A POLYNOMIAL-TIME REDUCTION OF THE 3-SAT TO THE QUADRATIC CONGRUENCE PROBLEM AND OTHER RELATED PROBLEMS

Renato Lui Geh
NUSP: 8536030

Computational Number Theory (MAC6927)
Prof. Sinai Robins
University of São Paulo

Abstract. In this term paper for MAC6927 — Computational Number Theory, we explore the history behind the quadratic congruence problem (QCP) and other related number theoric problems; show a polynomial-time reduction from the 3-SAT to the QCP quoting Adleman and Manders' 1978 theorem [MA78], implying that quadratic congruence is NP-complete; and show some solved and unsolved problems in Number Theory that are directly (or indirectly) related to the QCP problem and its membership in NP.

1. History

German mathematician David Hilbert published [Hil02] in 1902 a set of 23 unsolved problems in mathematics he deemed to be most important unanswered mathematical problems of the 20th century. Since then 9 of them have been solved (at the time the author is writing this line and as far as the author is aware), 9 are considered partially solved, three of them are unsolved and two of them are considered too vague. Unsolved problems include the infamous Riemann Hypothesis and an extension to the Kronecker-Weber Theorem. Amongst solved problems is the 10th Hilbert problem.

10th Hilbert Problem: Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: *to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.*

It was answered in 1970 by Matiyasevich [Mat70] to be impossible. The question now becomes, in which cases is there an algorithm for solvability and what is the complexity of such algorithms? In 1976, Adleman and Manders [MA76] partially answered these questions by proving that, for the quadratic cases, there exists an algorithm and it is NP-complete. In their proof, they also found that, through a slight modification in the final step of their proof, it was possible to answer the quadratic congruence problem. A cleaner version of this proof was published in 1978 by Adleman and Manders [MA78], proof we try to explain in this paper.

In this paper we focus on the second result of Adleman and Manders' 1978 article, but also show the main result, namely that the set of quadratic diophantine equations with natural numbers solutions is NP-complete. The proof is done through a polynomial-time reduction from the 3-SAT problem. This reduction implies that both problems covered in Adleman and Manders' article are in NP-complete.

In 1971, American mathematician Stephen Cook published "The Complexity of Theorem-proving Procedures" [Coo71], and in the next year, his fellow countryman Richard Karp published "Reducibility Among Combinatorial Problems" [Kar72]. The two articles introduced the concepts of P and NP classes, yielding the duo a Turing Award. Interestingly in 1973, on the other side of the Iron Curtain, Ukrainian Leonard Levin published [Lev73] in the USSR equivalent results to Cook's and Karp's, but considering search problems instead of decision problems (an interesting remark is that Levin did not receive a Turing Award for his work, despite having achieved equivalent results). Both works resulted in the following statement: that any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the problem of satisfiability of a Boolean formula, i.e. the SAT problem. Additionally, if there exists a deterministic polynomial time algorithm for solving SAT, then every NP problem can be solved by a deterministic polynomial time algorithm.

This independent, parallel work from opposite parts of the world, ideologically and geographically, gave rise to what is considered one of the most important open questions in theoretical computer science, the P vs NP problem.

- Satisfiability (SAT)
  - 0–1 integer programming
  - Clique
    - Set packing
    - Vertex cover
      - Set covering
      - Feedback node set
      - Feedback arc set
      - Directed Hamiltonian cycle
        - Undirected Hamiltonian cycle
  - Satisfiability with at most 3 literals per clause (3-SAT)
    - Chromatic number
      - Clique cover
      - Exact cover
        - Hitting set
        - Steiner set
        - 3-dimensional matching
        - Knapsack
          - Job sequencing
          - Partition
            - Max cut

In his 1972 paper, Karp also published a list of 21 NP-complete problems in which he showed reductions implying its membership in the NP-complete class. The list above is Karp's 21 NP-complete problems, where the nesting indicates the direction of the reduction. For instance, the exact cover problem was reduced to the knapsack problem, chromatic number was reduced to exact cover, 3-SAT was reduced to exact cover, and the SAT was reduced to the 3-SAT problem.

From this we know that the 3-SAT is NP-complete. Not only that, any problem that reduces 3-SAT is also in NP. In the next sections we will provide a brief review on polynomial-time reductions, present proper definitions on the QCP and 3-SAT, and prove the reduction. This section provided a brief historical overview, but we do not restrict historical remarks to only this section. When relevant, we provide short historical anecdotes on the subject.

2. Brief Review on Complexity Theory

In this section we define decision problems, the P and NP complexity classes and all the tools we need to prove a polynomial-time reduction. We give a shallow definition of the 3-SAT problem and give other examples of NP-complete problems.

Before we prove reductions, we first need to properly define the concepts we are going to use. We shall define a Problem as a question that takes a set of objects as input and returns another set of objects as output. There are many types of problems: decision problems, where the output is restricted to yes or no (or true or false) answers; optimization problems, where the answer is given by a particular element of a set such that such an object optimizes certain criteria; and search problems, which is when the answer is an element of the set of output answers.

We can give a format for problems:

Problem: vector sorting

Input $n \in \mathbb{N}$ and a vector $A[1..n]$ with n integers
Output An ordered vector
Sorting the vector A in increasing order.

It is natural to conclude that problems have a certain complexity attached to them. Furthermore, we can provide several algorithms that take the same input and yield the same output. In this case in particular we know that sorting a vector is $\mathcal{O}(n \ln n)$ at best. Examples of algorithms for the above problem are InsertSort, MergeSort and BubbleSort, with each one having different worst case complexities. We next show a few examples of different types of problems:

Search problem: greatest common divisor

Input $a, b \neq 0 \in \mathbb{N}$
Output $d|a$, $d|b$, and for all $d'|a$, $d'|b$ we have $d' \leq d$
Finding the $\gcd(a, b)$

The Euclidean algorithm is an example of an algorithm that solves the gcd problem, with $\mathcal{O}(\ln(\max\{a, b\}))$.

Optimization problem: longest common subsequence

Input Two strings $X[1..m]$ and $Y[1..n]$
Output A string
Finding the longest common subsequence in X and Y.

