CS 354: Unfulfilled Algorithmic Fantasies April 17, 2019

Lecture 6: Complexity Class PWPP

Lecturer: Aviad Rubinstein *Scribe: Mingda Qiao*

1 Definition of PWPP

In the last lecture, we briefly introduced most of the following complexity classes, especially the complexity class PPAD and its complete problem of finding Nash equilibria.

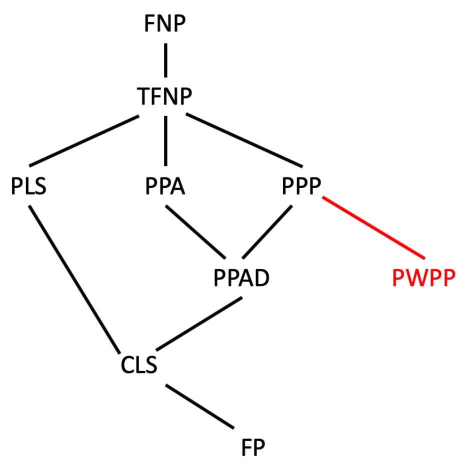


Figure 1: PWPP and its relation to other complexity classes.

In this lecture, we study PWPP (Polynomial Weak Pigeon Principle), a subclass of PPP, and also introduce a complete problem for PWPP that naturally arises in lattice-based cryptography.

Definition 1.1 ([1])**.** The class PWPP is the set of all problems that is polynomial-time reducible to the following problem: given a circuit $C : \{0,1\}^m \to \{0,1\}^n$ where $m > n$, find $x, x' \in \{0,1\}^m$ such that $x \neq x'$ and $C(x) = C(x')$.

By the weak pigeonhole principle, any function from a finite set to a smaller set has a collision, so the problem in the above definition always has a solution. The statement of the problem resembles the definition of collision-resistant hash functions.

2 Weak Constrained SIS Problem

We first define the Shortest Integer Solution (SIS) problem.

Definition 2.1 (SIS)**.** The Shortest Integer Solution problem is defined as follows:

- **Input:** Matrix $A \in \mathbb{Z}_q^{r \times t}$, where q is a power of two and $t > r \log q$.
- **Output:** Distinct vectors $x, x' \in \{0,1\}^t$ such that $Ax = Ax'$.

Again, since $2^t > q^r$, the weak pigeonhole principle guarantees the existence of a collision in the function $x \mapsto Ax$.

It is unknown whether SIS is PWPP-complete; instead, we focus on a constrained version of SIS defined as follows:

Definition 2.2 (Weak Constrained SIS (wc-SIS) [2])**.** The Weak Constrained SIS problem is defined as follows:

- **Input:** Matrices $A \in \mathbb{Z}_q^{r\times t}$ and $G \in \mathbb{Z}_q^{d\times t}$, where q is a power of two and $t > (r+d)\log q$. Moreover, G is guaranteed to be of the following form for $l = \log q$:

$$G = \begin{bmatrix} 2^0\ 2^1\ \cdots\ 2^{l-1} & \cdots & \cdots & \cdots & \cdots \\ 0 & 2^0\ 2^1\ \cdots\ 2^{l-1} & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 2^0\ 2^1\ \cdots\ 2^{l-1} & \cdots \end{bmatrix}$$

  In words, the i-th row of G always starts with $(i-1)l$ zeroes and then the first l powers of two, i.e., $2^0, 2^1, \ldots, 2^{l-1}$.

- **Output:** Distinct vectors $x, x' \in \{0,1\}^t$ such that $Ax = Ax'$ and $Gx = Gx' = 0$.

Note that the promise of matrix G guarantees that we can satisfy the constriant $Gx = 0$ in the following way: First, choose the last $(t - dl)$ bits of x arbitrarily. Then, using the fact that any integer between 0 and $2^l - 1$ can be uniquely written as the sum of a subset of $\{2^0, 2^1, \ldots, 2^{l-1}\}$, we can determine the first dl bits uniquely, l bits at a time.

3 wc-SIS is PWPP-complete

We prove the main result of this lecture: wc-SIS is PWPP-complete.

Lemma 3.1 ([2, Lemma 5.2])**.** *wc-SIS $\in$ PWPP.*

Proof. The goal is to define a circuit $C : \{0,1\}^{t-dl} \to \mathbb{Z}_q^r$, where the co-domain can be equivalently viewed as $\{0,1\}^{rl}$. Since the wc-SIS instance guarantees $t - dl > rl$, the resulting circuit will be a valid instance for the PWPP problem. Based on the previous observation, we can construct C using the following two parts:

- C_1 maps $x \in \{0,1\}^{t-dl}$ to the unqiue vector $\begin{bmatrix} u \\ x \end{bmatrix} \in \{0,1\}^t$ such that $G\begin{bmatrix} u \\ x \end{bmatrix} = 0$.

- C_2 simply maps $\begin{bmatrix} u \\ x \end{bmatrix} \in \{0,1\}^t$ to $A\begin{bmatrix} u \\ x \end{bmatrix} \in \mathbb{Z}_q^r$.

Let $C = C_2 \circ C_1$. If $C(x) = C(x')$ for $x \neq x'$, $\begin{bmatrix} u \\ x \end{bmatrix}$ and $\begin{bmatrix} u' \\ x' \end{bmatrix}$ give a solution to the original wc-SIS instance. □

In the following, we give a reduction in the other direction.

