Pigeonhole Subset-Sum and related problems

KLINGELHOEFER Felix

August 2018

1 Introduction

During my internship, I did some bibliographic work on quantum computing (in particular quantum walks), but didn't make any contribution, and only acquired a limited understanding because the field is very complex. Therefore, I decided to focus my internship report on another problem I worked on, and in which I was able to find new state-of-the-art performance algorithms: the Pigeonhole Subset-Sum problem, and some variations around it.

2 The Pigeonhole Subset-Sum Problem

2.1 The Subset-Sum Problem

We first recall the usual Subset-Sum decision problem.

Subset-Sum
Input: A set $\{a_1, \dots, a_n\}$ of positive integers, a target number t.
Output: Is there a subset $S \subset \{1, \dots, n\}$ such that

$$\sum_{i \in S} a_i = t \ ?$$

Most of the results about Subset-Sum still hold if we are working with a modulus. We call this variant Modular Subset-Sum.

Modular Subset-Sum
Input: A set $\{a_1, \dots, a_n\}$ of positive integers, a target number t and a modulus p.
Output: Is there a subset $S \subset \{1, \dots, n\}$ such that

$$\sum_{i \in S} a_i \equiv t \mod p \ ?$$

Both the Subset-Sum and the Modular Subset-Sum problems can be solved exactly in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ on a classical computer (the $\widetilde{\mathcal{O}}$ notation ignores polynomial

factors in n). This can be done as follows. For each $S \subset \{1, \ldots, n\}$ denote $\Sigma(S) = \sum_{i \in S} a_i$. Compute first the sorted list of all the $\Sigma(S)$ for $S \subset \{1, \ldots, n/2\}$ (in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$). Then, compute $\Sigma(S)$ for each $S \subset \{n/2+1, \ldots, n\}$ and see if $t - \Sigma(S)$ occurs in the first list. This is a $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ algorithm for Subset-Sum.

On a quantum computer, the previous result can be improved to $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$ using Grover search. First, compute the sorted list of all the $\Sigma(S)$ for $S \subset \{1, \ldots, n/3\}$ (in time $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$). Then, we want to find $S \subset \{n/3+1, \ldots, n\}$ such that $t - \Sigma(S)$ occurs in the first list. Given $S \subset \{n/3+1, \ldots, n\}$, the last condition can be checked efficiently in time $\mathcal{O}(n)$ (the list is sorted). Consequently, using Grover search on the set $\{n/3+1, \ldots, n\}$ of size $\mathcal{O}\left(2^{2n/3}\right)$, we can solve Subset-Sum in time $\widetilde{\mathcal{O}}\left(\sqrt{2^{2n/3}}\right) = \widetilde{\mathcal{O}}\left(2^{n/3}\right)$.

There are more space-efficient variants of these algorithms but the asymptotic time complexities are the best known so far for *exact* algorithms. Moreover, finding explicitly c subsets that sum to the given target number t can be done at an extra cost $\widetilde{\mathcal{O}}(c)$ (if such subsets exist). See [BJLM13] for a review of these results, plus some heuristic algorithms.

2.2 The class PPP

Here we are interested in a variant of the Subset-Sum problem, which we call Pigeonhole Subset-Sum.

Pigeonhole Subset-Sum
Input: A set $\{a_1, \ldots, a_n\}$ of positive integers such that $\sum_{i=1}^{n} a_i < 2^n - 1$.
Output: Two distinct subsets $S_1, S_2 \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$$

There are 2^n subsets $S \subset \{1, \ldots, n\}$. Since they all verify $0 \leq \sum_{i \in S} a_i \leq 2^n - 2$ there must exist two distinct subsets S_1, S_2 that sum to the same value, according to the pigeonhole principle.

This is one of the most famous problems of the complexity class **PPP** defined by Papadimitriou in [Pap94]. **PPP** stands for Pigeonhole Principle Problem, which is the algorithmic class of problems that can be reduced to the Pigeonhole Circuit problem, for which there is both a weak and a strong version. THe first defines the class **PPP**.

Strong Pigeonhole Circuit
Input: A boolean circuit with n input bits and n output bits.
Output: Two distinct boolean vectors that map to the same output, or an input vector that maps to 0.

and the weak version defines the class **weak-PPP**.

Weak Pigeonhole Circuit
Input: A boolean circuit with n input bits and $n-1$ output bits.

Output: Two distinct boolean vectors that map to the same output.

The class **PPP** is a sub-class of True Functional Nondeterministic Polynomial, or **TFNP**, which is the class of problems that can be solved in polynomial time by a nondeterministic Turing machine, and for which a solution is guaranteed for every input, and at most polynomially larger. **TFNP** is naturally a subclass of **NP**, as its name indicates.

At this time, only two somewhat natural problems have been shown to be **PPP**-complete: BLICHFELDT and constrained-Short Integer Solution (cSIS), in [KS18]. We will not study these problems so we won't define them mathematically, but BLICHFELDT is searching for a point or a difference of two points that belong to a lattice, in a set of cardinality greater or equal to the volume of a lattice, in Z^n, while cSIS is a generalization of the Short Integer Solution problem.

During my internship, my advisor found a reduction from one version of the Pigeonhole Subset-Sum problems (Pigeonhole Abelian Group Subset-Sum) to the Pigeonhole Circuit problem, which would make it the third (and probably most natural) complete problem of the **PPP** class, which highlights its interest.

