E0-224: Computational Complexity Theory

Manish Kumar (21044)
M.Tech., Quantum Technology

Assignment-I

September 17, 2023

Problem 1

- (a) Show that QUADEQ is NP-complete.
- (b) 2-SAT is in P

Solution. Part (a):

Strategies :

• QUADEQ is in NP

• QUADEQ is NP-Hard. We demonstrate 3-SAT reduces to QUADEQ. (We assume 3-SAT is known to be NP-complete, as discussed in the class.)

QUADEQ is in NP: Polynomial length certificate is tuple $U = \{u_i, ..., u_n\}$ over field $\mathbb{Z}_2$. The verification algorithm plugs U into the m-system of quadratic equations and tests if they are simultaneously satisfied. It consists of $\mathcal{O}(n^2)$ multiplications and additions per equation. The overall scaling is $\mathcal{O}(m \cdot n^2)$.

Algorithm for Karp reduction of 3-SAT to QUADEQ: Take any 3-SAT associated CNF formula, for example $(x_1 \vee x_2 \vee \neg x_3) \wedge (...)$. The goal is to convert the problem of finding a satisfiable assignment into an equivalent problem of finding zeros of a system of multivariate quadratic equations(MQE). Let's assume there are m clauses. We employ the following strategies for each clause:

(Step- I) Replace $x_i \to z_i$ and $\neg x_i \to (1 - z_i)$. Multiply each new element (i.e. $z_i, 1 - z_i$) with a free variable, say $a \in \mathbb{Z}_2$. Replace logical OR with the addition operation. Finally, equate it with one.

(Step-II) An example could be $(x_1 \vee x_2 \vee \neg x_3) \to az_1 + bz_2 + c(1 - z_3) = 1$. This implies the satisfiability of each clause is equivalent to the existence of zeros of the quadratic equation.

(Step- III) We repeat this for all m clauses to get the m-system of MQE.

Working or the reduction: Suppose an oracle solves MQE. Then, we can also find a satisfying assignment of the associated formula via the above reduction. Suppose the oracle gives the solution set $\{z_i\}$. We map each z_i to x_i as per step-I. This assignment of $\{x_i\}$ is bound to make each clause True.

• If an oracle provides a satisfying assignment to CNF, we can also find the root of the MQE via the above reduction. Suppose the oracle gives the assignment $\{x_i\}$. We map each x_i to z_i and so on. We can find the value of free variables $(a, b, ...)$ by solving the system of linear equations in poly-time (Gauss Method).

Note: The use of Oracle is just to demonstrate how to use this poly-time reduction to find the solution to one problem from the other.

The reduction is poly-time: Reducing all m clauses to m system of MQE requires $\mathcal{O}(m)$ change of variables. About the same number of addition and multiplication are required, too.

Conclusion: $3SAT \leq_p QUADEQ$ and $QUADEQ$ is in NP. In fact, we showed reductions in both ways.

Part (b):

A poly-time algorithm for 2-SAT :

Let us take an example of a CNF with n variables and m clauses: $(x_0 \vee x_2) \wedge (x_0 \vee \neg x_3) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4)$...

- This is satisfiable if each clause evaluates to be true for at least one variable assignment.

• **Step-I**: Choose a variable x_i and assign $x_i \to T$ in all the clauses. Drop all the clauses (x_i) and $(x_i \vee x_j)$. Change $(\neg x_i \vee x_j) \to x_j$.

- After this assignment, check if clause $(\neg x_i)$ appears. If so, then declare the current trial FAILURE. Otherwise, focus on the generated single literal clause, like (x_j). Assign $(x_j) \to T$ to all such single literal clauses. This step requires $\mathcal{O}(m)$ clause evaluations.

• **Step-II**: In the case of FAILURE, reassign the current variable, say $(x_i \to F)$. Drop all the clauses $(\neg x_i)$ and $(\neg x_i \vee x_j)$. Change $(x_i \vee x_j) \to x_j$.

- After this assignment, check if clause (x_i) appears. If so, declare the CNF is UNSATISFIABLE, as neither T nor F works for x_i. Otherwise, focus on the generated single literal clause, like (x_j). Assign $(x_j) \to T$ to them.

• **Step-III**: If either of the two steps works for x_i, then we know its assignment. Now, CNF must have fewer variables and clauses at this stage. We repeat the above steps by choosing any other existing variables. If the algorithm runs without any FAILURE, it implies the given CNF has a satisfiable assignment.

