Computational Complexity Theory

Lavesh Mangal *Divyansh Agarwal* *Keshav Ranjan*
Kumar Saurav *Tanush Kumar*
MTH401A Presentation
Department of Mathematics and Statistics,
Indian Institute of Technology Kanpur.
Course Instructor: *Mohua Banerjee*

April 15, 2024

Outline

Time Complexity of Turing Machines, P and NP

Big - O notation

Definition

(*Big - O*) For functions $f, g : \mathbb{N} \to \mathbb{R}$, we say that f is of order g, denoted $f(n) = O(g(n))$, if for all large n, f is upper bounded by a constant multiple of g. Formally,

$$\exists c \in \mathbb{R}^+, \exists N \in \mathbb{N}, \forall n > N \qquad f(n) \leq cg(n))$$

Examples

1. $n^2 + 2n + 2 = O(n^2)$
2. $n! = O(n^n)$
3. $\forall \epsilon > 0, \log n = O(n^\epsilon)$

Time Complexity

Definition

(*Time complexity of a TM M*) Let M be a Turing machine that halts on all inputs. The *time complexity* of M is defined as the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M takes before halting, on any input of size n.

Remark

For a non-deterministic Turing Machine we require that each branch of the TM halts on all inputs. $f(n)$ for an NTM is the maximum number of steps on any branch of its computation.

Definition

($t(n)$-Turing Machine) A Turing Machine with time complexity $t(n)$ is called a $t(n)$-Turing machine.

Examples

1. The TMs R, L, σ are $O(1)$-TMs.
2. $R_{\#}$ is an $O(n)$-TM.

Definition

The class of all languages that are decidable by a deterministic $t(n)$-TM is called DTIME$(t(n))$. The class of languages decidable by a non-deterministic $t(n)$-TM is called NTIME$(t(n))$.

Complexity relationships between different models

Theorem

Every $t(n)$ multi-tape Turing Machine with $t(n) \geq n$, can be simulated using an $O(t^2(n))$ single-tape Turing Machine.

Proof.

Let $M = (Q, \Sigma, \delta, q_0, h)$be a k-tape Turing Machine. Here Σ denotes the tape alphabet, containing the blank symbol $\#$. Formally, the transition function is as follows:

$$\delta : Q \times \Sigma^k \rightarrow (Q \cup \{h\}) \times (\Sigma \cup \{L, R\})^k$$

That is, on reading the current tape alphabets $a_1, a_2, \ldots a_k$ on the k tapes, it moves to some other state, and on each tape, either writes something or moves left or right. We construct a single-tape machine S that simulates the execution of M.

Multitape to single-tape: Design

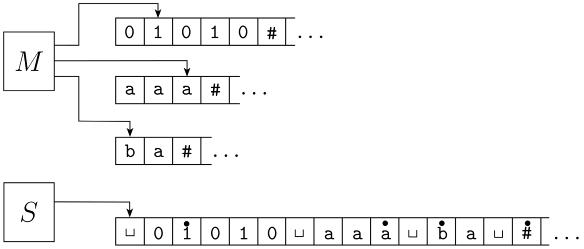


Figure: Equivalent single-tape machine

The alphabet for the single-tape machine is constructed as follows:

- The contents of the k tapes are separated by the delimiter ⊔.
- For each section in S with the contents of one tape in M, there is a single dotted alphabet, indicating that the head is there.

Multitape to single-tape: Execution

On input w,

1. S puts its tape into a format representing all the k tapes of M. The formatted tape is

$$\sqcup \dot{w}_1 w_2 w_3 \dots w_n \sqcup \dot{\#} \sqcup \dot{\#} \sqcup \dot{\#} \dots \sqcup$$

2. To simulate a single move, S scans its tape from the first $\#$ to the last $\#$ to determine the symbols under the heads of each tape. Then, it makes a second pass to update the tapes according to the transition function of M.
3. If at any point S moves one of the virtual heads from onto a $\sqcup$, this means that M has equivalently moved the head of the respective tape into a previously unread blank portion of the tape. Thus, whenever this happens, we write a $\#$ there and move the contents of S's tape from this $\#$ to the right-end, one cell to the right.