2.3 Related Problems

This is one of the main problems of our study:

Pigeonhole Abelian Group Subset-Sum
Input: A set $\{a_1, \ldots, a_n\}$ of positive integers, an Abelian group G such that $|G| < 2^n - 1$.
Output: Two distinct subsets $S_1, S_2 \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$$

It is a generalization of the modular version of the pigeonhole subset sum problem:

Pigeonhole Modular Subset-Sum
Input: A set $\{a_1, \ldots, a_n\}$ of positive integers, a modulus p such that $p < 2^n - 1$.
Output: Two distinct subsets $S_1, S_2 \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in S_1} a_i \equiv \sum_{i \in S_2} a_i \mod p$$

We will also study this one, because it allows us to show the ideas behind the generalization methods used for going from Pigeonhole Subset-Sum to Pigeonhole Abelian Group Subset-Sum while having much easier notations.

Our results

- a $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ classical algorithm for Pigeonhole Subset-Sum (Section 3.1)
- a $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ classical algorithm for Pigeonhole Modular Subset-Sum (Section 3.3)
- a $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ classical algorithm for Pigeonhole Abelian Group Subset-Sum (Section 3.4)
- a $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ quantum algorithm for Pigeonhole Subset-Sum (Section 3.5)

3 Algorithms for Pigeonhole Subset-Sum

The Pigeonhole Subset-Sum problem can be solved naively in time $\widetilde{\mathcal{O}}\left(2^n\right)$ without making use of the promise $\sum_{i=1}^{n} a_i < 2^n - 1$. Indeed, it suffices to compute the sum of each of the 2^n possible subsets of $\{a_1, \ldots, a_n\}$, sort these values, and search for a collision in the sorted list. Quantumly, this can be improved to $\widetilde{\mathcal{O}}\left(2^{2n/3}\right)$ by using the collision finding algorithm of Ambainis [Amb07].

Here we describe a non-trivial $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ classical algorithm for solving Pigeonhole Subset-Sum (using the promise $\sum_{i=1}^{n} a_i < 2^n - 1$). We then show how to adapt it to Pigeonhole Modular Subset-Sum and Pigeonhole Abelian Group Subset-Sum. The first proof was made by Antoine Joux and Yassine Hamoudi, but we will show it since it serves as a basis for our next algorithms. We then improve these results in the quantum setting with a $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ quantum algorithm for Pigeonhole Subset-Sum and some instances of Pigeonhole Modular Subset-Sum.

3.1 A $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ classical algorithm for Pigeonhole Subset-Sum

We prove the following result:

Theorem 1. *There is a classical algorithm for the* Pigeonhole Subset-Sum *problem that runs in time* $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

Proof. Given an input $\{a_1, \ldots, a_n\}$ of Pigeonhole Subset-Sum, for each $S \subset \{1, \ldots, n\}$ we let $\Sigma(S)$ be the sum $\sum_{i \in S} a_i$. Our goal is to find two distinct subsets S_1, S_2 of $\{1, \ldots, n\}$ such that $\Sigma(S_1) = \Sigma(S_2)$.

We denote by $[0], [1], \ldots [2^{n/2} - 1]$ the congruence classes modulo $2^{n/2}$. Each of these classes contains exactly $2^{n/2}$ numbers between 0 and $2^n - 2$, except the last class $[2^{n/2} - 1]$ that has only $2^{n/2} - 1$ numbers. Since all the 2^n subsets of $S \subset \{a_1, \ldots, a_n\}$ have a sum $\Sigma(S)$ between 0 and $2^n - 2$ there are two possible cases:

1. either there is some class $[\sigma]$ such that $\Sigma(S) \in [\sigma]$ for strictly more than $2^{n/2}$ subsets $S \subset \{1, \ldots, n\}$
2. or there are $2^{n/2}$ subsets $S \subset \{1, \ldots, n\}$ such that $\Sigma(S) \in [2^{n/2} - 1]$

Denote by $[\sigma]$ a class that verifies one of these two points. By definition, there are *strictly more* subsets S such that $\Sigma(S) \in [\sigma]$ than the number of elements between 0 and

$2^n - 2$ that belong to $[\sigma]$. However, for all $S \subset \{1, \ldots, n\}$ we have $\Sigma(S) \leq 2^n - 2$. Thus, there must be two subsets S_1, S_2 such that $\Sigma(S_1), \Sigma(S_2) \in [\sigma]$ and $\Sigma(S_1) = \Sigma(S_2)$.

Once we know such a value σ, we can make a call to Modular Subset-Sum to efficiently find a collision $\Sigma(S_1) = \Sigma(S_2)$ such that $\Sigma(S_1), \Sigma(S_2) \in [\sigma]$. Indeed, as explained in introduction, it is possible to find explicitly c subsets S that verifies $\Sigma(S) \equiv \sigma \mod 2^{n/2}$ at a total cost $\widetilde{\mathcal{O}}\left(2^{n/2} + c\right)$. Thus, if $\sigma < 2^{n/2} - 1$ and we know $2^{n/2} + 1$ subsets S that sum to σ modulo $2^{n/2}$ then we are guaranteed to find a collision among them. If $\sigma = 2^{n/2} - 1$ then it suffices to know $2^{n/2}$ subsets that sum to σ modulo $2^{n/2}$ to find a collision.

It remains to show how to find σ efficiently. We cannot compute $\Sigma(S) \mod 2^{n/2}$ for all S separately since there are 2^n such subsets. Instead, we are going to build an $(n+1) \times 2^{n/2}$ array T that has the following property:

$$T[i, j] = \left|\left\{S \subset \{1, \ldots, i\} : \Sigma(S) \equiv j \mod 2^{n/2}\right\}\right|$$

Given such an array, we can find in the last row a value σ such that $T[n, \sigma] > 2^{n/2}$, or $T[n, 2^{n/2} - 1] = 2^{n/2}$.