- If we get an empty formula at any iteration, choose either T or F for the current variable (of that iteration). We have n variables. In the worst case, both steps (I and II) have to be computed for each variable. Hence, the total runtime scales as $\mathcal{O}(m \cdot n)$.

Problem 2

Prove that SUBSET SUM is NP-complete.

Solution. Strategies :
• SUBSET SUM is in NP
• SUBSET SUM is NP-Hard. As 3-SAT reduces to SUBSET SUM. (We assume 3-SAT is known to be NP-complete, as discussed in the class.)

SUBSET is in NP: Let the given set be $A = \{a_1, ..., a_n\}$. Polynomial length certificate is a subset $S = \{a_i\} \in A$.
- The verification algorithm checks if elements of the set S belong to set A. A trivial way is to pick an element of S and check if it belongs to A. Repeat it for all the elements in S. It scales as $\mathcal{O}(|S| \cdot |A|)$. - If true, it checks if they add up to T. This addition can be done in poly-time.

Algorithm for Karp reduction of 3-SAT to SUBSET SUM:
Input of SUBSET SUM is a set of integers and a target integer. In contrast, SAT is about the Boolean formula. The algorithm below shows the reduction. We take a concrete example of this reduction, which could be easily generalized to other cases.

• Take 3-SAT: $(\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3)$. Construct a reduction table using the below algorithm.
(Table on next page)

- Details of the table.
• Column indices are x_1 *to* x_l and C_1 *to* C_k . In our case, $l = 3$ and $k = 2$
• Row indices are $x_1, \neg x_2,$ ..., $x_l, \neg x_l, C_1^{'}, C_1^{''}$ ... $C_k^{'}$, $C_k^{''}$. The Last row contains T.

- Filling of the table:
• In column x_i, input 1 if the corresponding row has $x_i, \neg x_i$ *and* T.
• In column C_i, input 1 or 0 to mark the presence or absence of $x_i, \neg x_i$ in the clause C_j. Also, input 1 if the corresponding row has $C_j^{'}, C_j^{''}$. While the row corresponding to T is filled with an Integer of concern. We take 3 as an example.

- Inference from the table: We interpret the table as below:
• Take each table row (excluding the last) representing an Integer belonging to the original set A. $x_1 = 10001$, $\neg x_1 = 10010$, $x_2 = 01010$, $\neg x_2 = 01001$, $x_1 = 00101$, $\neg x_3 = 00100$, $C_1^{'} = C_1^{''} = 00010$, $C_2^{'} = C_2^{''} = 00001$.
• The last row is the target integer T. The last row is 11133.
• Now, we use the predictive power of the table. Given a valid combination of rows that adds up to T, we can use those rows to find an assignment of the Boolean variables that