Multitape to single-tape: Time Complexity

Let the part of a tape that we have already seen be the *active* part of the tape. We will need the following key observation: since we can only move one cell in one step on any tape, the active part of any tape cannot have length greater than $t(n)$. Thus, in order to do one pass of S's tape, it will take at most $O(kt(n)) = O(t(n))$ time.
The first step, where S puts the tape into the starting configuration takes $O(n)$ time. Afterwards, it takes $O(t(n))$ time for any step of M. Thus, the execution takes $O(t^2(n))$ time. The total time complexity is thus, $O(n + t^2(n)) = O(t^2(n))$.

Remark

The $t(n) \geq n$ is a reasonable assumption, because it takes $O(n)$ time to just read the input completely.

NTM to TM

Theorem

Let $t(n) \geq n$. Then every $t(n)$-NTM can be simulated using an $2^{O(t(n))}$-TM.

Let M be an $t(n)$- NTM. Let b be the number of legal transitions allowed by M. Then we can visualize the set of all possible computations of M as proceeding along a b-ary tree of depth at most $t(n)$.

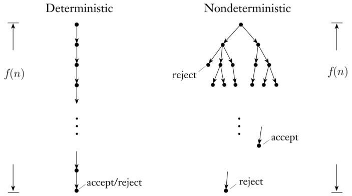


Figure: b-ary tree of executions of an NTM

NTM to TM: Time Complexity

We consider 3-tape Turing machine S that tries out all the possible branches of an NTM. Effectively, we perform a breadth-first search of the tree, visiting all the nodes at depth 1, followed by all the nodes at depth 2 and so on.

1. S maintains the input on the first tape without changing it.
2. For each node in the tree in the breadth-first order, it writes down the location of that node on the third tape for example, the node 23 denotes the third child of the second child of the root. Then it copies the first tapes contents onto the second tape and simulates the execution of M along the branch defined by the third tape.
3. If any any branch halts and rejects or no more transitions can be made, we proceed to the next node. Otherwise, if it accepts, we accept.

NTM to TM: Time Complexity

- The execution corresponding to a single node : $t(n)$
- Time for $O(b^{t(n)})$ nodes : $O(t(n)b^{t(n)}) = 2^{O(t(n))}$
- Converting 3-tape TM to single tape TM : $2^{2O(t(n))} = 2^{O(t(n))}$

The class P

Definition

(P) P is the class of languages that are decidable in polynomial time on a deterministic single-tape TM, i.e.

$$\mathsf{P} = \bigcup_k \mathsf{DTIME}(n^k)$$

The class P is important because:

- It is invariant under all the polynomially-equivalent models of computation.
- It corresponds roughly to the class of problems that can efficiently be solved on a computer.

The class NP

Definition

(NP) NP is the class of languages that are decidable in polynomial time on a non-deterministic single-tape TM, i.e.

$$\mathsf{NP} = \bigcup_k \mathsf{NTIME}(n^k)$$

As we shall see, NP contains several algorithmically significant problems and admits a distinct algorithmic characterization.

The class NP

Encodings for Turing Machines

We can encode a wide variety of mathematical objects using the alphabet $\{0, 1\}$. These include:

1. Integers in binary encoding
2. Elements of finite fields, $\mathbb{Q}$
3. Polynomials over the above fields
4. Graphs $G = (V, E)$ by storing their adjcacency lists/adjacency matrices

We will also assume that we can encode finite sequences of the above objects using some pairing method. The input consisting of the objects $i_1, i_2, \ldots i_k$ will be denoted by $\langle i_1, i_2, \ldots, i_k \rangle$.