This classic computer science problem turns out to be $\Theta(m \cdot n)$. Following Cook and Karp's, we treat only the case for decision problems. One might think this severly restricts the generality of complexity classes, but it is easy to see that one can treat optimization and search problems as special cases of decision problems. We do this by restricting these problems into their decision problem subproblems:

Decision problem: greatest common divisor

Input $a, b \neq 0 \in \mathbb{N}$ and $k \in \mathbb{N}$
Output Boolean value
Is $\gcd(a, b) = k$ true?

Decision problem: longest common subsequence

Input Two strings $X[1..m]$ and $Y[1..n]$
Output Boolean value
Is there an LCS of X and Y such that its length $\geq k$?

We simply added a k restriction on the input and modified the question so that the answer is a yes or no question.

For the next part we assume Turing machines to be already defined, as we wish to keep this section brief, as the title says. We consider an algorithm a series of steps that can be performed by a Turing machine. A problem is solvable in polynomial time if there exists an algorithm that takes a polynomial number of steps on the size of the instance. We define an instance as a particular input of a problem. In the gcd decision problem, we could take the tuple $(253, 37, 1)$ as an instance of the input. Both the Euclidean algorithm and the optimal 2-dimension LCS algorithm run in polynomial time on the size of the instance. However, the general case of the LCS algorithm is $\mathcal{O}(2^{n_1})$, where n_i is the length of the i-th string, and thus is not polynomial.

Definition 2.1. *The P class is the set of all decision problems that can be solved by polynomial (on the size of the instance) time algorithms.*

Let Π be a Problem. We say that Π admits a polynomial verifier $\mathcal{V}$ for a YES answer if there exists a polynomial algorithm that takes an instance I of Π and an object $\mathcal{C}$ such that the size of $\mathcal{C}$ is polynomial and returns YES for some $\mathcal{C}$ if the answer $\Pi(I)$ is YES and NO for all $\mathcal{C}$ if $\Pi(I)$ is NO.

In other words, $\mathcal{V}$ takes an instance I and checks whether such an instance is a true answer to the problem. We call the object $\mathcal{C}$ a polynomial certificate of problem Π.

Analogally, a polynomial verifier for a NO answer takes a polynomial certificate $\mathcal{C}$ and returns NO if the answer $\Pi(I)$ is NO and YES for all $\mathcal{C}$ if $\Pi(I)$ is YES.

Problem: Hamiltonian cycle

Input A graph $G = (V, E)$

Output Boolean value

Is there a Hamiltonian cycle, i.e. a path $p = (e_1, e_2, \ldots, e_k)$ s.t. $\forall v \in V$, v is visited by p exactly once, and p is a cycle?

Consider the Hamiltonian cycle problem above. In the image below, the polynomial certificate is the red path, and the polynomial verifier is an algorithm that checks whether the instance given is an actual solution to the problem. It is easy to see that a polynomial verifier for the Hamiltonian cycle is an algorithm that checks whether the set of edges in $\mathcal{C}$ traverse all the vertices and that the edges form a cycle.

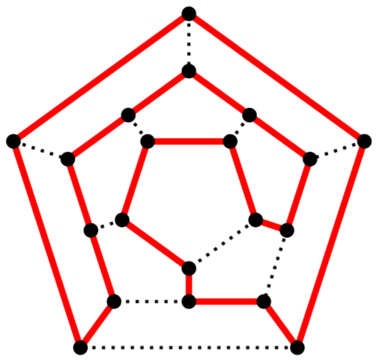

FIGURE 1. The red path is a polynomial certificate for the Hamiltonian cycle problem. Source: Wikipedia.

Definition 2.2. *The NP class is the set of all decision problems that admit a polynomial verifier for the YES answer.*

Definition 2.3. *The co-NP class is the set of all decision problems that admit a polynomial verifier for the NO answer.*

Note how every problem in P is already in NP and co-NP, since the algorithm that solves the problem could be used as a verifier for both the YES and NO answer.

Let Π and Π' be problems. A polynomial-time reduction from Π to Π' is an algorithm that solves Π using an algorithm that solves Π' as a subroutine, and that, excluding its subroutine, is polynomial on the size of its instance. We denote such a reduction by $\Pi \leq_p \Pi'$ if there exists a polynomial reduction, from Π to Π'.

Problem: SAT

Input A Boolean formula with n variables written in CNF, e.g. $(x_1 \vee x_2 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee x_5) \wedge \cdots$

Output Boolean value

Is there a valuation $\nu : \{x_1, x_2, \dots, x_n\} \to \{\text{false}, \text{true}\}$ such that the input formula evaluates to true?

Problem: 3-SAT

Input A Boolean formula with n variables written in CNF with at most 3 literals in each clause, e.g. $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_2 \vee \overline{x}_1 \vee x_5) \wedge \cdots$

Output Boolean value

Is there a valuation $\nu : \{x_1, x_2, \dots, x_n\} \to \{\text{false}, \text{true}\}$ such that the input formula evaluates to true?

We can prove that SAT $\leq_p$ 3-SAT by turning the clauses in the SAT problem into 3-SAT clauses [Kar72]. Recall Karp's 23 NP-complete list of problems. Each nesting means that the nested item Π' and its parent item Π obey $\Pi \leq_p \Pi'$. The SAT problem, can also be reduced to the Hamiltonian cycle problem by turning edges and vertices into SAT clauses. The vertex cover problem can also be reduced to the Hamiltonian cycle problem [Kar72].

Definition 2.4. *A problem Π in NP is NP-complete if $\forall\ \Pi' \in$ NP, $\Pi' \leq_p \Pi$.*

From the Cook-Levin Theorem [Coo71; Lev73], we know that the SAT problem is NP-complete. So if $\Pi \leq_p \Pi'$, and Π is NP-complete, then Π' is also NP-complete. So showing that a problem Π' is NP-complete consists of the following steps:

(1) Take $\Pi' \in$ NP,
(2) Take Π NP-complete,
(3) Show that $\Pi \leq_p \Pi'$.

Definition 2.5. *A problem Π is NP-hard if the existence of a polynomial algorithm for Π implies on $P = NP$.*

So every NP-complete problem is NP-hard, but an NP-hard problem is not necessarily in NP.

We wish to show that the 3-SAT problem can be reduced to the quadratic congruence problem, proving that QCP is NP-complete.

