write 0.
- Read the input tape from left to right, and do the following:
  - If the symbol read is a 0, the simply keep reading the input (i.e., move right) without changing anything else.
  - If the symbol read is a 1, then switch the symbol on the work tape (from 0 to 1 and vice-versa). Continue reading the input after.
  - If the symbol read is a $\square$, copy the symbol from the work tape to the output and halt.

It's not hard to formalize the above description, and it is easy to see that `parity` $\in DTIME(n) \cap DSPACE(1)$.

What is a Time Efficient Computation?

Returning to the formalizing efficient computability. Specificially, for time-efficiency,:

Definition (Polynomial Time)

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

What is a Time Efficient Computation?

Returning to the formalizing efficient computability. Specifically, for time-efficiency,:

Definition (Polynomial Time)

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

Why P? Isn't an algorithm with time complexity $\mathcal{O}(n^{\log \log \log n})$ superior to one with complexity $\mathcal{O}(n^{1000})$? Why not define it as $DTIME(n^2)$?

What is a Time Efficient Computation?

Returning to the formalizing efficient computability. Specifically, for time-efficiency,:

Definition (Polynomial Time)

$$P = \bigcup_{k \geq 1} DTIME(n^k)$$

Why P? Isn't an algorithm with time complexity $\mathcal{O}(n^{\log \log \log n})$ superior to one with complexity $\mathcal{O}(n^{1000})$? Why not define it as $DTIME(n^2)$?

- Extended Church-Turing hypothesis: on all typical models of computations, the notion of P is invariant under simulation.
- P is closed under composition (calling polynomially many subroutines each taking polynomial time will not increase the running time beyond polynomial).

Another Time Complexity Class

We have the following definition:

Definition (Exponential Time)

$$EXP = \bigcup_{k \geq 1} DTIME(2^{n^k})$$

Another Time Complexity Class

We have the following definition:

Definition (Exponential Time)

$$EXP = \bigcup_{k \geq 1} DTIME(2^{n^k})$$

Clearly $P \subseteq EXP$. In fact, it is known that $P \subsetneq EXP$.

What is the space-efficiency analogue of P?

We now want a notion of space-efficient computation. A naive idea would be to mimic the previous definition, to get

Definition (Polynomial Space)

$$PSPACE = \bigcup_{k \geq 1} DSPACE(n^k)$$

What is the space-efficiency analogue of P?

We now want a notion of space-efficient computation. A naive idea would be to mimic the previous definition, to get

Definition (Polynomial Space)

$$PSPACE = \bigcup_{k \geq 1} DSPACE(n^k)$$

However, here's a heuristic for why this is not a very efficient notion of space complexity: which way does the inclusion between P and $PSPACE$ go?

What is the space-efficiency analogue of P?

We now want a notion of space-efficient computation. A naive idea would be to mimic the previous definition, to get

Definition (Polynomial Space)

$$PSPACE = \bigcup_{k \geq 1} DSPACE(n^k)$$

However, here's a heuristic for why this is not a very efficient notion of space complexity: which way does the inclusion between P and $PSPACE$ go?

It is known that $P \subseteq PSPACE$, and it is actually expected that $P \subsetneq PSPACE$ (for example, $P \neq NP$ would imply this).

What is the space-efficiency analogue of P?

The analogue for P is space-complexity is the following:

Definition (LogSpace)

$$L = DSPACE(\log n)$$

What is the space-efficiency analogue of P?

The analogue for P is space-complexity is the following:

Definition (LogSpace)

$$L = DSPACE(\log n)$$

Note that clearly $L \subseteq PSPACE$. In fact, $L \subseteq P$, as we will show soon.

How do these notions relate?

We have the following theorem:

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

How do these notions relate?

We have the following theorem:

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

Proof.

The first inclusion is trivial: note that in time t, no head of the TM could have used more space than t.

How do these notions relate?

We have the following theorem:

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

Proof.

The first inclusion is trivial: note that in time t, no head of the TM could have used more space than t.

For the second, suppose that M uses s space on some input x. We define the configuration graph $G_{M,x}$ to be the directed graph whose vertices are configurations, and $v \to u \iff \delta(v) = u$. Then, there are at most $2^{\mathcal{O}_M(s)}$ many configurations. Note $G_{M,x}$ must be a directed acyclic graph (otherwise there would be infinite loops), and hence the time taken is at most the largest walk in the graph, and hence $\ll 2^{\mathcal{O}_M(s)}$. □

How do these notions relate?

