1 Karp-Lipton Theorem

Dick Karp and Dick Lipton showed that \( \mathsf{NP} \) is unlikely to be contained in \( \mathsf{P/poly} \) [KL80]. Since \( \mathsf{P} \subset \mathsf{P/poly} \), if we could rule out the possibility that \( \mathsf{NP} \subset \mathsf{P/poly} \), then we would have separated \( \mathsf{P} \) and \( \mathsf{NP} \). Karp-Lipton theorem stirred a lot of effort trying to show circuit lower bounds. Indeed many successes followed, but unfortunately, we hit some barriers later on.

**Theorem 1.** If \( \mathsf{NP} \in \mathsf{P/poly} \), then \( \mathsf{PH} = \Sigma^P_2 \).

**Proof.** As shown in the last time, we just need to show that \( \Pi^P_2 \subset \Sigma^P_2 \) to conclude \( \mathsf{PH} = \Sigma^P_2 \). Let \( L \in \Pi^P_2 \). Then, there exists a poly-time TM \( M \) such that

\[
x \in L \iff \forall y \exists z, \ M(x, y, z) = 1, \text{ and } y \text{ and } z \text{ are poly-size}.
\]

All strings appear in this proof actually have polynomial length, and thus we omit this below. Due to the Cook-Levin theorem, the expression \( \exists z, M(x, y, z) = 1 \) can be converted in polynomial-time to a CNF formula \( \varphi \) such that

\[
x \in L \iff \forall y, \varphi(x, y) \in \mathsf{SAT}.
\]  

(1)

To show \( L \in \Sigma^P_2 \), we need to somehow change the quantifier in (1).

Our assumption is that \( \mathsf{NP} \in \mathsf{P/poly} \), namely \( \mathsf{SAT} \) has a polynomial-size circuit family \( \mathcal{C} = \{C_1, C_2, \cdots\} \).

For an input \( x \), \( |x| = n \), we give an \( \Sigma^P_2 \) algorithm to decide whether \( x \in L \). We first guess a circuit \( C \) for \( \mathsf{SAT} \). It may have two types of error: \( \varphi \in \mathsf{SAT} \) but \( C(\varphi) = 0 \) or \( \varphi \notin \mathsf{SAT} \) but \( C(\varphi) = 1 \). There is at least one correct guess, namely \( C_{p(n)} \) where \( p(n) = |\varphi(x, y)| \), that works correctly on \( \varphi(x, y) \) by assumption.

For any such guess \( C \), we construct a TM \( M' \) on input \( (C, \varphi) \) so that its error is “one-sided”. Recall that \( \mathsf{SAT} \) is self-reducible, namely given an oracle to the decision version, we can efficiently find a solution. \( M' \) takes \( C \) as a subroutine, and if \( C \) answers “Yes”, then \( M' \) proceeds to find a satisfying assignment by repeatedly asking \( C \). If there is any inconsistency from \( C \) (namely \( C \) says that a formula is satisfiable but both assignment of some variable \( x \) make it unsatisfiable), or the final assignment is not satisfying, then \( M' \) rejects. Otherwise \( M' \) accepts if \( C \) accepts. This machine \( M' \) will not accept a formula \( \varphi \notin \mathsf{SAT} \). It can only reject \( \varphi \in \mathsf{SAT} \) if \( C \) is an incorrect guess for \( \mathsf{SAT} \).

Now, we claim that the following:

\[
x \in L \iff \exists C \forall y, \ M'(C, \varphi(x, y)) = 1.
\]  

(2)
Comparing to (1), we have changed the order of quantifiers, and the right hand side of (2) can be easily converted to a $\Sigma^P_2$ expression (namely compute $\varphi(x, y)$ first). What we are left to do is to verify (2).

- If $x \in L$, then there is a correct guess $C_{p(n)}$ making the right hand side of (2) true.
- If $x \not\in L$, then by (1), there is $y_0$ such that $\varphi(x, y_0) \not\in \text{SAT}$. Recall that $M'$, whatever the guess $C$ is, never accepts $\varphi \not\in \text{SAT}$. Hence, the right hand side of (2) is false.

Why doesn’t the proof above work if we don’t construct $M'$ and simply use $C$?

## 2 Randomized computation

A great discovery in the theory of computation is the power of randomness. There are a lot of surprising randomized algorithms discovered since the 80s. Examples include, primality testing [Mil76, Rab80], polynomial identity testing [Zip79, Sch80], volume computation [DFK91], and many more. Here we showcase a simple randomized algorithm to test perfect matchings in a bipartite graph.