3. The Reduction

In this section we first define the Quadratic Congruence Problem (QCP), state the second theorem in [MA76], quote the algorithm used in the article that will be used in the proof, and then we show a complete and detailed proof of the reduction.

We first define the QCP as a decision problem.

Problem: quadratic congruence

Input $\alpha, \beta, \gamma \in \mathbb{Z}$
Output Boolean value
Is there a solution $x \in \mathbb{Z}$ where

$$x^2 \equiv \alpha \mod \beta$$

and such that $0 \leq x \leq \gamma$?

It turns out that the QCP is NP-complete, as we shall prove later. One might think that the QCP's complexity difficulty lies on the factorization of β. However, even when the full factorization of β is given, as the reduction shows, it is still NP-complete.

Another interesting note regarding the QCP problem is that, finding a solution x without the second restriction (i.e. no upper bound on x, $\gamma = \infty$) is solvable in polynomial time if we are given β's factorization. Furthermore, if we assume the Extended Riemann Hypothesis to be true, the problem is also solvable in polynomial time when β is prime [GJ79]. Additionally, the problem is trivially solvable in pseudo-polynomial time, meaning given a nondeterministic Turing machine, one can solve by running a "guess a solution and check whether it is a correct" algorithm in polynomial time [MA78].

We now state the theorem that covers the reduction.

Theorem 1 (Adleman and Manders, 1976)**.** *The (problem of accepting the) set of quadratic congruences (in a standard encoding)*

$$x^2 \equiv \alpha \mod \beta$$

with solutions $x \in \omega$ satisfying

$$0 \leq x \leq \gamma; \quad \alpha, \beta, \gamma \in \omega$$

is NP-complete.

It is easy to prove QCP's membership in NP. One could design an algorithm that first checks whether the input instance x obeys the condition $0 \leq x \leq \gamma$. If it passes, we simply check whether $x^2 \equiv \alpha \mod \beta$. This algorithm runs in $\mathcal{O}(1)$, and thus is polynomial. This is a polynomial verifier for the YES answer for the QCP, so QCP is in NP.

Before we prove Theorem 1, we first quote the algorithm used in the reduction from [MA78]. We use this algorithm in the proof extensively. In the article, Adleman and Manders recommend the reader to simply skim over the algorithm, and then to refer it back when reading the proof. We also recommend this, as the intent of the algorithm is not clear at first glance.

The algorithm itself is directly quoted from [MA78], but is also used in [MA76]. The former is a modified version of the latter. We chose the first to quote as it looked to be simpler and cleaner. Comments of the form [Comment: ...] are comments taken directly from the article.

3.1. The algorithm

On input ϕ, read ϕ and eliminate all duplicate conjuncts and those in which, for some variable x_i, both x_i and $\overline{x}_i$ occur. Count the l variables occurring in the remaining formula ϕ_R. Let

$$\Sigma = \{\sigma_1, \ldots, \sigma_m\}$$

be a standard enumeration of all possible disjunctive clauses, formed from $x_1, \ldots, x_l$ and their complements, with at most three literals per clause and no variable occurring twice or both complemented and uncomplemented in a clause. Compute

$$\tau_\phi = -\sum_{\sigma_l \in \phi_R} 8^j,$$

where, as below, we use $\in$ to denote the relation of an expressing *occurring* in another expression. [Comment: τ_ϕ is the only quantity computed which depends specifically on ϕ_R, rather than just on the number l of variables occurring in ϕ_R.] Compute:

$$\begin{aligned} f_i^+ &= \sum_{x_i \in \sigma_j} 8^j, & i &= 1, 2, \ldots, l, \\ f_i^- &= \sum_{\overline{x}_i \in \sigma_j} 8^j, & i &= 1, 2, \ldots, l. \end{aligned}$$

Set $n = 2m + l$ and compute $c_j, j = 0, \ldots, n$, as

$$\begin{aligned} & c_0 = 1 \\ & \left.\begin{aligned} c_j &= -\tfrac{1}{2}8^k, && \text{j=2k-1,} \\ c_j &= -8^k, && \text{j=2k,} \end{aligned}\right\} && j = 1, \ldots, 2m \\ & c_{2m+j} = \frac{1}{2}(f_j^+ - f_j^-), && j = 1, 2, \ldots, l, \end{aligned}$$

and

$$\tau = \tau_\phi + \sum_{j=0}^{n} c_j + \sum_{i=1}^{l} f_i^-$$

[Comment: At this point we have in fact obtained a knapsack problem $sum_{j=0}^{n} c_j \alpha_j = \tau$, $\alpha_j \in \{-1, +1\}$, which is solvable if and only if ϕ is satisfiable; moreover, for any value of $\alpha_j \in \{-1, +1\}$, $|\sum_{j=0}^{n} c_j\alpha_j - \tau| < 8^{m+1}$, so the knapsack problem is equivalent to $\sum_{j=0}^{n} c_j\alpha_j \equiv \tau \mod 8^{m+1}$, $\alpha_j \in \{-1, +1\}$. These assertions will become clear from the proof of correctness.]

Determine the first $n+1$ primes, $p_0, \ldots, p_n$, exceeding

$$(4(n+1)8^{m+1})^{\frac{1}{n-1}}$$

[This is in fact never exceeds 12, so we can set $p_0 = 13$.]

Determine parameters θ_j, $j = 0, 1, \ldots, n$, as: the least $\theta_j \in \omega$ such that

$$\begin{aligned} \theta_j &\equiv c_j \mod 8^{m+1}, \\ \theta_j &\equiv 0 \mod \prod_{i \neq j}^{n} p_i^{n+1}, \\ \theta_j &\not\equiv 0 \mod p_j. \end{aligned}$$

(Note: the following part of the algorithm was changed since Theorem 1 in article [MA78] is the quadratic Diophantine solutions problem and Theorem 2 is the QCP. In our paper, we refer to Theorem 1 as the QCP, and Theorem 2 as the Diophantine problem.)