Finally, we show how to build the array T in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ with dynamic programming. First, we have $T[0, 0] = 1$ (the empty set sums to zero) and $T[0, j] = 0$ for $j > 0$. Then, remark that if $S \subset \{1, \ldots, i+1\}$ verifies $\Sigma(S) \equiv j \mod 2^{n/2}$, then either $S \subset \{1, \ldots, i\}$, or there exists $S' \subset \{1, \ldots, i\}$ such that $S = S' \cup \{a_{i+1}\}$ and $\Sigma(S') \equiv j - a_{i+1} \mod 2^{n/2}$. Thus, knowing a_{i+1} and $T[i, j]$ for all $j \in \{0, \ldots, 2^{n/2} - 1\}$, we can compute the next values $T[i+1, j]$ as follows:

$$T[i+1, j] = T[i, j] + T[i, j - a_{i+1} \mod 2^{n/2}]$$

Consequently, the $(i+1)^{th}$ row of T can be deduced from the i^{th} row and a_{i+1} in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$. The total computation time is $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$. □

Remark 2. This algorithm can be extended to Pigeonhole Modular Subset-Sum in the case where the modulus p is the factor of two numbers: $p = p_1 * p_2$. Indeed, any number congruent to $\sigma \mod p_1$ can only have p_2 different values mod p: $\{\sigma + k * p_1 : k \in [0, p_2 - 1]\}$, so we can use p_1 as modulus in the algorithm instead of $2^{n/2}$. The complexity is then $\widetilde{\mathcal{O}}\left(p_1 + p_2\right)$, and in particular if $p_1 \approx p_2 \approx 2^{n/2}$, the complexity is $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

3.2 A $\widetilde{\mathcal{O}}\left(3^{n/2}\right)$ classical algorithm for all pigeonhole problems

While it is no longer our best result for any problem, this was the first algorithm (better than the naive one) we found that worked for all cases of the Pigeonhole Modular Subset-Sum and Pigeonhole Abelian Group Subset-Sum problems.

Theorem 3. *There is a classical algorithm for the* Pigeonhole Modular Subset-Sum *and* Pigeonhole Abelian Group Subset-Sum *problems that runs in time* $\widetilde{\mathcal{O}}\left(3^{n/2}\right)$.

Proof. This algorithm is based on the classical algorithm for the Subset-Sum problem. Given an input consisting of n numbers, we partition them into two sets of (roughly) $n/2$ elements, that we call S_1 and S_2. We then explicitly determine the sets $T_1 = \{ \sum_{x \in S_1} x * i :$

$i \in \{-1;0;1\}\}$ and $T_2 = \{\sum_{x\in S_2} x * i : i \in \{-1;0;1\}\}$, and order them. We can then find a collision between the values in T_1 and those in T_2 that will give us a solution. Indeed, if two subsets that are equal can also be seen as one assignment of $\{-1;0;1\}$ as multipliers to each value, that equals 0: 1 if the element is in the first subset, -1 if it is in the second, and 0 if it is in neither. We need to point out that there is necessarily a disjoint solution, since there is a solution, and if a same value is in two distinct but equal subsets, then taking that value out of both they will remain distinct and equal.

The complexity of this algorithm is $\widetilde{\mathcal{O}}\left(3^{n/2}\right)$ because that is the size of both sets we compute. Ordering them only adds a polynomial factor (logarithmic in $3^{n/2}$), and finding a collision is then done in polynomial time of the lists' length.

This algorithm works for all versions of the Pigeonhole Subset-Sum problem because we only compare the final values when looking for a collision. We just need to add an ordering on the Abelian group version, but that does not make it more difficult.

Note that this algorithm does not use the promise: $\sum_{i\in S_1} a_i \equiv \sum_{i\in S_2} a_i \mod p$. Therefore, it would work for a more general Equal Subsets problem where no solution is guaranteed to exist. This is probably what makes it less efficient than our other algorithms, that all exploit the promise in order to achieve better results.

Finally, this algorithm can easily be extended to a quantum setting, as for the normal Subset-Sum problem, by computing one set for the $n/3$ and ordering it, and then searching among the second set that has assignments for the other $2n/3$ elements, for an element that has its opposite in the ordered list (can be checked in polynomial time). This gives us a quantum complexity of $\widetilde{\mathcal{O}}\left(3^{n/3}\right)$, which is still worse than the classical complexity of $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ we manage to achieve for all the problems.

□

3.3 Adaptation to Pigeonhole Modular Subset-Sum

We will adapt the previous algorithm to prove the following theorem:

Theorem 4. *There is a classical algorithm for the* Pigeonhole Modular Subset-Sum *problem that runs in time* $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

Proof. Our goal is the same as previously, except we only need to find two distinct subsets S_1, S_2 of $\{1,\dots,n\}$ such that $\Sigma(S_1) \equiv \Sigma(S_2) \mod p$.

In order to do so, we decide to compute an array giving the number of sets whose modular sum is in each of the (at most $2^{n/2}$) intervals of the form $[2^{n/2} * i, 2^{n/2} * (i+1) - 1]$ for $i \in [0, \lfloor p/2^{n/2} \rfloor]$. These intervals are of size $2^{n/2}$, except for the last one which is of size at most $2^{n/2}$ if there are $2^{n/2}$ intervals. More precisely, our goal is to get:

$$A[i,j] = \left| \left\{ S \subset \{1,\dots,i\} : \lfloor \sum_{k\in S} a_k / 2^{n/2} \rfloor \equiv j \mod p \right\} \right|$$

Given such an array, we could find in the last row a value σ such that $T[n,\sigma] > 2^{n/2}$, or $T[n, 2^{n/2}-1] = 2^{n/2}$, which would guarantee us a solution by finding a collision amongst the elements of that set, because of the size of the interval of values that they can take.