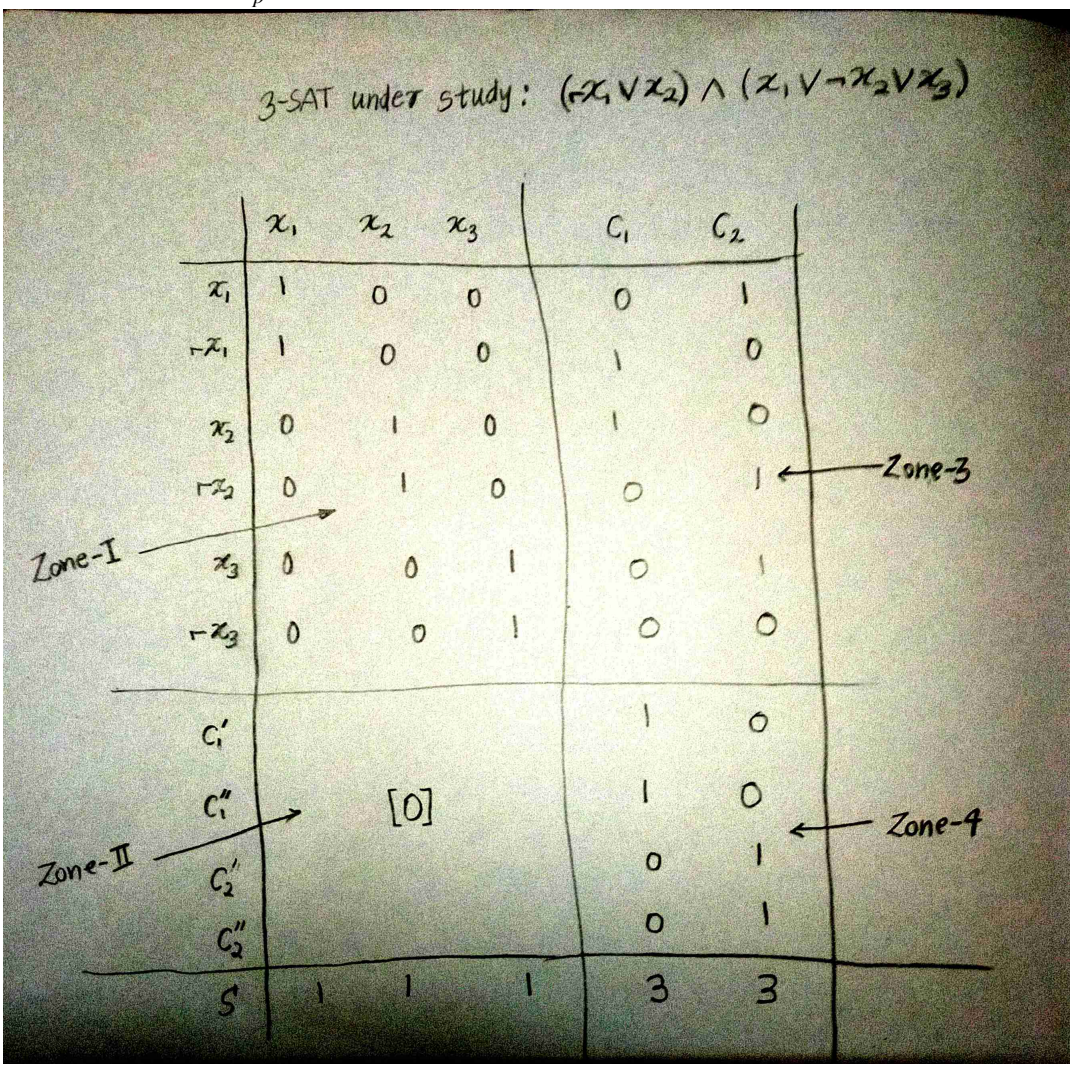
satisfy the formula.

• Since adding row corresponding to $\neg x_1,\ \neg x_2,\ x_3,\ C_1^{'},\ C_1^{''},\ and\ C_2^{''}$ results in $T = 11133$.

• It predicts $\neg x_1 = 1,\ \neg x_2 = 1,\ x_3 = 1$. Indeed this is a valid assignment satisfying the given formula.

• The argument holds for the opposite direction too. From a satisfying assignment for the given 3-SAT formula, the rows corresponding to those variables shall add up to T.

Conclusion: $3SAT \leq_p$ *SUBSET SUM* and *SUBSET SUM* is in *NP*.



Problem 3

The existence of OWF $\implies$ The existence of a Language $L_f \in$ (NP $\cap$ co-NP $\cap \neg$P).

Solution.

Given:

• There exists a TM to compute a certain $f(x)$ in poly-time. But there exists no such TM for $f^{-1}(x)$. The language $L_f := \{(x, y) : f^{-1}(x) < y\}$

Strategies:

• L_f is in NP

• L_f is in co-NP

• L_f is not in P

Approach:

L_f is in NP :

- There exists an efficient algorithm to verify the poly-time certificate for the L_f. The certificate is the tuple (x, y). The verification algorithm is as follows:
- A function having an inverse is strictly monotonic. Hence, $f^{-1}(x) < y \implies x < f(y)$ or $x > f(y)$. This can be checked by evaluating $f(x)$ at any two points x_1 and x_2.
- Let's assume it is monotonically increasing. It means $f^{-1}(x) < y \implies x < f(y)$. Now check for condition $x < f(y)$ for the given certificate (x, y). This is a poly-time process per the definition of $f(x)$.

L_f is in co – NP :

- An efficient algorithm exists to verify the poly-time certificate and verifier for the $\overline{L_f} = \{0, 1\}^*/L_f$. We find the dual problem of L_f as $\overline{L_f} := \{(x, y) : f^{-1}(x) \geq y\}$.
- This dual problem L_f has an efficient certificate and verifier. The proof technique is similar to the one given for the case of L_f.

L_f is not in P :

- This can be inferred based on the assumption mentioned in the question. There exists no efficient algorithm for computing $f^{-1}(x)$.
- Since this problem can be in P if an efficient algorithm exists to compute x from described f and y.
- This is efficient if computing $f^{-1}(x)$ is efficient. This is not consistent with the main assumption.
- Exhaustively searching x from the given f and y would take an exponential number of trials (on average).
- Thus L_f is not in P.

Problem 4

Prove that the given variant of the graph isomorphism problem (i.e., Subgraph isomorphism problem) is NP-complete.