Example problems in P

Theorem

Let PATH *be the following problem on directed graphs:*

$$\mathrm{PATH} = \{\langle G, s, t\rangle : \textit{ there is a path from } s \textit{ to } t \textit{ in } G\}$$

Then, $\mathrm{PATH} \in \mathsf{P}$*.*

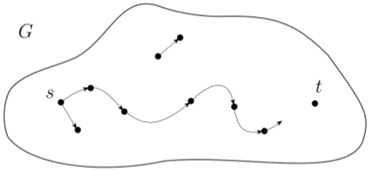


Figure: The PATH problem: is there a path from s to t

Proof.

Consider the Turing Machine that on input $\langle G, s, t\rangle$ does the following:

1. Mark the vertex s.
2. Repeat the following procedure until no more vertices are marked:
    1. For each edge $(u, v) \in E$, if u has been marked and v has not been marked, mark v.
3. If t has been marked, then accept. Otherwise, reject.

Let $|V| = n, |E| = m \leq n^2$.
It is easy to see that any step of the process takes at most polynomial time.
Note that since each iteration of step 2, we must mark a vertex, therefore, there can be at most n such steps.
Thus, M is a polynomial-time algorithm for PATH.

Alternative characterization of NP

Consider the following problems:

1. $$\mathrm{COMPOSITES} = \{x \in \mathbb{N} : (\exists p, q \in \mathbb{N})(p \geq 1 \wedge q \geq 1 \wedge x = pq)\}$$

2. We call a path in a directed graph G that goes through all the vertices exactly once a *Hamiltonian Path*. Then, consider

   $$\mathrm{HAMPATH} = \{\langle G, s, t\rangle : \text{there is a Hamiltonian path from } s \text{ to } t\}$$

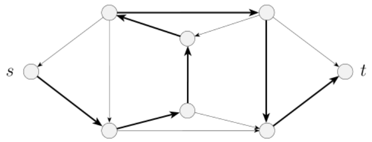


Figure: An example of a Hamiltonian Path

$\mathrm{HAMPATH} \in \mathsf{NP}$

Consider the following NTM for deciding $\mathrm{HAMPATH}$. On input $\langle G, s, t\rangle$ it does:

1. Non-deterministically write down a sequence of vertices $v_1, v_2, \ldots v_n$.
2. Check for any repetitions in the list. If there is any vertex that appears twice, *reject*.
3. Check if $v_1 = s$ and $v_n = t$. If not, *reject*.
4. For each $i \in \{1, 2, \ldots, n-1\}$, check if $(v_i, v_{i+1}) \in E$. If not, *reject*.
5. If not rejected yet, *accept*.

It is easy to see that all the steps run in non-deterministic polynomial time in the length of the input. Thus, $\mathrm{HAMPATH} \in \mathsf{NP}$.

Remark

$\mathrm{COMPOSITES}$ *follows similarly by "guessing"* p *and* q. *Further, it has been shown that* $\mathrm{COMPOSITES} \in \mathsf{P}$.

Alternative characterization of NP(contd)

There is a common element to both $\mathrm{COMPOSITES}$ and $\mathrm{HAMPATH}$ which is that while we may not know of a polynomial time algorithm for deciding them, given a solution we can *verify* it efficiently.

1. For any $x \in \mathbb{N}$, given a factorization p, q we can test in time polynomial in the size of p, q whether $x = pq$ by multiplying p, q.
2. For any $\langle G, s, t \rangle$, given a path of length n, $v_1, v_2, \ldots v_n$, we can verify in time polynomial in the size of the input whether it is a Hamiltonian path or not.

That is, both of these problems have *efficient verifiers*.