However, such an array can not be easily computed, so instead we build a slightly different array:

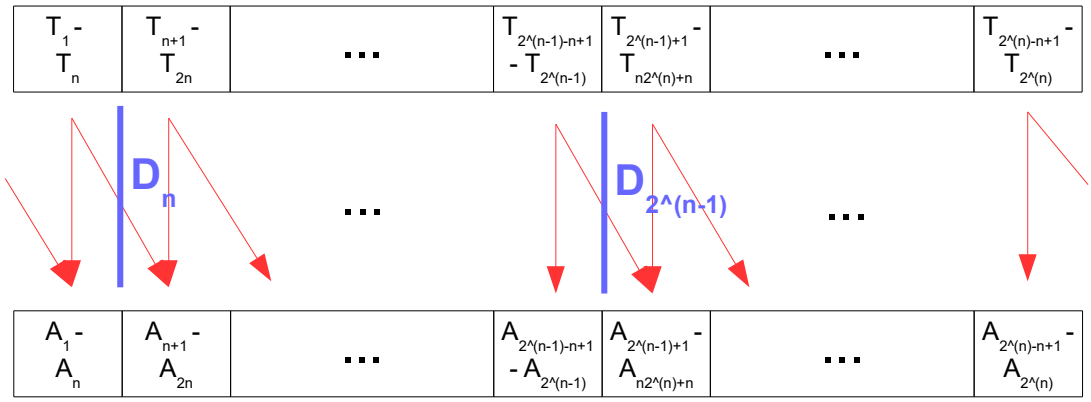

Figure 1: Relation between the different arrays used in our proof. The red arrows show where the elements from one group of cells go.The blue bars show what we need to determine in order to be able to do a dichotomy search on the larger side of the array A.

$$T[i,j] = \left| \left\{ S \subset \{1,\dots,i\} : \sum_{k \in S} \lfloor a_k / 2^{n/2} \rfloor \equiv j \mod p \right\} \right|$$

This array can be constructed the same way as the one in the previous subsection. We now observe that $\sigma \in T[n,j] \implies \exists k \in [0,n-1], \sigma \in T[n,j+k]$. Indeed, by adding rounded down divisions of elements by $2^{n/2}$, the rounded down division of the sum can only be off by at most the number of elements minus one. We will use this fact to find the right value of sigma for the array A by doing a dichotomy search on the array T, based on the following equality:

$$\sum_{j=x}^{y} A[n,j] = \sum_{j=x}^{y} T[n,j] + \sum_{j=x-n}^{x-1} D_x[n,j] - \sum_{j=y-n}^{y} D_{y+1}[n,j]$$

Where we define:

$$D_x[i,j] = \{S \subset \{1,\dots,i\} : S \in T[i,j]; \exists k \in \{x,...,x+n\}, S \in A[i,k]\}$$

Which basically comes down to saying that if we define arbitrary boundaries x and y, the number of elements between them in A is equal to the number of elements between them in T, plus the number of elements crossing the boundary x between T and A minus those crossing y between T and A. Since the elements can only "advance" by at most $n-1$ cells between T and A, we can compute those border crossings by explicitly computing all the elements of $2*n$ cells of T thanks to the SUBSET-SUM algorithm.

Once we created the array T as in Section 2.1, we can calculate $\sum_0^{2^{(n/2)-1}-1} A[n,j]$. When that is done, since there are a total of 2^n sets that we just split into two categories, we know that either $\sum_0^{2^{(n/2)-1}-1} A[n,j] > 2^{n/2}/2$ in which case we can find $\sigma \in \{0,...,2^{(n/2)-1}-1\}$ such that $A[n,\sigma] > 2^{n/2}$ or $\sum_{2^{(n/2)-1}}^{2^{n/2}-1} A[n,j] > (2^{n/2}/2) - 1$ and then we can find $\sigma \in \{2^{(n/2)-1},...,2^{n/2}-1\}$ with $A[n,\sigma] > 2^{n/2}$. This is the first step of the dichotomy, which we then need to repeat $\log_2 2^{n/2} = n/2$ times in order to find the final σ such that $A[n,\sigma] > 2^{n/2}$ (or $A[n,2^{n/2}-1] > 2^{n/2}-1$).

We now only need to find a collision in the sets of $A[n, \sigma]$ which as previously is done in $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$. To sum everything up, we start by computing T in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$, then make $n/2$ dichotomy steps which each have complexity $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$, before finding a collision in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$. This means the algorithm runs in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

□

Remark 5. If at some point during the calculations we find a $T[n, j] > n * 2^{n/2}$, we can directly determine $n * 2^{n/2} + 1$ elements of $T[n, j]$ and we will be guaranteed to find a collision, because they all lie in some $A[n, k]$ with $k \in \{j, ..., j + n\}$. This allows us to avoid computing all the elements of a T[n,j] which would be asymptotically larger than $2^{n/2}$ and lead to an a worse complexity.

3.4 Extension to Pigeonhole Abelian Group Subset-Sum

Using similar ideas to the previous subsection, we prove the following result:

Theorem 6. *There is a classical algorithm for the* Pigeonhole Abelian Group Subset-Sum *problem that runs in time* $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

Proof. Given a finitely generated abelian group G, the primary decomposition formulation states that G is isomorphic to a direct sum of primary cyclic groups. Since G is finite, let us consider G as $\mathbb{Z}_{p_1} \oplus \mathbb{Z}_{p_2} \oplus ... \oplus \mathbb{Z}_{p_k}$. We then find l such that $\prod_{i=1}^{l-1} p_i < 2^{n/2}$ and $\prod_{i=1}^{l} p_i > 2^{n/2}$ (if there is no such l, there are at most $2^{n/2}$ elements in G and we can solve the instance by computing the value of all subsets and finding a collision). For ease of notation, we will define the following:

$$q = \prod_{i=1}^{l-1} p_i$$

In the following a_i^k will denote the value of a_i on $\mathbb{Z}_{q_k}$.