Compute $H = \sum_{j=0}^{n} \theta_j$, $K = \prod_{j=0}^{n} p_j^{n+1}$ and output:

(a) for Theorem 1 (QCP):

$$x^2 \equiv (2 \cdot 8^{m+1} + K)^{-1} \cdot (K\tau^2 + 2 \cdot 8^{m+1}H^2) \mod 2 \cdot 8^{m+1} \cdot K, \quad 0 \leq x \leq H,$$

where, $(2 \cdot 8^{m+1} + K)^{-1}$ is the inverse of $(2 \cdot 8^{m+1} + K) \mod 2 \cdot 8^{m+1} \cdot K^n$.

(b) for Theorem 2 (Diophantine):

$$(K+1)^3 \cdot 2 \cdot 8^{m+1} \cdot (H^2 - x_1^2) + K(x_1^2 - \tau^2) - x_2 \cdot 2 \cdot 8^{m+1} \cdot K = 0.$$

3.2. The proof

In this subsection we show the proof of correctness of the algorithm (i.e. the proof of the reduction). We follow the proof given in [MA78], but with all passages explicitly explained, as well as a proof of Lemma 2 that was left to the reader.

We need to first show that the propositional formula ϕ is satisfiable if and only if the following expression is solvable.

$$\sum_{j=0}^{n} \theta_j \alpha_j \equiv \tau \mod 8^{m+1}, \qquad \alpha_j \in \{-1, +1\}, \quad j = 0, \dots, n$$

It is easy to see that the formula ϕ_R is satisfiable if and only if ϕ is, since ϕ_R is the result of taking out all irrelevant clauses. But ϕ_R is satisfiable if and only if there is a valuation $r : \{x_1, \dots, x_l\} \to \{0, 1\}$ such that for each disjunctive clause $\sigma_k \in \Sigma$

$$(1) \qquad 0 = R_k \begin{cases} = y_k - \sum_{x_i \in \sigma_k} r(x_i) - \sum_{\overline{x}_i \in \sigma_k} (1 - r(x_i)) + 1, & \text{if } \sigma_k \in \phi_R, (1) \\ = y_k - \sum_{x_i \in \sigma_k} r(x_i) - \sum_{\overline{x}_i \in \sigma_k} (1 - r(x_i)), & \text{if } \sigma_k \notin \phi_R. (2) \end{cases}$$

is solvable by $y_k \in \{0, 1, 2, 3\}$. We also add the constraint

$$0 = R_0 = \alpha_0 + 1, \qquad \alpha_0 \in \{0, 1\}$$

that does not influence the satisfiability of the system. We next prove that Equation 1 does in fact hold for the if and only if satisfiability.

Proof of Equation 1. We first consider case (1):

$$0 = y_k - \sum_{x_i \in \sigma_k} r(x_i) - \sum_{\overline{x}_i \in \sigma_k} (1 - r(x_i)) + 1, \quad \text{if } \sigma_k \in \phi_R$$

So y_k must be of the form:

$$y_k = \sum_{x_i \in \sigma_k} r(x_i) + \sum_{\overline{x}_i \in \sigma_k} (1 - r(x_i)) - 1$$

With no loss of generality, we can enumerate all possible clauses σ_k as follows

$$\begin{aligned}
(x_p \vee x_q \vee x_l): &\qquad y_k = r(x_p) + r(x_q) + r(x_l) - 1 \Rightarrow -1 \le y_k \le 2 \\
(\overline{x}_p \vee x_q \vee x_l): &\qquad y_k = r(x_q) + r(x_l) + 1 - r(x_p) - 1 \Rightarrow -1 \le y_k \le 2 \\
(\overline{x}_p \vee \overline{x}_q \vee x_l): &\qquad y_k = r(x_l) + 2 - r(x_p) - r(x_q) - 1 \Rightarrow -1 \le y_k \le 2 \\
(\overline{x}_p \vee \overline{x}_q \vee \overline{x}_l): &\qquad y_k = 3 - r(x_p) - r(x_q) - r(x_l) - 1 \Rightarrow -1 \le y_k \le 2
\end{aligned}$$

Note how in all cases above, the clause is satisfiable if and only if $y_k \neq -1$. In the first case, when $y_k = -1$, no literal x_i is valuated as true. In the second case, when $y_k \neq -1$, no literal x_i is valuated as true and $\overline{x}_p$ is valuated as false. The third case is similar, with x_l false, and $\overline{x}_p, \overline{x}_q$ false. For the last case, all $\overline{x}_i$ are set to false. When $y_k \neq -1$, there exists some literal being valuated as true, and therefore the clause σ_k is valuated as true.

For case (2), we do the same. In this case, y_k is of the form:

$$y_k = \sum_{x_i \in \sigma_k} r(x_i) + \sum_{\overline{x}_i \in \sigma_k} (1 - r(x_i))$$

Enumerating all literal combinations we have:

$$\begin{aligned}
(x_p \vee x_q \vee x_l): \qquad & y_k = r(x_p) + r(x_q) + r(x_l) \Rightarrow 0 \leq y_k \leq 3 \\
(\overline{x}_p \vee x_q \vee x_l): \qquad & y_k = r(x_q) + r(x_l) + 1 - r(x_p) \Rightarrow 0 \leq y_k \leq 3 \\
(\overline{x}_p \vee \overline{x}_q \vee x_l): \qquad & y_k = r(x_l) + 2 - r(x_p) - r(x_q) \Rightarrow 0 \leq y_k \leq 3 \\
(\overline{x}_p \vee \overline{x}_q \vee \overline{x}_l): \qquad & y_k = 3 - r(x_p) - r(x_q) - r(x_l) \Rightarrow 0 \leq y_k \leq 3
\end{aligned}$$

The analysis for this case goes analogally to the previous case, but with $y_k = 0$ being instatisfiable. Note how $y_k = 0$ is still a solution to the previous case. This inconsistency does not break the hypothesis, since in (2) we are considering clauses $\sigma_k \notin \phi_R$. That is, σ_k does not need to necessarily satisfy for ϕ_R to satisfy. However, for ϕ_R to be satisfiable, y_k needs to be in $\{0, 1, 2, 3\}$, as we showed. □

In order for ϕ_R to be satisfiable, every $\sigma_k \in \phi_R$ must be satisfiable. This is an if and only if condition on $y_k \in \{0, 1, 2, 3\}$, as we showed earlier. For any satisfiable ϕ_R, we have the following inequalities.