Solution. Strategies :
• SUBGRAPH ISOMORPHISM (SGI) is in NP
• SGI is NP-Hard. As 3-SAT reduces to CLIQUE. CLIQUE reduces to SGI. (We assume 3-SAT is known to be NP-complete, as discussed in the class.)

SGI is in NP: We are given two graphs $G = (V,\ E)$ and $H = (U,\ F)$. The polynomial length certificate is a bi-directional map from a pair of vertices of G to a pair of vertices of H. This could be a list of a tuple of size polynomials in $|G|$ and $|H|$. The verification algorithm checks whether the pair suggested by the function is valid. It looks at the adjacency matrix of the graphs. The number of such checks required is bounded above by the certificate's size (i.e. polynomial). The Check process in a matrix of size n takes at most a polynomial time (in n).

Karp reduction of 3-SAT to CLIQUE: The k-clique problem in a graph G is to find if there exists a set of k vertices all adjacent to each other. The problem has an efficient verification. The certificate is the list of edges $e \in G$. The verification process checks if the entries of the adjacency matrix of G predict whether these edges are adjacent.

Now we take a CNF for 3-SAT and convert it into a graph problem as below:
The graph construction algorithm is as follows.
I. Take a clause and associate a vertex to each element. Do it for all clauses.
II. Make an edge between two vertices if their partial assignment could be consistent. It implies we can't connect x_i to $(\neg x_i)$ as they are never consistent.
III. Place no edge between vertices associated with the same clause.

• **Claim:** Due to the very own construction of the graph. A 3-CNF with k clause is satisfiable iff the graph has a k-clique. For example, if CNF is $(x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3)$. (See picture on next page.)

- Suppose an oracle provides a satisfying assignment to the CNF. We pick one true literal per clause. It is bound to exist if CNF is satisfiable. The vertices corresponding to these literals form a k-clique in the graph.
- Suppose we have access to the k-clique in the graph. Due to construction, at least one vertex (or the associated literal) is bound to be in each clause. The set of these literal, taken as TRUE, is the satisfying assignment.

• **The reduction is poly-time:** This reduction can be computed in poly-time. It re-

quires creating an adjacency matrix of size polynomial in CNF's size.

Karp reduction of CLIQUE to SGI:

- SGI ask if there exists a subgraph in G isomorphic to H. There is no contain on the subgraph to be a complete graph. By definition, a complete graph of size n is structurally equivalent to a clique of size n. Hence, if we get Oracle access to the SGI solver, we ask if a complete subgraph H exists in G. Now, this instance can be used to give an example of a clique in G of size $|H|$. It can inferred that SGI tends to generalize the idea of clique.
- This reduction is poly-time as the computation cost is mapping of the subgraph (of polynomial size).

Conclusion: $3SAT \leq_p CLIQUE \leq_p SGI$, and SGI is in NP.

Problem 5

Give an oracle that separates class NP from co-NP.

Solution.

Strategy: The key strategy remains similar to the Baker-Gill-Solovay (BGS) theorem that separates P and NP via an Oracle.

Approach:

- Construction of Oracle B is via sequentially populating it such that the co-NP Machine cannot answer correctly. A tactic similar to diagonalization is employed.

- The main difference between NP and co-NP machines comes from acceptance criteria. For NP, the input is accepted if at least one accepting path exists. But for co-NP, input is accepted if all computation paths must be accepted. This property can be employed to design the oracle B.

Oracle design:

- The definition of Unary language remains the same.
  $U_B = \{1^n : there's\ is\ a\ string\ of\ length\ n\ in\ B\}$
- The same argument holds for $U_B \in NP$. To make a non-deterministic guess and query B for its membership.
- At i-th stage, check if the co-NP machine with access to B (say, M_i^B) is accepting or rejecting the language.
- If it accepts, then do no change in B. This behaviour is the same as the BGS theorem case.
- In case of rejection, We need to modify B sequentially to make the machine's answer wrong (in the same way as most diagonalization tricks work).
- The modifications are as follows. Since this is the case of rejection, there must be a rejecting path.
- Each time the machine queries the oracle, our strategy should be to keep the answer to the path unchanged.
- The machine runs in poly-time. Hence, it is bound to make polynomial queries. Hence, there is always some unqueried string. We add this string to the B.

Acknowledgement: cs.StackExchange

Problem 6

- (a). The existence of p-isomorphism between NP-complete languages separates class P from NP.
- (b). NP-completeness of a sparse language would collapse NP to P.

Solution.