The general idea is the same as for the modular version of the problem, but it is only done on one of the groups in the direct sum, which we have called $\mathbb{Z}_{p_k}$. For the other groups, the exact value is taken in the array, or it isn't considered until the collision is determined in one of the cells of the array. We will detail the operations with the correct formalism, but the algorithm should be understood as performing the same tasks as the previous one, but only on $\mathbb{Z}_{p_l}$.

As previously, we will have two arrays: T which we will compute, and A which gives us a collision if one cell has more than $2^{n/2}$ elements:

$$T[i, j] = \left| \left\{ S \subset \{1, \ldots, i\} : \sum_{m=1}^{l-1} [(\sum_{k \in S} a_k^m) * \tfrac{2^{n/2}}{q} * \prod_{m'=1}^{m-1} p_{m'}] + (\sum_{k \in S} \lfloor a_k^l * q/2^{n/2} \rfloor \mod p_l * q/2^{n/2}) = j \right\} \right|$$

$$A[i, j] = \left| \left\{ S \subset \{1, \ldots, i\} : \sum_{m=1}^{l-1} [(\sum_{k \in S} a_k^m) * \tfrac{2^{n/2}}{q} * \prod_{m'=1}^{m-1} p_{m'}] + (\lfloor \sum_{k \in S} a_k^l * q/2^{n/2} \rfloor \mod p_l * q/2^{n/2}) = j \right\} \right|$$

As previously, we will not compute all of A, but instead focus on finding the relevant sigma by dichotomy. Before doing so however, we will restrict the interval we are working on for j from $\{0, ..., 2^{n/2} - 1\}$ to $\{b, ..., b + 2^{n/2}/p - 1\}$ with b some multiple of $2^{n/2}/p$ such that $\sum_{j=b}^{b+2^{n/2}/p-1} T[n, j] >= p * 2^n$. This can be done because $\sum_{j=b}^{b+2^{n/2}/p-1} T[n, j] = \sum_{j=b}^{b+2^{n/2}/p-1} A[n, j]$, which is explained by the fact that the computations are exact on $\mathbb{Z}_{q_1} \oplus \mathbb{Z}_{q_2} \oplus ... \oplus \mathbb{Z}_{q_{l-1}}$. In what follows, additions and subtractions on j must be understood mod $2^{n/2}/p$ in the interval $\{b, ..., b + 2^{n/2}/p - 1\}$.

It still holds that for a given S we have $\sum_{k \in S} \lfloor a_k * p/2^{n/2} \rfloor = j$, then $\lfloor \sum_{k \in S} a_k * p/2^{n/2} \rfloor \in \{j, \ldots, j + n\}$. We use this fact to establish the following rule that serves as the basis of our dichotomy:

$$\sum_{j=x}^{y} A[n, j] = \sum_{j=x}^{y} T[n, j] + \sum_{j=x-n}^{x-1} D_x[n, j] - \sum_{j=y-n}^{y} D_{y+1}[n, j]$$

Where we define:

$$D_x[i, j] = \{S \subset \{1, \ldots, i\} : S \in T[i, j]; \exists k \in \{x, ..., x + n\}, S \in A[i, k]\}$$

$D_x[i, j]$ can be computed in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ since it suffices to find all the sets in T[i,j] which can be done in the same way as previously, and then sum their elements to decide whether they are in $D_x[i, j]$ or not.

We can now do the dichotomy as previously, int he interval $\{b, ..., b + 2^{n/2}/p - 1\}$.

Once that is done, it suffices to compute explicitly $A[n, \sigma]$ in order to find a collision. The complexity is the same as for the modular version because the same operations are done, so it is $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.

□

Remark 7. If at some point during the calculations we find a $T[n, j] > n * 2^{n/2}$, we can directly determine $n * 2^{n/2} + 1$ elements of $T[n, j]$ and we will be guaranteed to find a collision, because they all lie in some $A[n, k]$ with $k \in \{j, ..., j + n\}$. This allows us to avoid computing all the elements of a T[n,j] which would be asymptotically much larger than $2^{n/2}$ and lead to an a worse complexity.

3.5 A $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ quantum algorithm for Pigeonhole Subset-Sum

We prove the following result:

Theorem 8. *There is a quantum algorithm for the* Pigeonhole Subset-Sum *problem that runs in time* $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$.

Proof. We can compute in time $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ the same array T as in Section 3.1, except we use modulus $2^{2n/5}$ instead of modulus $2^{n/2}$. It will give us a value $\sigma \in \{0, \ldots, 2^{2n/5} - 1\}$ such that there exist $S_1, S_2 \subset \{1, \ldots, n\}$ with $\Sigma(S_1) \equiv \Sigma(S_2) \equiv \sigma \mod 2^{2n/5}$ and $\Sigma(S_1) = \Sigma(S_2)$. However, in the worst case, we need now to enumerate $2^{3n/5} + 1$ subsets S that sum to σ modulo $2^{2n/5}$ before having a collision.