Theorem 3.2 ([2, Lemma 5.4])**.** *wc-SIS is PWPP-hard.*

Proof. We aim to design matrices A and G for given circuit C, such that for any vector $\begin{bmatrix} y \\ h \\ x \end{bmatrix}$, where y and h aim to simulate the outputs and the values on the hidden gates of C when feeding x as input:

- The constraint $G \begin{bmatrix} y \\ h \\ x \end{bmatrix} = 0$ verifies that the computation is correct.
- Matrix A maps $\begin{bmatrix} y \\ h \\ x \end{bmatrix}$ to y, so that we can check the collision in the output.

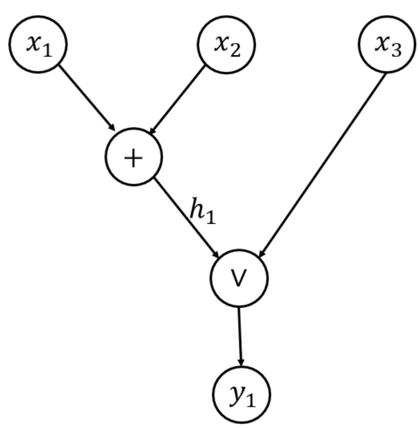
If the above two conditions hold, we know that a solution $A \begin{bmatrix} y \\ h \\ x \end{bmatrix} = A \begin{bmatrix} y' \\ h' \\ x' \end{bmatrix}$ to wc-SIS immediately implies a desired collision $C(x) = C(x')$.

For simplicity, we assume that circuit C only contains XOR and OR gates, denoted by $\oplus$ and $\vee$ respectively.[1] Moreover, we construct A and G over the ring Z_4 (i.e., $l = 2$).[2]

The following claim allows us to simulate XOR and OR gates using a single row in G:

Claim 3.3. *For* $a, b, c, d \in \{0, 1\}$*,* $a + 2b + c + d \equiv 0 \pmod 4$ *holds if and only if* $a = c \oplus d$ *and* $b = c \vee d$*.*

Using the above claim, we demonstrate how we transform a circuit into a matrix G by the following example:



[1] This is actually a cheat since $\{\oplus, \vee\}$ is not a complete set of gates. We will fix this in Homework 3.

[2] Technically this is not a cheat: showing that wc-SIS is hard for $l = 2$ suffices to prove the hardness of wc-SIS. However, proving the hardness of wc-SIS for larger values of l does require some more tricks.

For the above circuit, we first do a topological sort on the dependency graph of the gates (starting from the outputs) and obtain (y_1, h_1). Then we construct the following matrix G, where the rows corresponds to the gates (in the topological order), and each column correponds to either an input bit, an output bit of a hidden gate, or its "companion" output bit:

$$G = \begin{array}{c|ccccccc|} & \hat{y}_1 & y_1 & h_1 & \hat{h}_1 & x_1 & x_2 & x_3 \\ \hline y_1 & 1 & 2 & 1 & & & & 1 \\ h_1 & & & 1 & 2 & 1 & 1 & \\ \hline \end{array}$$

Then, the first line of G guarantees that $y_1 = h_1 \vee x_3$, and the second line guarantees that $h_1 = x_1 \oplus x_2$. Thus, the constraint $G \begin{bmatrix} \hat{y}_1 \\ y_1 \\ h_1 \\ \hat{h}_1 \\ x \end{bmatrix} = 0$ indeed verifies that $C(x) = y$.

Note that in the above construction, we need to compute $\hat{y}_1 = h_1 \oplus x_3$ and $\hat{h}_1 = x_1 \vee x_2$ even though they are never used as inputs. This is because Claim 3.3 only holds if we compute both operations at the same time. Moreover, we put $\hat{y}_1$ before y_1 and h_1 before $\hat{h}_1$ to make sure that the first two non-zero entries in each row are 1 and 2 in order, so that the promise of the wc-SIS instance is satisfied.

It remains to design the matrix A. A naïve approach is to choose A such that $A \begin{bmatrix} y \\ h \\ x \end{bmatrix} = y$. This requires A to have n rows (n is the number of output bits in C), which may result in an invalid wc-SIS instance (see the calculation in the next paragraph). Instead, we compress two bits in y into a single element in Z_4, which allows us to design an A with only $n/2$ rows.[3]

Finally, we show that the constructed instance (A, G) satisfies the assumption that $t > (r + d) \log q$ in the definition of wc-SIS. Let $|C|$ denote the number of gates in C and recall that m is the number of input bits. Then, both A and G have $t = 2|C| + m$ columns. Moreover, A has $r = n/2$ rows (thanks to the compression trick) and G has $d = |C|$ rows. Since the promise of the PWPP instance guarantees that $m > n$, we have

$$t = 2|C| + m > 2|C| + n = (r + d) \log q$$

as desired. □

References

[1] Emil Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016.

[2] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. In *Foundations of Computer Science (FOCS)*, pages 148–158, 2018.

[3] We can assume that n is even without loss of generality: if n is odd, we simply add a dummy input bit that directly goes to the output. This increases both n and m by 1 (so that the promise $m > n$ still holds) and the resulting instance is equivalent to the original one.