Efficient verifiers

Definition

A *verifier* for a language L, is a TM $\mathcal{A}$ such that

$$L = \{w \in \Sigma^* : \mathcal{A} \text{ accepts } \langle w, c \rangle \text{ for some } c\}$$

We measure the time complexity of $\mathcal{A}$ in terms of the length of w. If $\mathcal{A}$ runs in time polynomial in the length of w, then it is called a *polynomial time verifier*, whence L is said to be *polynomially verifiable*.

Remark

The string c is called a certificate for w. If $\mathcal{A}$ is a polynomial time verifier, then there must be a certificate with length polynomial in $|w|$ because a poly-time algorithm can only read a polynomial length certificate.

NP only if polynomially verifiable

Theorem

A language $L \in \mathrm{NP}$ *only if it is polynomially verifiable.*

Proof.

Let $L \in \mathrm{NP}$. Then, $\exists M$, an NTM, that decides L. We construct the verifier V as follows. V on input $\langle w, c\rangle$ does:

1. Simulate, M on w treating the string c as the sequence of non-deterministic choices made by M.
2. If this branch of M's computation accepts, *accept* else *reject*.

NP if polynomially verifiable

Theorem

A language $L \in \mathrm{NP}$ if it is polynomially verifiable.

Proof.

Let V be a polynomial verifier for L. Suppose that V runs in time n^k for some k and input size n. Consider the NTM M that does the following on input w:

1. Choose a string c of length at most n^k non-deterministically.
2. Run V on $\langle w, c \rangle$.
3. If V accepts, *accept*, else *reject*.

The theory of NP-Completeness

A motivating example

$S \subseteq V$ is a vertex-cover of G, if each edge in G is incident on some vertex in S i.e. S *covers* all the edges.

$$\text{VERTEX-COVER} = \{\langle G, k\rangle : G \text{ is an undirected graph that contains a vertex cover of size } \leq k\}$$

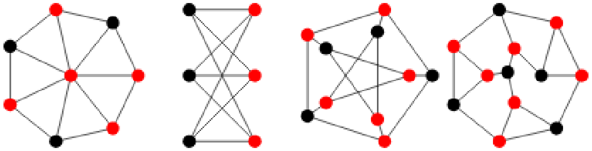


Figure: Examples of vertex covers for various graphs

Easy to see that $\text{VERTEX-COVER} \in$ NP, the vertex cover is a certificate verifiable in polytime.

A motivating example

$S \subseteq V$ is an independent set of G if no two vertices of S are connected by an edge i.e. the vertices of S are independent of each other.

$$\text{INDEPENDENT-SET} = \{\langle G, k\rangle : G \text{ is an undirected graph that contains an independent set of size } \geq k\}$$

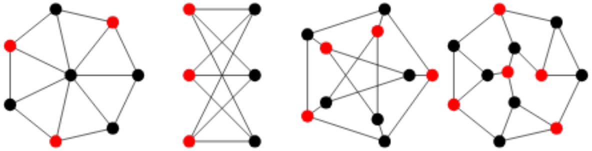


Figure: Examples of independent sets for various graphs

Easy to see that $\text{INDEPENDENT-SET} \in$ NP, the independent set is a certificate verifiable in polytime.

An observation

Let S be a vertex cover. Color each vertex of S red. Let $T = V \setminus S$. Color each vertex of T black.

Theorem

*There cannot be an edge between black vertices i.e. $T = V \setminus S$ is an **independent set**.*

Let T be an independent set. Color each vertex of T red. Let $S = V \setminus T$. Color each vertex of S black.

Theorem

*There cannot be an edge not covered by black vertices i.e. S is a **vertex cover**.*

Thus, there is a vertex-cover of size at most k, if and only if there is an independent set of size at least $n - k$.