Part (a)

• The p-isomorphism conjecture (p-Is) states all NP-complete sets are p-isomorphic to each other.

We show p-Is $\implies P \neq NP$. This requires briefly discussing basic notions of finite and infinite sets in the context of classes P and NP-complete.

• SAT is a set containing all satisfiable Boolean formulas. The cardinality of this set is as big as the cardinality of the Natural number (i.e. $\aleph_0$). A heuristic argument: take a trivial Boolean formula $(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_6) \wedge ....$ This is trivially satisfied if all variables are assigned TRUE. Now, we can put such CNF formulas of different lengths into the SAT set. Hence, we can populate the SAT set to match $\aleph_0$. The same argument holds for other NP-complete problems.

• We show there exists a finite set in class P.

ODDEST PRIME = $\{n \in \mathbb{N} |$ *n is prime as well as even number*$\}$. This is a finite set.

PRIME = $\{n \in \mathbb{N} |$ *n is prime number*$\}$. This is an infinite set ($\aleph_0$). Hence, class P has both finite and infinite sets.

• Consequence of $P = NP$ assumption: The direct consequence is that NP-complete problems have poly-time algorithms. It implies each class member is poly-time (Karp) reducible to each other. This is equivalent to asking for the existence of p-isomorphism among them. The p-isomorphism between a finite and infinite set is impossible because bijection can't exist between them. The pigeonhole principle can infer this. Bijection between sets is necessary for the existence of an inverse map, a requirement of p-isomorphism. Thus, $P = NP \implies$ non-existence of p-isomorphism. Taking its contraposition yields the statement we are required to prove.

Part (b)

• The definition of Sparse language L_s is the number of strings of length n is bounded above by a polynomial in n, say (n^{c_o}). (Note: this bound is superpolynomial in n for non-sparse cases.)

• We show Karp reduction of SAT to L_S implies $P = NP$. Let the reduction function be f.

• If the Sparse language L_s is NP-complete, then SAT should reduce to L_s in poly-time.

Construction of Poly-time algorithm for SAT under the assumption:

- Make a set of n-length Boolean formulas, say $T = \{F_i\}$. Augment the set by taking each $F_i \in T$ and creating two new formulas F_i^0 and F_i^1. When the first variable of F_i is replaced by 0, it is called F_i^0. Similarly, a replacement by 1 is called F_i^1.
- This will change the size of the set. Check if it crosses a pre-defined threshold t_0. If so, then the pruning of the set is required.
- Combine two formulas in the set to form a new one as $F_0 \vee F_j = G_j$.
- Using the reduction function f, estimate $f(G_j) = z_j$. This step converts the instances of the SAT to that of L_f.
- We must prune the set T to bring its size within threshold t_0. Check for repetition within $\{z_i\}$. If there is none, remove F_1. If $z_i = z_j$, remove F_i. This pruning method ensures that F is satisfiable if and only if T contains a satisfiable formula.
- Prune until threshold criteria are met. Now check if T contains any formula with variables. If there is none, see if any of the constant formulas are True. If so, then declare F is satisfiable, other unsatisfiable.
- Repeat this process until T has no formula with a variable. At this stage, we get F is satisfiable or not.

Runtime of the above algorithm

- We show runtime is polynomial in n.
- A consistent selection of the threshold t_o is essential. This comes from the sparsity restriction of L_f. Since $max\{z_j\} = max\{f(G_j)\}$, the threshold t_0 shall be upper bound to this quantity.
- The construction of G_j from F_1 and F_j implies $|G_j| \leq 2|F|$. The computation of G_j is poly-time due to the definition of f and the sparsity restriction of the set.
- This ultimately bounds the size of set T by a polynomial in $|F|$.

Correctness of the above algorithm:

- We formally argue pruning of T doesn't alter its property (i.e. it contains some satisfiable formula.) We use the relation $G_j = F_1 \vee F_j$. If either F_1 or F_j is True, G_j is True.
- If all z_j is distinct, then the sparse set overflows the bound t_0. Hence, F_1 must not be a satisfiable formula. Then, F_1 is removed to decrease the size of the set.
- If $z_i = z_j$, remove F_j from the set. This preserves the property as per $G_j = F_1 \vee F_j$.
- Suppose F_j is satisfiable and it is removed. Still, a satisfiable formula G_j remain in the set. If F_j is not satisfiable, its removal doesn't change $T's$ property.

Conclusion: We have constructed a poly-time algorithm for SAT under the assumption of NP-completeness of the sparse set.

Acknowledgement: [M. Agarwal '09]