$$\begin{aligned}
-3 \leq R_k \leq 4, \qquad & k = 1, 2, \ldots, m \\
0 \leq R_0 \leq 2 \qquad &
\end{aligned}$$

From this, it follows that

$$R_k = 0, \quad k = 0, 1, \ldots, m \iff \sum_{k=0}^{m} R_k \cdot 8^k = 0 \tag{2}$$

and also that

$$\left| \sum_{k=0}^{m} R_k \cdot 8^k \right| < 8^{m+1} \tag{3}$$

This is true because

$$\begin{aligned}\left|\sum_{k=0}^{m} R_k \cdot 8^k\right| = \left|R_0 \cdot 8^0 + \sum_{k=1}^{m} R_k \cdot 8^k\right| &\leq 2 + \sum_{k=1}^{m} 4 \cdot 8^k \\ &\leq 2 + (4 \cdot 8 + 4 \cdot 8^2 + \cdots + 4 \cdot 8^m) \\ &\leq 2 + 8\,\underbrace{(4 + 4 \cdot 8 + 4 \cdot 8^2 + \cdots + 4 \cdot 8^{m-1})}_{\text{geometric series}} \\ &\leq 2 + 8\left(\frac{4}{7} \cdot 8^m - \frac{4}{7}\right) \\ &\leq 2 + \frac{4}{7} \cdot 8^{m+1} - 8 \cdot \frac{4}{7} \\ &< 8^{m+1}\end{aligned}$$

From (2) and (3), we have that

$$R_k = 0, \quad k = 0, 1, \ldots, m \iff \sum_{k=0}^{m} R_k \cdot 8^k \equiv 0 \mod 8^{m+1}. \tag{4}$$

We now wish to prove that we can reorganize the RHS of Equation 4 into the following expression by using the terms from the algorithm. That is, that

$$\sum_{k=0}^{m} R_k \cdot 8^k \equiv 0 \mod 8^{m+1} \iff \sum_{j=0}^{n} c_j \alpha_j \equiv \tau \mod 8^{m+1}. \tag{5}$$

Proof of Equation 5. We replace variables y_k and valuations $r(x_i)$ by $\{-1, +1\}$-valued variables:

$$\begin{aligned} y_k =& \frac{1}{2}\left[(1 - \alpha_{2k-1}) + 2 \cdot (1 - \alpha_{2k})\right] \\ r(x_i) =& \frac{1}{2}\left(1 - \alpha_{2m+i}\right)\end{aligned}$$

We call R'_k the result of this substitution. It follows that

$$R'_k = \frac{1}{2}[(1-\alpha_{2k-1})+2(1-\alpha_{2k})] - \sum_{x_i \in \sigma_k} \frac{1}{2}(1-\alpha_{2m+i}) - \sum_{\overline{x}_i \in \sigma_k} \left[1 - \frac{1}{2}(1 - \alpha_{2m+i})\right] + 1$$

when $\sigma_k \in \phi_R$. Plugging R'_k into (4) gives us

(6)
$$\sum_{k=0}^{m}\left(\frac{8^k}{2}-\frac{8^k}{2}\alpha_{2k-1}+8^k-8^k\alpha_{2k}-\sum_{x_i\in\sigma_k}\left(\frac{8^k}{2}-\frac{8^k}{2}\alpha_{2m+i}\right)-\sum_{\overline{x}_i\in\sigma_k}\left(\frac{8^k}{2}+\frac{8^k}{2}\alpha_{2m+i}\right)+8^k\right)\equiv 0 \mod 8^{m+1}$$

Recall the c_j, $j=0,1,\ldots,n$ variables computed earlier in the algorithm:

$$\left.\begin{aligned} c_{2k-1} &= -\tfrac{1}{2}8^k \\ c_{2k} &= -8^k \end{aligned}\right\} \qquad\qquad j=1,\ldots,2m$$
$$c_{2m+j}=\frac{1}{2}(f_j^+-f_j^-), \qquad\qquad j=1,2,\ldots,l$$

We wish to take equation (6) and make it look like

$$\sum_{j=0}^{n}c_j\alpha_j\equiv\tau \mod 8^{m+1},\quad \alpha_j\in\{-1,+1\}.$$

Replacing the values from (6) with the corresponding c_j values gives us

(7)
$$\sum_{k=0}^{m}\left(\underbrace{-c_{2k-1}+c_{2k-1}\alpha_{2k-1}-c_{2k}+c_{2k}\alpha_{2k}}_{(\star)}\underbrace{-\sum_{x_i\in\sigma_k}8^k\left(\frac{1}{2}-\frac{1}{2}\alpha_{2m+i}\right)-\sum_{\overline{x}_i\in\sigma_k}8^k\left(\frac{1}{2}+\frac{1}{2}\alpha_{2m+i}\right)}_{(\star\star)}+\underbrace{8^k}_{(\star\star\star)}\right)$$

on the LHS. We first turn our attention to $(\star)$. Notice how the pair $(2k-1,2k)$ over the range $[0,m]$ covers the range $[0,2m]$, meaning that

$$\sum_{k=0}^{m}(c_{2k-1}+c_{2k})=\sum_{i=0}^{2m}c_i.$$

In particular, separating the terms of the summation into two parts gives us

$$\overbrace{\sum_{k=0}^{m}(c_{2k-1}\alpha_{2k-1})-\sum_{k=0}^{m}(c_{2k-1}+c_{2k})}^{(\star)}=\sum_{j=0}^{2m}c_j\alpha_j-\sum_{j=0}^{2m}c_j$$

Recall from the algorithm that

$$f_i^+ = \sum_{x_i \in \sigma_j} 8^j, \qquad i = 1, 2, \ldots, l,$$
$$f_i^- = \sum_{\overline{x}_i \in \sigma_j} 8^j, \qquad i = 1, 2, \ldots, l.$$

Since f_j^+ and f_j^- are both counting the number of literals in ϕ_R, it is easy to see that $\sum_{x_i \in \phi_R} f_i^+ = \sum_{\overline{x}_i \in \phi_R} f_i^-$, since they range over all σ_k, and Σ is the set of all σ_k relevant clause permutations. So every literal x_i must appear the same number of times as $\overline{x}_i$. This means $\sum_{j=1}^{l} c_{2m+j} = 0$. Note how there exactly l variables in ϕ_R. From this we have that

$$\sum_{j=0}^{2m} c_j \alpha_j - \sum_{j=0}^{2m} c_j = \sum_{j=0}^{n} c_j \alpha_j - \sum_{j=0}^{n} c_j.$$