We would like to apply Ambainis's collision finding algorithm [Amb07] on these $\approx 2^{3n/5}$ elements, so as to find a collision $\Sigma(S_1) = \Sigma(S_2)$ in time $\widetilde{\mathcal{O}}\left(\left(2^{3n/5}\right)^{2/3}\right) = \widetilde{\mathcal{O}}\left(2^{2n/5}\right)$. However, we have to ensure before that the cost of each query of the collision finding algorithm

is small. In other words, if we denote $\Omega_\sigma = \left\{S \subset \{1,\ldots,n\} : \Sigma(S) \equiv \sigma \mod 2^{2n/5}\right\} = \left\{S_1,\ldots,S_{|\Omega_\sigma|}\right\}$, we want to find an algorithm that given $I \in \{1,\ldots,|\Omega_\sigma|\}$ returns S_I in (say) polynomial time. If we make calls to Modular Subset-Sum, as in the proof of Theorem 1, it would not be efficient (finding one element in Ω_σ with this method has cost $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$ on a quantum computer). Instead, we are going to reuse the array T and "uncompute" the paths that led to $T[n,\sigma]$ in order to find the elements of Ω_σ. This is provided by Lemma 10 below. □

Definition 9. Given $\{a_1,\ldots,a_n\}$, we define a (lexicographic) total order $\prec$ over $\{1,\ldots,n\}$ as follows: for all $S_1 \neq S_2 \subseteq \{1,\ldots,n\}$, we have $S_1 \prec S_2$ if and only if $\max\{i : i \in (S_1 \cup S_2)\backslash(S_1 \cap S_2)\} \in S_2$.

Lemma 10. *Denote* $\Omega_\sigma = \left\{S \subset \{1,\ldots,n\} : \Sigma(S) \equiv \sigma \mod 2^{2n/5}\right\} = \left\{S_1,\ldots,S_{|\Omega_\sigma|}\right\}$ *where* $S_1 \prec \cdots \prec S_{|\Omega_\sigma|}$. *Given the array* T *computed in the proof of Theorem 12, and any* $I \in \{1,\ldots,|\Omega_\sigma|\}$, *we can compute* S_I *in time* $\mathcal{O}(n)$.

Proof. Starting from $T[n,\sigma]$ (that contains the total number of $S \subset \{1,\ldots,n\}$ that sum to σ modulo $2^{2n/5}$) and $\bar{S} = \varnothing$, we are going to reconstruct S_I by going backward ($i = n-1,\ldots,0$) in T. At each step, we will examine two elements in the i-th row of T and decide if we include a_i in $\bar{S}$ or not. At the end, we have $\bar{S} = S_I$. We describe the first step of the reconstruction:

1. Notice that:
   - $T[n-1,\sigma] = |\{S \in \Omega_\sigma : a_n \notin S\}|$ and $\{S \in \Omega_\sigma : a_n \notin S\} = \{S_1,\ldots,S_{T[n-1,\sigma]}\}$
   - $T[n-1,(\sigma - a_n \mod 2^{2n/5})] = |\{S \in \Omega_\sigma : a_n \in S\}|$ and $\{S \in \Omega_\sigma : a_n \in S\} = \{S_{T[n-1,\sigma]+1},\ldots,S_{|\Omega_\sigma|}\}$
2. If $I \leq T[n-1,\sigma]$ then keep $\bar{S}$ unchanged, and proceed to position $(n-1,\sigma)$ in T
3. If $I > T[n-1,\sigma]$ then set $\bar{S} = \bar{S} \cup \{a_n\}$, set $I = I - T[n-1,\sigma]$, and proceed to position $(n-1,(\sigma - a_n \mod 2^{2n/5}))$ in T

According to the definition of the $\prec$ order, this first step will correctly decide if a_n is in S_I or not. The change in the value of I is required for the next iterations (basically, we have removed from I the number of elements in Ω_σ that are $\prec S_I$ and can no longer appear in the path we took in T). The whole algorithm is described below. □

Input: T, σ, I
Output: S_I
$j = \sigma$
$\bar{S} = \varnothing$
for $i = n, \ldots, 1$ **do**
  if $I \leq T[i-1, j]$ **then**
    Do nothing
  else
    $\bar{S} = \bar{S} \cup \{a_i\}$
    $I = I - T[i-1, j]$
    $j = j - a_i \mod 2^{2n/5}$
Return S_I

The previous algorithm can be easily adapted to the Pigeonhole Modular Subset-Sum problem:

Theorem 11. *Given two numbers p_1, p_2, the* Pigeonhole Modular Subset-Sum *problem with modulus $p = p_1 p_2 < 2^n - 1$ can be solved in time $\widetilde{\mathcal{O}}\left(p_1 + (p_2)^{2/3}\right)$ on a quantum computer. In particular, if $p_1 = \widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ and $p_2 = \widetilde{\mathcal{O}}\left(2^{3n/5}\right)$ this is a $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ algorithm.*

Proof. We use the same algorithm as for Pigeonhole Subset-Sum. Indeed, if $p_1 | p$, then there are at most $p_2 = p/p_1$ possible values for an element congruent to a given number modulus p_1. So we simply fill the array using p_1 as the modulus, and then find a collision among the cell with the most elements using the Ambainis collision algorithm. □

3.6 A $\widetilde{\mathcal{O}}\left((4/3)^n\right)$ classical probabilistic algorithm for well-distributed Pigeonhole Subset-Sum and related problems

Since every pair of distinct and disjointed subsets of the input set can be matched uniquely to an assignment of $\{-1; 0; 1\}$ multipliers to each input element, there are a total of 3^n elements such pairs. Since every such pair has a total sum (sum of the elements in one set minus those in the other) in the integer interval $[-2^n, 2^n]$ (this comes from the promise) in the Pigeonhole Subset-Sum problem, we could expect that in a randomly distributed instance, roughly $3^n/2^n = (3/2)^n$ of these to be 0. This means that making the assumption of having $(3/2)^n$ pairs of equal, distinct and disjointed sets comes down to saying that the instance is well distributed (uniformly would suffice, for example).