Complexity theoretic consequence

The observation implies that it is possible for us to decide $\mathrm{VERTEX\text{-}COVER}$ using an algorithm for $\mathrm{INDEPENDENT\text{-}SET}$ by converting instances of the first into another in polynomial time. Let $f(\langle G, k\rangle) = \langle G, |V(G)| - k\rangle$. Then,

$$\langle G, k\rangle \in \mathrm{VERTEX\text{-}COVER} \iff f(G, k) \in \mathrm{INDEPENDENT\text{-}SET}$$

Polynomial time reducibility

Definition

A function $f : \Sigma^* \to \Sigma^*$ is called a polynomial time computable function, if there is a polynomial time TM that on input w, halts with $f(w)$ on the tape.

Definition

Let A, B be languages. We say that A is *polynomial time reducible* to B written $A \leq_P B$, if there is a **polynomial-time** computable function $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \iff f(x) \in B$$

f is called the *polynomial-time reduction* of A to B

Understanding reducibility

$\leq_P$ is trivially reflexive with $f = \mathrm{id}$.

Theorem

$\leq_P$ is transitive.

Proof.

Let $A \leq_P B$ and $B \leq_p C$ with reductions f, g computable in time p, q. Then, consider the Turing Machine that on input w does the following:

1. Compute $f(w)$ and write it on the tape.
2. Compute $g(f(w))$ and write it on the tape.

Observe that the time taken for step 1 is $p(n)$. Further we know that $|f(w)| \leq p(n)$. The time taken for step 2 is, thus, at most $q(p(n))$. Total time $= O(p(n) + q(p(n)))$ which is polynomial.

Understanding reducibility

- Formally, $\leq_P$ is a *preorder* on the class NP.
- Informally, $\leq_P$ defines a relative measure of the hardness of problems. If $A \leq_P B$, then B is *"harder"* to solve than A, because from an algorithm for B, we can construct an algorithm for A.
- Key Question : *What is the hardest problem in* NP*?*
  What does such a problem Q look like?
  - For all $L \in \text{NP}, L \leq_P Q$
  - Such a problem has to capture the intrinsic hardness of every problem in NP.

Logic captures the hardness of NP

$$\text{INDEPENDENT-SET} = \{\langle G, k\rangle : G \text{ is an undirected graph} \\ \text{that contains an independent set of size } \geq k\}$$

Let $x_1, x_2, \ldots, x_n$ be Boolean variables, where $x_i = T$ iff $i \in S$.

1. For each edge $e = \{u, v\} \in E$, both u, v cannot be in the set. So, either $\neg x_u \vee \neg x_v$.

$$\bigwedge_{\{u,v\}\in E} (\neg x_u \vee \neg x_v)$$

2. No subset of vertices of size $n - k + 1$ should not be in S.

$$\bigwedge_{U\subseteq V, |U|=n-k+1} \bigvee_{v\in U} x_v$$

This is a somewhat bad encoding of the cardinality constraint. Can take $\Theta(\frac{n}{2}\binom{n}{n/2}) = \Theta(\sqrt{n}2^n)$ length. Better encodings exist.

Parting Shots

Definition

We call a language L NP-hard, if $\forall K \in \text{NP}, K \leq_P L$.

Definition

An NP-hard problem that is further in NP, is called NP-complete.

The Cook-Levin Theorem

SATISFIABILITY

- Let $x_i, 1 \leq i \leq n$ denote a boolean variable
- $X = \{x_1, x_2....x_n\}$ are positive literals
- $\bar{X} = \{\bar{x}_1, \bar{x}_2.....\bar{x}_n\}$ are negative literals
- $C \subseteq X \cup \bar{X}$ is a clause
- A **Boolean Formula** F is a set of clauses
- A **truth assignment** for F is a mapping of variables in X to the set $\{T, F\}$, where T stands for **true** and F stands for **false**.
- A clause C is said to be **satisfied** if it contains atleast one true literal
- A Boolean Formula F is said to be **satisfiable** if there exists a truth assignment s.t all clauses in F are satisfied.

SATISFIABILITY

Given a Boolean Formula F, is it satisfiable?

Cook-Levin Theorem

Cook-Levin Theorem

SATISFIABILITY is $\mathcal{NP}-$ Complete.