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

As an immediate corrolary to the above theorem, we see that $L \subseteq P \subseteq PSPACE \subseteq EXP$, some of which we claimed earlier.

None of these are known to be strict (though it is known that $L \subsetneq PSPACE$ and $P \subsetneq EXP$).

The decision problem `ustcon`

We now describe `ustcon`.

Definition (Undirected *st*-connectivity)

`ustcon` is the following decision problem:

Input: an undirected graph $G = (V, E)$, and two vertices $s, t \in V$

Output: 1 if $s \rightsquigarrow t$ in G, and 0 otherwise.

The decision problem `ustcon`

We now describe `ustcon`.

Definition (Undirected *st*-connectivity)

`ustcon` is the following decision problem:

Input: an undirected graph $G = (V, E)$, and two vertices $s, t \in V$

Output: 1 if $s \rightsquigarrow t$ in G, and 0 otherwise.

Note that this can be encoded in a way such that the input size is a fixed polynomial in $|V|$, so in particular, we can replace the size of the input with $|V| = n$ in our estimates.

Reingold's Theorem

This brings us to the goal of the next talk:

Theorem (Reingold, 2005)

$$\texttt{ustcon} \in L$$

We will prove this next time. For now, we provide some motivation for why this result is possibly surprising.

Nondeterministic Turing Machine

Before we motivate the importance of Reingold's theorem, we consider an augmented model of computation, called the nondeterministic Turing machine.

Definition

A nondeterministic Turing machine N is a tuple (Q, Γ, δ) where the definition is the same as that of a Turing machine, except that instead of being a single-valued function, δ is a multi-valued function (i.e., it is a relation). We say that an NTM N accepts precisely when at least one of the paths accepts, and it rejects when all paths reject.

Nondeterministic Turing Machine

Before we motivate the importance of Reingold's theorem, we consider an augmented model of computation, called the nondeterministic Turing machine.

Definition

A nondeterministic Turing machine N is a tuple (Q, Γ, δ) where the definition is the same as that of a Turing machine, except that instead of being a single-valued function, δ is a multi-valued function (i.e., it is a relation). We say that an NTM N accepts precisely when at least one of the paths accepts, and it rejects when all paths reject.

Clearly, every deterministic Turing machine is a nondeterministic one.

Nondeterministic Complexity

All of the complexity classes we defined earlier generalize:

Definition (*NTIME*)

For $\mathtt{L} \subseteq \{0,1\}^*$,$\mathtt{L} \in NTIME(f(n))$, if there is a nondeterministic Turing machine N that decides $\mathtt{L}$ such that

$$T_N(n) = \mathcal{O}(f(n))$$

Definition (*NSPACE*)

$\mathtt{L} \in NSPACE(f(n))$, if there is a nondeterministic Turing machine N that decides $\mathtt{L}$ such that

$$S_N(n) = \mathcal{O}(f(n))$$

Nondeterministic Complexity

Definition (Nondeterministic Polynomial Time)

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

Definition (Nondeterministic LogSpace)

$$NL = NSPACE(\log n)$$

Nondeterministic Complexity

Definition (Nondeterministic Polynomial Time)

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

Definition (Nondeterministic LogSpace)

$$NL = NSPACE(\log n)$$

Big open problem: does adding nondeterminism to Turing machines change efficiently computable classes? Clearly $DTIME \subseteq NTIME$ and $DSPACE \subseteq NSPACE$. Is $P = NP$? Is $L = NL$? Is $EXP = NEXP$?

Nondeterministic Complexity

Definition (Nondeterministic Polynomial Time)

$$NP = \bigcup_{k \geq 1} NTIME(n^k)$$

Definition (Nondeterministic LogSpace)

$$NL = NSPACE(\log n)$$

Big open problem: does adding nondeterminism to Turing machines change efficiently computable classes? Clearly $DTIME \subseteq NTIME$ and $DSPACE \subseteq NSPACE$. Is $P = NP$? Is $L = NL$? Is $EXP = NEXP$?