We therefore prove the following result:

Theorem 12. *There is a probabilistic algorithm for the* Pigeonhole Subset-Sum *problem that runs in expected time $\widetilde{\mathcal{O}}\left((4/3)^n\right)$, given the assumption that there are approximately $(3/2)^n$ pairs of equal, distinct and disjointed sets.*

Proof. We start by randomly partitioning the set of inputs into two sets S_1 and S_2. Our main argument is that with constant probability, there is a solution such that its sum with only the elements of S_1 (which must be equal to the opposite of its sum on the

elements of S_2) is between $-(4/3)^n$ and $(4/3)^n$. This is justified by the random nature of our partitioning, which gives a random value to the value of each solution on S_1. With a random value on S_1 that falls in the interval $[-2^n, 2^n]$, each pair of distinct, disjointed and equal sets has a probability of $(4/3)^n/2^n) = (2/3)^n$ of falling within $[-(4/3)^n, (4/3)^n]$. Given that there are approximately $(3/2)^n$ pairs of equal, distinct and disjointed sets, we can expect roughly one to fall in the desired interval, which means there is a constant probability of finding a solution with sum on S_1 within $[-(4/3)^n, (4/3)^n]$.

We will use the argument above by computing all the subsets of S_1 and S_2 with value in $[-2^n, 2^n]$. In order to do so, we use a variant of the algorithm explained in 3.2. For both sets S_1 and S_2, we start by dividing them in two, and then compute the ordered list of sums on each half. Then, instead of finding collisions between both half-sets, we use the same method to find all value assignments with sum in $[-(4/3)^n, (4/3)^n]$. This has a complexity of $\widetilde{\mathcal{O}}\left((4/3)^n + 3^{n/4}\right) = \widetilde{\mathcal{O}}\left((4/3)^n\right)$ because each list has length $3^{n/4}$, but we might take up to $\widetilde{\mathcal{O}}\left((4/3)^n\right)$ values (if there are more than $2 * (4/3)^n$, we can directly find a solution and need not look at the remaining assignments).

□

4 Open problems

Here are some open problems about Pigeonhole Subset-Sum:

1. Find a $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$ quantum algorithm for Pigeonhole Subset-Sum (using Grover search?).
2. Find a $\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$ quantum algorithm for Pigeonhole Modular Subset-Sum or Pigeonhole Abelian Group Subset-Sum.
3. Find a same-size reduction from Pigeonhole Modular Subset-Sum or Pigeonhole Abelian Group Subset-Sum to Pigeonhole Subset-Sum.
4. Find a reduction from Pigeonhole Subset-Sum to Subset-Sum.

Attempt for open problem 1

We can start as in Sections 3.1 and 3.5, but with modulus $2^{n/3}$. It will give us a value $\sigma \in \{0, \ldots, 2^{n/3} - 1\}$ such that there exist $S_1, S_2 \subset \{1, \ldots, n\}$ with $\Sigma(S_1) \equiv \Sigma(S_2) \equiv \sigma \mod 2^{n/3}$ and $\Sigma(S_1) = \Sigma(S_2)$. In the worst case, we need to find $2^{2n/3} + 1$ subsets S that sum to σ modulo $2^{n/3}$ before having a collision.

We can look for a quadratic quantum speed-up to perform this last step, as it is the case for Grover search (which is used to solve Subset-Sum in $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$). Since we know that $\Sigma(S_1) = \Sigma(S_2) = \sigma + k \cdot 2^{n/3}$ for some $0 \leq k < 2^{2n/3}$, then Grover search can find k with $\widetilde{\mathcal{O}}\left(\sqrt{2^{2n/3}}\right) = \widetilde{\mathcal{O}}\left(2^{n/3}\right)$ queries. However, each of these queries corresponds to a call to Subset-Sum with target $t = \sigma + k \cdot 2^{n/3}$, which requires times $\widetilde{\mathcal{O}}\left(2^{n/3}\right)$ on a quantum computer. Thus, the total time complexity of this approach is $\widetilde{\mathcal{O}}\left(2^{n/3} \cdot 2^{n/3}\right) = \widetilde{\mathcal{O}}\left(2^{2n/3}\right)$.

Attempt for open problem 2

We can try combining the quantum algorithm for Pigeonhole Subset-Sum with the generalization methods used in 3.3 and 3.4, but it doesn't work. The crucial step that cannot be used in the quantum case is the explicit computation of the cells of the array T by using the algorithm for Subset-Sum. Indeed, in the quantum setting, each cell of the array contains $2^{3n/5}$ elements, and while a collision can be found in complexity $2^{2n/5}$, it is not possible to find out how many sets cross the "borders" for our dichotomy between the two arrays in complexity less than $2^{3n/5}$.

We also explored the possibility of using a quantum counting algorithm. The best known method however, developed by Gilles Brassard in [BHT98] has a complexity $\widetilde{\mathcal{O}}\left(\sqrt{N * c}\right)$ where N is the number of elements and c the final count. In this case, it would give us $\widetilde{\mathcal{O}}\left(\sqrt{2^{3n/5} * 2^{3n/5}}\right) = \widetilde{\mathcal{O}}\left(2^{3n/5}\right)$ in the worst case if we want to count the number of sets crossing a boundary, which is not satisfactory.