Recall that $\mathcal{L} \in \Sigma^*$ is $\mathcal{NP}-$ Complete if:

- $\mathcal{L} \in \mathcal{NP}$, and
- for all $\mathcal{L}' \in \mathcal{NP}$, there is a polynomial-time reduction from $\mathcal{L}'$ to $\mathcal{L}$.

It is easy to design a non-deterministic Turing Machine that can solve SATISFIABILITY in polynomial time. Therefore SATISFIABILITY $\in \mathcal{NP}$

Proving $\mathcal{NP}-$ Completeness

Theorem

Let $\mathcal{L}_1$ be $\mathcal{NP}-$ Complete and $\mathcal{L}_2 \in \mathcal{NP}$. If $\mathcal{L}_1 \leq_P \mathcal{L}_2$, then $\mathcal{L}_2$ is $\mathcal{NP}-$ Complete.

Proof. Let $\mathcal{L}$ be any language in $\mathcal{NP}$.

- As $\mathcal{L}_1$ is $\mathcal{NP}-$ Complete, there must be a polynomial-time reduction, $\mathcal{L} \xrightarrow{\tau_1} \mathcal{L}_1$.
- As $\mathcal{L}_1 \leq_P \mathcal{L}_2$, there must be a polynomial-time reduction, $\mathcal{L}_1 \xrightarrow{\tau_2} \mathcal{L}_2$.
- By transitivity of polynomial-time reductions, $\mathcal{L} \xrightarrow{\tau_2 \circ \tau_1} \mathcal{L}_2$ is also a polynomial-time reduction.

Hence, $\mathcal{L}_2$ is $\mathcal{NP}-$ Complete. We will show SATISFIABILITY is $\mathcal{NP}-$ Complete by showing BOUNDED-TILING $\leq_P$ SATISFIABILITY

Bounded Tiling Problem

A **Tiling System** is denoted by $\mathcal{D} = (D, H, V)$, where

- D denotes a finite set of tiles that can be used for tiling.
- Two tiles d, d' can be placed horizontally adjacent iff. $(d, d') \in H \subseteq D^2$.
- Two tiles d, d' can be placed vertically adjacent iff. $(d, d') \in V \subseteq D^2$.

Bounded Tiling Problem

Given a tiling system $\mathcal{D} = (D, H, V)$, an integer s, and a function describing the first row of the tiling, $f_0 : \{0, \ldots, s-1\} \to D$, is there an $s \times s$ tiling by $\mathcal{D}$ extending f_0?

Here, a **Tiling** by $\mathcal{D}$ is a function $f : \{0, \ldots, s-1\}^2 \to D$, such that

- $f(m, 0) = f_0(m)$, $\forall\ m < s$;
- $\big((f(m,n), f(m+1,n)\big) \in H$, $\forall\ m < s-1, n < s$;
- $\big((f(m,n), f(m,n+1)\big) \in V$, $\forall\ m < s, n < s-1$.

Bounded Tiling is $\mathcal{NP}-$ Complete

Theorem

The **Bounded Tiling Problem** is $\mathcal{NP}-$ Complete.

Proof.

1. Bounded Tiling is in $\mathcal{NP}$:
2. For any language $\mathcal{L} \in \mathcal{NP}$, $\mathcal{L} \leq_P$ Bounded Tiling :
   - i.e. we must define a polynomial-time computable function τ such that

$$x \in \mathcal{L} \xrightarrow{\tau} \left(\mathcal{D} = (D, H, V), s, f_0\right)$$

$$x \in \mathcal{L} \Leftrightarrow \exists\ s \times s \text{ tiling } f, \text{ extending } f_0$$

The Reduction τ

Since $\mathcal{L} \in \mathcal{NP}$, there is a non-deterministic Turing Machine $M = (K, \Sigma, \delta, s)$ such that

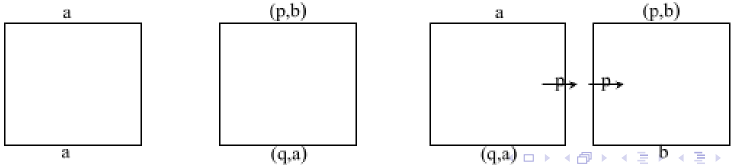
- All computations of M on input x halt within $p(|x|)$ steps for some polynomial p.
- There is an accepting computation on input x if and only if $x \in \mathcal{L}$.