All open! (modulo some interrelations, like $EXP = NEXP \implies P = NP$).

How does nondeterministic space relate to time?

In fact, a more careful proof of the theorem relating deterministic space to deterministic time tells us that

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq NSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

In particular, this has a corrolary that $NL \subseteq P$.

How does nondeterministic space relate to time?

In fact, a more careful proof of the theorem relating deterministic space to deterministic time tells us that

Theorem (Theorem 4.3 from [AB09])

$$DTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq NSPACE(f(n)) \subseteq DTIME(2^{\mathcal{O}(f(n))})$$

In particular, this has a corrolary that $NL \subseteq P$.

We also have the following relationship between the various logarithmic space classes:

Theorem

$$L \subseteq SL \subseteq RL \subseteq NL \subseteq L^2$$

Reductions and Completeness

We say that

$$\mathtt{L}_1 \leq_P \mathtt{L}_2$$

that is $\mathtt{L}_1$ is Karp reducible (or polynomial-time reducible) to $\mathtt{L}_2$, if there is a polynomial time Turing machine $M : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in \mathtt{L}_1 \iff M(x) \in \mathtt{L}_2$$

Reductions and Completeness

We say that

$$\mathtt{L}_1 \leq_P \mathtt{L}_2$$

that is $\mathtt{L}_1$ is Karp reducible (or polynomial-time reducible) to $\mathtt{L}_2$, if there is a polynomial time Turing machine $M : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in \mathtt{L}_1 \iff M(x) \in \mathtt{L}_2$$

We say $\mathtt{L}$ is *NP*-hard, if for every $\mathtt{L}' \in NP$, $\mathtt{L}' \leq_P \mathtt{L}$. If $\mathtt{L} \in NP$ is *NP*-hard, then we say that $\mathtt{L}$ is *NP*-complete.

Reductions and Completeness

We say that

$$\mathtt{L}_1 \leq_P \mathtt{L}_2$$

that is $\mathtt{L}_1$ is Karp reducible (or polynomial-time reducible) to $\mathtt{L}_2$, if there is a polynomial time Turing machine $M : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in \mathtt{L}_1 \iff M(x) \in \mathtt{L}_2$$

We say $\mathtt{L}$ is *NP*-hard, if for every $\mathtt{L}' \in NP$, $\mathtt{L}' \leq_P \mathtt{L}$. If $\mathtt{L} \in NP$ is *NP*-hard, then we say that $\mathtt{L}$ is *NP*-complete.

Theorem (Cook-Levin, 1971)

SAT *is NP-complete.*

Reductions and Completeness

When working with NL, the appropriate notion of reduction is logspace reducible, denoted by $\leq_L$. We will not go into the technical definition of this.
We have the following theorem:

Theorem

`stcon` *is NL-complete.*

Here `stcon` is directed connectivity of s, t in a digraph G.

Reductions and Completeness

When working with NL, the appropriate notion of reduction is logspace reducible, denoted by $\leq_L$. We will not go into the technical definition of this.
We have the following theorem:

Theorem

`stcon` *is NL-complete.*

Here `stcon` is directed connectivity of s, t in a digraph G.

The proof idea is convert any NL problem into the configuration graph of the nondeterministic Turing machine that solves it in log-space, and then ask whether the accept state is reachable from the start state.

Connecting deterministic and nondeterministic space

Finally, we have the following theorem:

Theorem (Savitch)

$$NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$$

The proof idea is the following; first, there is an algorithm which demonstrates that $\texttt{stcon} \in L^2$. In particular, this shows that $NL \subseteq L^2$.

Connecting deterministic and nondeterministic space

Finally, we have the following theorem:

Theorem (Savitch)

$$NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$$

The proof idea is the following; first, there is an algorithm which demonstrates that $\texttt{stcon} \in L^2$. In particular, this shows that $NL \subseteq L^2$.

Now, for $\mathtt{L} \in NSPACE(f(n))$, there is a nondeterministic Turing machine N which decides it. Thus, solving L is equivalent to figuring out whether the accepting configuration is reachable from the starting configuration in the configuration graph of N. This graph has size $2^{\mathcal{O}(f(n))}$, and so, this takes time $\mathcal{O}(f(n)^2)$.

The End