We now consider $(\star\star)$. It follows from $(\star\star)$ that

$$-\underbrace{\sum_{x_i \in \sigma_k} \frac{8^k}{2}}_{\frac{f_i^+}{2}} - \underbrace{\sum_{\overline{x}_i \in \sigma_k} \frac{8^k}{2}}_{\frac{f_i^-}{2}} - \frac{8^k}{2} \underbrace{\left(\sum_{x_i \in \sigma_k} \alpha_{2m+i} - \sum_{\overline{x}_i \in \sigma_k} \alpha_{2m+i} \right)}_{=0 \text{ when summing over } [0,m]} = -\sum_{\hat{x}_i \in \sigma_k} \frac{1}{2} \left(f_i^+ + f_i^- \right)$$

where we say $\hat{x}_i \in \sigma_k$ if $x_i \in \sigma_k$ or $\overline{x}_i \in \sigma_k$. The same argument from $(\star)$ can be used to show that

$$-\sum_{k=0}^{m} \sum_{\hat{x}_i \in \sigma_k} \frac{1}{2}(f_i^+ + f_i^-) = -\sum_{k=0}^{m} \sum_{\hat{x}_i \in \sigma_k} f_i^-$$

Now we have a summation over all disjunctive clauses σ_k, and for each k we are summing all occurrences in which $\hat{x}_i \in \sigma_k$. But this means we are summing all times in which each variable occurs in all clauses. The number of variables is l, so it follows that

$$-\sum_{k=0}^{m} \sum_{\hat{x}_i \in \sigma_k} f_i^- = -\sum_{i=1}^{l} f_i^-.$$

Let us now focus on $(\star\star\star)$. It is easy to see that

$$\sum_{k=0}^{m} 8^k = \sum_{\sigma_j \in \phi_R} 8^j = -\tau_\phi$$

Plugging the results (note that these results already take into account the outer summation $\sum_{k=0}^{m}$, and thus we ommit it) from $(\star), (\star\star), (\star\star\star)$ in Equation 7 gives us the following expression:

$$\underbrace{\sum_{j=0}^{n} c_j\alpha_j - \sum_{j=0}^{n} c_j}_{(\star)} - \underbrace{\sum_{i=1}^{l} f_i^-}_{(\star\star)} - \underbrace{\tau_\phi}_{(\star\star\star)} \equiv 0 \mod 8^{m+1}$$

Isolating the first summation of $(\star)$ yields

$$\sum_{j=0}^{n} c_j\alpha_j \equiv \tau_\phi + \sum_{j=0}^{n} c_j + \sum_{i=1}^{l} f_i^- \mod 8^{m+1}$$

The RHS of the congruence is exactly the definition of τ from the algorithm. Therefore:

$$\sum_{j=0}^{n} c_j\alpha_j \equiv \tau \mod 8^{m+1}, \quad \alpha_j \in \{-1,+1\}$$

□

From the definition of $\theta_j,\ j = 0,\dots,n$, it follows directly that the above is equivalent to

$$\sum_{j=0}^{n} \theta_j\alpha_j \equiv \tau \mod 8^{m+1}, \quad \alpha_j \in \{-1,+1\}$$

Lemma 1. *Let K and H be as in the algorithm. The general solution of the system*

$$0 \le |x| \le H, \qquad x \in \mathbb{Z} \tag{1}$$
$$(H+x)(H-x) \equiv 0 \mod K \tag{2}$$

is given by

$$x = \sum_{j=0}^{n} \alpha_j\theta_j, \quad \alpha_j \in \{-1,+1\}, \quad j = 0,1,\dots,n.$$

Proof. Let us recall the definitions of K and H from the algorithm:

$$K = \prod_{i=0}^{n} p_i^{n+1} \quad \text{, where } p_0 = 13 \text{ and } p_1, p_2, \ldots, p_n \text{ primes exceeding 12.}$$
$$H = \sum_{i=0}^{n} \theta_i$$

It is obvious that x is a solution to the system. We need to prove that these are the only solutions to the system. Let x be a solution to the system. Then

$$(H + x)(H - x) \equiv 0 \mod p_j^{n+1}, \qquad j = 0, 1, \ldots, n$$

We will prove by contradiction. Assume that for some j_0:

$$p_{j_0} | (H + x) \text{ and } p_{j_0} | (H - x)$$

Then $p_{j_0} | (H + x) + (H - x) = 2H$. But $p_{j_0} > 2$ and p_{j_0} is prime, so $p_{j_0} | H$. It then follows that $p_{j_0} | \sum_{j=0}^{n} \theta_j$. But from the definition of θ_j, we know that $p_{j_0} | \theta_j$ for all $j \neq j_0$, since we have

$$\theta_j \equiv 0 \mod \prod_{i \neq j}^{n} p_i^{n+1} \Rightarrow \prod_{i \neq j}^{n} p_i^{n+1} | \theta_j \Rightarrow p_i | \theta_j, \quad i \neq j$$

So it must be that $p_{j_0} | \theta_{j_0}$, but from θ_j's definition:

$$\theta_j \not\equiv 0 \mod p_j \Rightarrow p_j \nmid \theta_j$$

We end up in a contradiction. Therefore we can conclude that for all j, p_j^{n+1} divides either $(H + x)$ or $(H - x)$, but not both.