### 2.1 Lovász’s perfect matching algorithm

Let $G = (V, E)$ be a bipartite graph with two equal parts. Namely, $V = V_1 \cup V_2$ where $V_1$ and $V_2$ are disjoint, $|V_1| = |V_2|$, and $E \subseteq V_1 \times V_2$. A perfect matching (PM) is a subset $M \subseteq E$ of edges so that every vertex is adjacent to exactly one edge in $M$. Alternatively, a perfect matching $M$ can be thought of as a permutation $\sigma_M$ of $[n]$, so that $(i, \sigma_M(i)) \in E$.

**Name:** Bi-PM

**Input:** A bipartite graph $G$ with two equal parts.

**Output:** Does $G$ have a perfect matching?

Let $|V_1| = |V_2| = n$. Let $A$ be the $n$-by-$n$ bi-adjacency matrix of $G$. Rows of $A$ are indexed by vertices in $V_1$, and columns by vertices in $V_2$. $A_{u,v} = 1$ if $(u, v) \in E$ and $A_{u,v} = 0$ otherwise. The determinant of $A$ is defined as

$$
\det(A) = \sum_{\sigma \in S_n} (-1)^{\text{sgn}(\sigma)} \prod_{i=1}^{n} A_{i, \sigma(i)},
$$

where $S_n$ is the symmetric group whose elements are permutations of $[n]$. Notice that if $\det(A) \neq 0$, then we know that there must be some non-zero term, and thus $G$ has a perfect matching. However, if $\det(A) = 0$, then it might be because $G$ has no perfect matching, or non-zero terms cancel. We need to figure out which is the case.

To get around this, we associate a variable $x_{u,v}$ to each edge $(u, v)$. In other words, consider $A_X$, a $n$-by-$n$ matrix, where $A_{u,v} = x_{u,v}$ if $(u, v) \in E$, and 0 otherwise. Then
\[ \det(A_X) \text{ becomes a polynomial in variables } (x_{u,v})_{(u,v) \in E}. \text{ Notice that if } G \text{ has a perfect matching } M, \text{ then } \sigma_M \text{ induces a monomial in } \det(A_X), \text{ and } \det(A_X) \text{ is not identically zero. Otherwise if } G \text{ does not have a perfect matching, then } \det(A_X) \text{ is identically zero.} \]

Lovász’s algorithm [Lov79] is thus to simply test whether \( \det(A_X) \) is identically zero, by randomly assigning values to these variables. The algorithm works because of the following lemma, due to Zippel [Zip79] and Schwartz [Sch80].

**Lemma 2.** Let \( p(x_1, x_2, \ldots, x_n) \) be a non-zero polynomial of degree \( d \). Let \( S \) be a finite set of integers. Then, if \( a_1, \ldots, a_m \) are randomly chosen from \( S \) (with replacement), then

\[ \Pr[p(a_1, \ldots, a_m) \neq 0] \geq 1 - \frac{d}{|S|}. \]

The proof of Lemma 2 can be found in [AB09, Lemma A.36].

The degree of \( \det(A_X) \) is at most \( n \). To apply Lemma 2 to Bi-PM, we just randomly assign values from \([4n]\) to \( x_{i,j} \), and then evaluate the determinant. (The determinant has exponentially many terms, but it can be evaluated efficiently using, say, Gaussian elimination.) Under this random assignment, by Lemma 2, \( \det(A_X) \neq 0 \) with probability at least \( 1 - \frac{n}{4n} = 3/4 \) if \( G \) has a PM, and \( \det(A_X) = 0 \) with probability 1 if \( G \) does not have a PM. If we want to get better success probability, then we can simply run it again if \( \det(A_X) = 0 \). After \( t \) independent runs, the error probability goes down to \( 4^{-t} \).

The advantage of this algorithm is that \( \det(\cdot) \) is not only efficiently computable, it can even be computed efficiently in parallel. It is contained in a complexity class called \( \text{NC} \), which captures efficient parallel computation. For more details, see [AB09, Chapter 6.7.1].

You might have noticed that the following quantity, called the *permanent* of the matrix,

\[ \text{per}(A) := \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}, \]

does not have the annoying cancellation, and \( \text{per}(A) \) indeed counts the number of PM’s in \( G \). Unfortunately, \( \text{per}(A) \) is a quantity intractable to compute, and we will come back to that in counting complexity.

**Remark (Bibliographic).** In Theorem 1, Karp and Lipton [KL80] first proved collapse to the third level of \( \text{PH} \), and in their paper they attributes this stronger version to Mike Sipser. The proof given here uses an idea first noted by John Hopcroft. Relevant chapters are [AB09, Chapter 6.4, 7.1] and [Pap94, Chapter 11, 17].

**References**