Attempt for open problem 3

Finding a size-preserving reduction from Pigeonhole Modular Subset-Sum or Pigeonhole Abelian Group Subset-Sum to Pigeonhole Subset-Sum would be of interest because it would allow us to use the quantum algorithm for Pigeonhole Subset-Sum on instances of these different problems, and therefore give us a better quantum complexity for these (and solve problem 2).

Given an instance $(\{a_1, \ldots, a_n\},\ p)$ of Pigeonhole Modular Subset-Sum, the most "natural" reduction to Pigeonhole Subset-Sum would be to consider the input $\{a_1 \mod p, \ldots, a_n \mod p, 2p, 4p, \ldots, 2^{\log n}p\}$.

Indeed, given $S_1, S_2 \subset \{1, \ldots, n\}$ such that $\Sigma(S_1) \equiv \Sigma(S_2) \mod p$, it must be the case that $\sum_{i\in S_1}(a_i \mod p) = \sum_{i\in S_2}(a_i \mod p) + kp$ for $-n \leq k \leq n$. Reciprocally, if there are $S_1, S_2 \subset \{1, \ldots, n\}$ and $S_1', S_2' \subset \{0, 1, \ldots, \log n\}$ such that $S_1' \cap S_2' = \varnothing$ and $\sum_{i\in S_1}(a_i \mod p) + \sum_{j\in S_1'} 2^j p = \sum_{i\in S_2}(a_i \mod p) + \sum_{j\in S_2'} 2^j p$ then we must have $\Sigma(S_1) \equiv \Sigma(S_2) \mod p$ and $S_1 \neq S_2$.

Unfortunately, the instance $\{a_1 \mod p, \ldots, a_n \mod p, 2p, 4p, \ldots, np\}$ of Pigeonhole Subset-Sum contains $n + \log n + 1$ numbers whose total sum can be bounded above by $n(p-1) + 2np - 1 < 2^{n+\log n+\log 3} - 1$. This is not the promise required in the definition of Pigeonhole Subset-Sum (it must be bounded above by $2^{n+\log n+1} - 1$ since there are $n + \log n + 1$ elements).

Note that this analysis can be refined a little bit, but it doesn't improve the upper bound much. One solution to this problem would be to find one (or many) small element(s) that could be added to $\{a_1 \mod p, \ldots, a_n \mod p, 2p, 4p, \ldots, np\}$ without increasing the total sum much, and that do not interfere with the solutions.

Adding $\{(n+1)p, \ldots, 2^n p\}$ does give an instance of Pigeonhole Subset-Sum where the promise is fulfilled, but the size of the new instance is double that of the original, which does not allow any algorithmic gains by applying the quantum algorithm.

5 Conclusion

The Pigeonhole Subset-Sum problem is an interesting variation of the Subset-Sum problem that hasn't garnered much interest yet, but might become more relevant if indeed proven to be a **PPP**-complete problem. This allowed us to develop novel algorithms that improved upon the best known complexities in different settings.

The following table sums up the complexities of our best algorithms for each problem in every setting. Probabilistic also implies that the distribution assumption holds.

Setting	Pigeonhole Subset-Sum	Pigeonhole Modular / Abelian Group
Classical	$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$	$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$
Quantum	$\widetilde{\mathcal{O}}\left(2^{2n/5}\right)$	$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$
Probabilistic	$\widetilde{\mathcal{O}}\left((4/3)^n\right)$	$\widetilde{\mathcal{O}}\left((4/3)^n\right)$

The Pigeonhole Subset-Sum problem provided a lot of interesting algorithmic challenges for a seemingly simple problem. Making use of the promise seemed essential since that is what puts it in a sub-class of **NP** (which intuitively means it is algorithmically simpler), but proved to be a difficult task. It seems unlikely that polynomial algorithms will be developed for this problem, since it would imply that **PPP** and its subclass **PPAD** would all be in **P**, and there are some well-studied problems in both the two classes (for example searching for a Nash equilibrium or a three-colored point of the Sperner lemma) for which no one has found a polynomial time algorithm in decades. It is also highly improbable that there is a polynomial reduction from Pigeonhole Subset-Sum to an **NP**-complete problem, since it would imply **NP**=**co-NP** which is widely conjectured not to be true.

The two other versions of this problem: Pigeonhole Modular Subset-Sum and Pigeonhole Abelian Group Subset-Sum also provided their own set of challenges, and at least for the quantum setting, hurdles that we did not manage to overcome yet. The difficulty was in finding a way to adapt the counting array in order to find information with which we can exploit the promise. We chose to do this by shifting from a modulus categorization to a first-bit categorization. Our original algorithm actually also used the modulus, but it could "shift" between the array we computed and the one that provided information enabling us to find a solution. We chose to rather use the first bits of the value because it allowed for easier notations and formalism. The main idea was to correct the array that is computed and use a dichotomy to find the relevant cell.

Finally, we are hopeful that the quantum result for the classical algorithm can be expanded to the two other problems, or that a better ($\widetilde{\mathcal{O}}\left(2^{n/3}\right)$) algorithm is found that can work for all the problems. Indeed, the Pigeonhole Subset-Sum problem is very similar to the Subset-Sum problem, and in an "easier" complexity class, so we could expect to have algorithms with a complexity at least as good as those for Subset-Sum in all settings.

References

[Amb07] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[BHT98] Gilles Brassard, Peter HØyer, and Alain Tapp. Quantum counting. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, pages 820–831, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[BJLM13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. *Quantum Algorithms for the Subset-Sum Problem*, pages 16–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[KS18] Giorgos Zirdelis Katerina Sotiraki, Manolis Zampetakis. Ppp-completeness with connections to cryptography, 2018.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, June 1994.