We define:

$$\alpha_j = \begin{cases} 1, & \text{if } p_j^{n+1} | (H - x) \\ -1, & \text{if } p_j^{n+1} | (H + x) \end{cases}$$
$$x' = \sum_{j=0}^{n} \alpha_j \theta_j$$

So we have $x' \equiv \alpha_j \theta_j \equiv \alpha_j H \equiv x \mod p_j^{n+1}$, and $x' \equiv x \mod p_j^{n+1}$, for all j, and so $x' \equiv x \mod K$.

$$\left.\begin{aligned} -H \leq x' \leq H \\ -H \leq x \leq H \end{aligned}\right\} \Rightarrow |x - x'| \leq 2H$$

But $p_j \geq (4(n+1)8^{m+1})^{\frac{1}{n+1}}$, and let

$$\lambda_j = \frac{\theta_j}{\sum_{\substack{i=0 \\ i \neq j}}^{n} p_i^{n+1}}$$

We have that $\lambda_j < 2 \cdot 8^{m+1}$ for each j. Each term in H is bounded by $K/2(n+1)$, so $2H < K$. This means $|x - x'| < K$, but $x' \equiv x \mod K$. Therefore, it must be that $x = x'$.

Thus x' is a solution and is the same as x, meaning any solution of the system is of that form. □

From Lemma 1, the condition

$$\sum_{j=0}^{n} \theta_j \alpha_j \equiv \tau \mod 8^{m+1}$$

is equivalent to the system:

$$\begin{aligned} &(i) && 0 \leq |x| \leq H, \quad x \in \mathbb{Z}, \\ &(ii) && x \equiv \tau \mod 8^{m+1}, \\ &(iii) && (H+x)(H-x) \equiv 0 \mod K \end{aligned} \tag{I}$$

Lemma 2. *Let τ be odd, $x \in \mathbb{Z}$, $k \geq 3$*

$$(\tau + x)(\tau - x) \equiv 0 \mod 2^{k+1} \iff \begin{aligned} &\textit{either } \tau + x \equiv 0 \mod 2^k \\ &\textit{or } \tau - x \equiv 0 \mod 2^k \end{aligned}$$

Proof. We first prove the converse ($\Longleftarrow$):

Assume it is possible for both cases to be true.

$$\begin{aligned} \tau + x \equiv 0 \mod 2^k &\Rightarrow \tau \equiv -x \mod 2^k \\ \tau - x \equiv 0 \mod 2^k &\Rightarrow \tau \equiv x \mod 2^k \end{aligned}$$

So $\tau \equiv x \equiv 0 \mod 2^k$. But τ is odd, leading to a contradiction. Therefore, only one of the two can be true at the same time. Consider the first case ($\tau + x \equiv 0 \mod 2^k$). So $(\tau + x) \equiv 0 \mod 2^k \Rightarrow (\tau + x)(\tau - x) \equiv 0 \mod 2^k$. In other words, $(\tau + x)(\tau - x) = 2^k \cdot c$. Multiplying 2 on both sides yields $2(\tau + x)(\tau - x) = 2^{k+1} \cdot c$. Converting it back to its congruence form gives us $2(\tau + x)(\tau - x) \equiv 0 \mod 2^{k+1}$.

But $\tau + x \equiv 0 \mod 2^k$, so we can substitute $(\tau + x)$ for 2^k, giving $2 \cdot 2^k \cdot c(\tau - x) = 2^{k+1}c(\tau - x) \equiv 0 \mod 2^{k+1}$. We do the same for $\tau - x \equiv 0 \mod 2^k$.

We now prove the direct implication ($\Rightarrow$):

$$\begin{aligned}(\tau + x)(\tau - x) \equiv 0 \mod 2^{k+1} \Rightarrow \text{ either } & \tau + x \equiv 0 \mod 2^k \\ \text{or } & \tau - x \equiv 0 \mod 2^k\end{aligned}$$

We can rewrite $(\tau + x)(\tau - x) \equiv 0 \mod 2^{k+1}$ as $(\tau + x)(\tau - x) = 2^{k+1} \cdot c = 2^k(2c) = (\tau + x)(\tau - x)$. So $(\tau + x)(\tau - x) \equiv 0 \mod 2^k$.

Let $a = \tau + x$ and $b = \tau - x$. We need to show that the cases:

$$\begin{aligned}&(1) \quad a \equiv 0 \mod 2^k \quad \text{and} \quad b \neq 0 \mod 2^k \\ &(2) \quad b \not\equiv 0 \mod 2^k \quad \text{and} \quad b \not\equiv 0 \mod 2^k\end{aligned}$$

are impossible if $a \cdot b \equiv 0 \mod 2^k$. We take case (1) into consideration first.

$$\begin{cases} \tau + x \equiv 0 \mod 2^k \\ \tau - x \equiv 0 \mod 2^k \end{cases}$$

But from the converse's proof, we know that this is impossible. Now we consider the second case.

$$\begin{cases} \tau + x \not\equiv 0 \mod 2^k \\ \tau - x \not\equiv 0 \mod 2^k \\ (\tau + x)(\tau - x) \equiv 0 \mod 2^k \end{cases} \Rightarrow \begin{cases} 2^k \nmid \tau + x \\ 2^k \nmid \tau - x \\ 2^k \nmid (\tau + x)(\tau - x) \end{cases}$$

Which is a contradiction. Therefore, the two cases are impossible. It is easy to see that all the remaining cases are the ones listed in the Lemma, and all are true. $\square$

Consider the system (I). The conditions of Lemma 2 apply to the system. Therefore system (I) is satisfiable if and only if system (II) is also satisfiable:

$$\begin{aligned} (i) \quad & 0 \leq |x| \leq H, \quad x \in \mathbb{Z}, \\ (ii) \quad & (\tau + x)(\tau - x) \equiv 0 \mod 2 \cdot 8^{m+1}, \qquad \text{(II)} \\ (iii) \quad & (H + x)(H - x) \equiv 0 \mod K \end{aligned}$$

From Lemma 2, (I) (ii) satisfies if and only if (II) (iii) satisfies, since if $\tau - x \equiv 0 \mod 8^{m+1}$, then it satisfies (I) (ii); and if $\tau + x \equiv 0 \mod 8^{m+1}$, then it satisfies (i) and (iii).