For some $x \in \mathcal{L}$, we define $\tau(x) = \big(\mathcal{D} = (D, H, V), s, f_0\big)$ as follows,

1. s in $\tau(x)$ is defined as, $s = p(|x|) + 2$.
2. We construct the tiling system $\mathcal{D}$ using the Turing Machine M.
    - The tiling system is arranged so that if a tiling is possible, then the markings on the horizontal edges between the n^{th} and $(n+1)^{st}$ rows of tiles, read off the configuration of M after n steps of its computation.
    - Successive configurations appear one above the next.

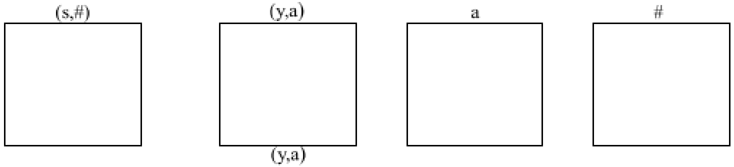
The Reduction τ

3. The tiles in D are defined as,
   - For each $a \in \Sigma$, this tile communicates any unchanged symbols upwards from configuration to configuration.
   - For each $a, b \in \Sigma$, and $p, q \in K$ such that $\delta(q, a) = (p, b)$, this tile communicates the head position upwards and also changes the state and scanned symbol appropriately.
   - For each $a \in \Sigma$, $q \in K$ such that $\delta(q, a) = (p, \rightarrow)$ for some $p \in K$, and for each $b \in \Sigma - \{\triangleright\}$, These tiles communicate head movement one square from left to right, while changing the state appropriately.
   - For each $a \in \Sigma - \{\triangleright\}$, $q \in K$ such that $\delta(q, a) = (p, \leftarrow)$ for some $p \in K$, and for each $b \in \Sigma$, These tiles communicate head movement one square from left to right, while changing the state appropriately.



The Reduction τ

- Some additional tiles which correspond to the initial configuration $(s, \triangleright\#x)$, and tiles which only have $\#$ symbol on their tops.
- There is a tile with both upper and lower marking (y, a) for each symbol $a \in \Sigma$, effectively allowing the tiling to continue after the computation has halted at state y, but not if it halts at state n. (Here, state y denotes the accepting state and state n denotes the non-accepting state.)



The Reduction τ

4. f_0 is defined corresponding to the initial configuration $(s, \triangleright\#x)$.
    - $f_0(0)$ is the tile with upper marking $\triangleright$.
    - $f_0(1)$ is the tile with upper marking $(s, \#)$.
    - $f_0(i+1)$ is the tile with upper marking x_i for $i = 0, 1, \ldots, |x|$.
    - $f_0(i)$ is the tile with upper marking $\#$ for $i > |x| + 1$.
5. The sets H, V are defined such that two tiles can abut each other iff. the markings on adjacent sides are identical.

Bounded-Tiling problem is NP-Complete

Bounded-Tiling problem is NP-Complete

$x \in \mathcal{L} \Leftrightarrow \exists\ s \times s$ tiling f, extending f_0

PROOF: ($\Longleftarrow$) As $p(|x|) = s - 2$, the upper markings of the $(s-2)^{th}$ row must contain one of the halting states y or n. But since the $(s-1)^{th}$ row exists and there is no tile with lower markings (n, a) for any $a \in \Sigma$, there must be y in the upper markings of the $(s-2)^{th}$ row. Hence, the computation is accepting and $x \in \mathcal{L}$.