Note how:

$$\begin{aligned} (i) &\quad (\tau+x)(\tau-x) \equiv \tau^2 - x^2 \equiv 0 \mod 2\cdot 8^{m+1} \\ (ii) &\quad (H+x)(H-x) \equiv H^2 - x^2 \equiv 0 \mod K \end{aligned}$$

So we can restrict condition (i) to only positive integers:

$$0 \le x_1 \le H, \quad x_1 \in \mathbb{Z}$$

Also note that $\gcd(2\cdot 8^{m+1}, K) = 1$, since K is a product of odd primes (in fact greater than 12) and $2\cdot 8^{m+1}$ a power of 2. Because of that, we can join both conditions (ii) and (iii) from system (II).

$$\lambda_1 2\cdot 8^{m+1}(H^2 - x_1^2) + \lambda_2 K(\tau^2 - x_1^2) \equiv 0 \mod 2\cdot 8^{m+1}\cdot K$$

Where λ_1 and λ_2 are parameters we can freely choose subject to the following condition:

$$\gcd(\lambda_1, K) = \gcd(\lambda_2, 2\cdot 8^{m+1}) = 1, \quad \lambda_1, \lambda_2 \in \mathbb{Z}$$

We join these conditions into the new system (III):

$$\begin{aligned} (i) &\quad 0 \le x_1 \le H, \quad x_1 \in \mathbb{Z}, \\ (ii) &\quad \lambda_1 2\cdot 8^{m+1}(H^2 - x_1^2) + \lambda_2 K(\tau^2 - x_1^2) \equiv 0 \mod 2\cdot 8^{m+1}\cdot K, \qquad \text{(III)} \\ (iii) &\quad \gcd(\lambda_1, K) = \gcd(\lambda_2, 2\cdot 8^{m+1}) = 1, \quad \lambda_1, \lambda_2 \in \mathbb{Z} \end{aligned}$$

In fact, choosing λ_1 and λ_2 provides proofs for two problems: the quadratic Diophantine problem (which we will mention later) and for the QCP. Furthermore, conditions (III) (i), (ii) are satisfiable for any λ_1, λ_2 obeying (III) (iii) or for no λ_1 and λ_2.

We now finally provide a proof for Theorem 1.

Proof of Theorem 1. Choose $\lambda_1 = \lambda_2 = 1$. It obviously satisfies (III) (iii). We rewrite (III) (ii) with these values:

$$\begin{aligned} &2\cdot 8^{m+1}(H^2 - x_1^2) + K(\tau^2 - x_1^2) \equiv 0 \mod 2\cdot 8^{m+1}\cdot K \\ &2\cdot 8^{m+1}H^2 - 2\cdot 8^{m+1}x_1^2 + K\tau^2 - x_1^2 K \equiv 0 \mod 2\cdot 8^{m+1}\cdot K \\ &K\tau^2 + 2\cdot 8^{m+1}\cdot H^2 \equiv (2\cdot 8^{m+1} + K)\cdot x_1^2 \mod 2\cdot 8^{m+1}\cdot K \end{aligned}$$

Note how $\gcd(2\cdot 8^{m+1} + K, 2\cdot 8^{m+1}\cdot K) = 1$, and as such there exists an inverse $(2\cdot 8^{m+1} + K)^{-1}$. Multiplying the inverse on both sides, yields:

$$(2 \cdot 8^{m+1} + K)^{-1} \cdot (2 \cdot 8^{m+1} + K) x_1^2 \equiv (2 \cdot 8^{m+1} + K)^{-1} (K\tau^2 + 2 \cdot 8^{m+1} \cdot H^2) \mod 2 \cdot 8^{m+1} \cdot K$$

Rewriting the expression above gives us:

$$x_1^2 \equiv (2 \cdot 8^{m+1} + K)^{-1} (K\tau^2 + 2 \cdot 8^{m+1} \cdot H^2) \mod 2 \cdot 8^{m+1} \cdot K$$

Note how this is exactly the output of the algorithm for the QCP. Call $\alpha = (2 \cdot 8^{m+1} + K)^{-1} (K\tau^2 + 2 \cdot 8^{m+1} \cdot H^2)$ and $\beta = 2 \cdot 8^{m+1} \cdot K$. Rename x_1 as x. Note how we now have:

$$x^2 \equiv \alpha \mod \beta$$

Renaming H as γ gives us the restriction

$$0 \leq x^2 \leq \gamma$$

where $\alpha, \beta, \gamma \in \omega$. But this is exactly the theorem. So the QCP is satisfiable if and only if system (III) is satisfiable. But (III) is satisfiable if and only if the propositional formula (i.e. 3-SAT) is satisfiable. From this we conclude that QCP is NP-complete. □

4. Related problems

Manders and Adleman proved in both [MA76; MA78] that not only the QCP is NP-complete, but also the following:

Theorem 2. *The (problem of accepting the) set of Diophantine equations (in a standard binary encoding) of the form*

$$\alpha x_1^2 + \beta x_2 - \gamma = 0; \quad \alpha, \beta, \gamma \in \omega,$$

which have natural-number solutions x_1, x_2 is NP-complete.

The proof of this theorem starts exactly as the reduction we showed, with the only difference being in the choice of λ_1 and λ_2. We leave it to the reader to prove this (take $\lambda_1 = (K + 1)^3$ and $\lambda_2 = -1$).

References

[Coo71] Stephen Cook. “The Complexity of Theorem-proving Procedures”. In: *STOC ’71 Proceedings of the third annual ACM symposium on Theory of computing* (1971).

[GJ79] Michael Garey and David Johnson. *Computers and Tractability: A Guide to the Theory of NP-completeness.* New York, NY, USA: W. H. Freeman & Co., 1979. ISBN: 0716710447.

[Hil02] David Hilbert. “Mathematical problems”. In: *Bulletin of the New York Mathematical Society* 10 (1902).

[Kar72] Richard Karp. “Reducibility Among Combinatorial Problems”. In: *Complexity of Computer Computations* (1972).

[Lev73] Leonard Levin. “Universal Sequential Search Problems”. In: *Probl. Peredachi Inf.* 9 (1973).

[MA76] Kenneth Manders and Leonard Adleman. “NP-complete Decision Problems for Quadratic Polynomials”. In: *STOC ’76 Proceedings of the eighth annual ACM symposium on Theory of computing* (1976).

[MA78] Kenneth Manders and Leonard Adleman. “NP-Complete Decision Problems for Binary Quadratics”. In: *Journal of Computer and System Sciences* 16 (1978).

[Mat70] Yuri Matiyasevich. “Diofantovost’perechislimykh mnozhestv (Enumerable sets are diophantine)”. In: *Doklady Akademii Nauk SSSR* (1970).