($\Longrightarrow$) There is an accepting computation by M, which can be simulated using the tiling system $\mathcal{D}$. Hence, there exists an $s \times s$ tiling extending f_0.

SATISFIABILITY is NP-Complete

Now, we must finally show that Bounded Tiling $\leq_P$ SATISFIABILITY.

- i.e. we must define a polynomial-time constructible function τ such that

$$\big(\mathcal{D} = (D, H, V), s, f_0\big) \xrightarrow{\quad\tau\quad} \text{Boolean Formula } x$$

$$\exists\ s \times s \text{ tiling } f,\ \text{extending } f_0 \Leftrightarrow x \in \text{SATISFIABILITY}$$

We construct the instance $\tau(\mathcal{D}, s, f_0)$ as follows,

- Variables will be $x_{m,n,d}$ for all $0 \leq m, n < s$ and $d \in D$.
  The intended correspondence is that: $x_{m,n,d}$ is $\top \Leftrightarrow f(m,n) = d$.

SATISFIABILITY is NP-Complete

- We now describe clauses which guarantee that f is a legal $s \times s$ tiling.
  1. $(x_{m,n,d_1} \vee \cdots \vee x_{m,n,d_k})$, for all $m, n < s$. Here, $D = \{d_1, \ldots, d_k\}$.
     These clauses ensure that each position has at least one tile.
  2. $(\overline{x_{m,n,d}} \vee \overline{x_{m,n,d'}})$ for all $m, n < s$ and $d \neq d' \in D$.
     These clauses ensure that each position has atmost one tile.
  3. $(x_{i,0,f_0(i)})$ for all $i < s$.
     These clauses force that $f(i, 0) = f_0$.
  4. $(\overline{x_{m,n,d}} \vee \overline{x_{m+1,n,d'}})$ for all $m < s - 1, n < s$ and $(d, d') \in D^2 - H$.
     These clause forbids any two tiles that are not horizontally compatible to be horizontally next to each other.
  5. $(\overline{x_{m,n,d}} \vee \overline{x_{m,n+1,d'}})$ for all $m < s, n < s - 1$ and $(d, d') \in D^2 - V$.
     These clause forbids any two tiles that are not vertically compatible to be vertically next to each other.

This completes the construction of clauses in $\tau(\mathcal{D}, s, f_0)$.

SATISFIABILITY is NP-Complete

$\exists\ s \times s$ tiling $f,$ extending $f_0 \Leftrightarrow \tau(\mathcal{D}, s, f_0) \in$ SATISFIABILITY

($\Longleftarrow$) Suppose that $\tau(\mathcal{D}, s, f_0)$ is satisfiable by the truth assignment T. Then because of clauses (i) and (ii), each position (m, n) has exactly one tile (say) $d_{m,n} \in D$. Hence, T represents a function $f : \{0, \dots, s-1\}^2 \to D$, defined by setting $f(m, n) = d_{m,n}$. It is easy to check that the clauses (iii) to (v) ensure that f is indeed a legal tiling on $\mathcal{D}$.

($\Longrightarrow$) Suppose that an $s \times s$ tiling f extending f_0 exists. Then define the following truth assignment $T : T(x_{m,n,d}) = \top$ iff $f(m, n) = d$. It is easy to check that T satisfies all clauses, and the proof is complete.

References

[Kar72] R. M. Karp. “Reducibility Among Combinatorial Problems”. In: *Complexity of Computer Computations* (1972), pp. 85–103.

[LP15] H. R. Lewis, C. H. Papadimitriou. *Elements of the Theory of Computation*. Pearson Education India, 2015. ISBN: 9789332549890.

[Sip97] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing, 1997. ISBN: 0-534-95097-3.

Thank You