1 2018 Paper 6 Question 4

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2018p6q4.pdf

Consider the following two decision problems:

- **Reach** – the problem of deciding, given a *directed graph* G and two vertices a and b in G, whether there is a path in G from a to b.
- **UReach** – the problem of deciding, given an *undirected graph* G and two vertices a and b in G, whether there is a path in G from a to b.

It is known that **Reach** is NL-complete (under logarithmic-space reductions) and that **UReach** is in the complexity class L.

(a) Based on the above information, for each of the following statements, state whether it is true, false or unknown. In each case, give justification for your answer and in the case where the truth of the statement is unknown, state any implications that might follow from it being true or false.

(i) **Reach** $\leq_L$ **UReach**, i.e **Reach** is reducible in logarithmic-space to **UReach**.

This is unknown.

If there is a logarithmic space reduction from any NL-complete language to any language in L then would be able to conclude $\mathsf{NL} = \mathsf{L}$. Since $\mathsf{L} \overset{?}{=} \mathsf{NL}$, we can conclude that there is no known reduction from any NL-complete language to any language in L and therefore the statement is unknown.

If it were true then **Reach** $\in \mathsf{L}$. This implies that $\mathsf{L} = \mathsf{NL}$.

If it is false then $\mathsf{L} \neq \mathsf{NL}$.

(ii) **UReach** $\leq_L$ **Reach**

This is true.

By the definition of NL-completeness and the fact that $\mathsf{L} \subseteq \mathsf{NL}$ by definition, we have:

$$\begin{aligned} \forall A \in \mathsf{NL}. A \leq_L \textbf{Reach} \implies \\ \forall A \in \mathsf{L}. A \leq_L \textbf{Reach} \implies \\ \textbf{UReach} \leq_L \textbf{Reach} \end{aligned}$$

(iii) **UReach** is in P

This is true.

It is known that $\mathsf{L} \subseteq \mathsf{P}$. Since **UReach** $\in \mathsf{L}$, we have **UReach** $\in \mathsf{P}$.

(iv) If **Reach** is in L, then $\mathsf{P} = \mathsf{NP}$.

This is false.

Reach is in L if, and only if, for any $f(n) \in \Omega(\lg n).\mathsf{SPACE}(f(n)) = \mathsf{NSPACE}(f(n))$. However, the second statement is clearly not equivalent to $\mathsf{P} = \mathsf{NP}$ – it bounds the space required to simulate a nondeterministic Turing machine – from which we can conclude $\mathsf{NTIME}(f(n)) \subseteq \mathsf{TIME}(c^{\lg n + f(n)})$ – which was known already! This result would place a tighter bound on the constant c – but does not bring the whole expression down exponentially.

(b) Let us say that a nondeterministic Turing machine is *symmetric* if for any two configurations c_1 and c_2 of M, if $c_1 \rightarrow_M c_2$, then $c_2 \rightarrow_M c_1$. We write SL for the class of all languages that are accepted by a symmetric Turing machine using $\mathcal{O}(\lg n)$ work space on inputs of length n.

By using the configuration graph and using the fact that **UReach** is in L, explain why it follows that $\mathsf{SL} \subseteq \mathsf{L}$.

Let SNTM denote symmetric nondeterministic Turing machine.

Let SYM denote the language of SNTMs S and inputs x which they accept. So $\mathsf{SYM} = \{([S], x) | S \text{ accepts } x\}$

Consider an arbitrary language $A \in \mathsf{SL}$ and the problem of testing whether $x \in A$ for arbitrary x and A. Let S denote the SNTM which accepts A. Consider the configuration graph of an S with x on its input tape. By definition of symmetry, it has the property that if $c_1 \to c_2$ then $c_2 \to c_1$. This means that all directed edges come in pairs. So the configuration graph can be interpreted as an undirected graph.

I propose the reduction which adds an (undirected) edge from every possible accepting state in the configuration graph of $S(x)$ into a single new state and asks whether this new state is reachable from the initial state of the SNTM. I prove this is equivalent to the initial problem:

- RTP $x \in A \implies f(x) \in$ **UReach**

  If $x \in A$ then by the definition of a Nondeterministic Turing machine, there must be some path from the start state to *some* accepting state. Since the new node is connected to every accepting state, we can conclude that in $f(x)$, the new node is reachable from this accepting state.

  So we can conclude that $f(x) \in$ **UReach**, as required.

- RTP $f(x) \in$ **UReach** $\implies x \in A$

  Assume the new node is reachable from the start state. Since this new node is *only* reachable from accepting states, we can conclude that some accepting state must be reachable from the start state. So $S(x)$ can reach an accepting state. By the definition of a nondeterministic Turing machine, we have that S accepts x. So $x \in A$.

Since S accepts in logarithmic space, we have that the size of the configuration graph of S is bounded by: $|Q||\Sigma|^{\lg n}|\lg n|$. Since **UReach** is in L, we can decide whether two nodes are reachable from each other in an undirected graph using logarithmic space.

So the complexity of deciding whether $x \in A$ in SL uses $\mathcal{O}\left(\lg |Q| + \lg n \lg |\Sigma| + \lg \lg n\right) = \mathcal{O}(\lg n) \subseteq \mathsf{L}$.

2 2016 Paper 6 Question 2

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2016p6q2.pdf

The *Graph Isomorphism* problem is the problem of deciding, given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, whether there is a bijection $\beta : V_1 \to V_2$ such that

$$(u, v) \in E_1 \qquad \text{if, and only if,} \qquad (\beta(u), \beta(v)) \in E_2$$

for all $u, v \in V_1$.

The Graph Isomorphism problem is not known to be in P nor known to be NP-complete.

We define GI to be the set of all languages L which are *polynomial-time reducible* to Graph Isomorphism.

What can you conclude from the above definitions and information about the truth of the following statements? If the statement is true or false, justify your answer and if you cannot conclude anything about its truth, explain why that is so.

(a) Graph Isomorphism is in NP

This is true.

Graph Isomorphism is polynomially verifiable by the algorithm which takes a certificate β (the size of which is clearly bound by a polynomial) and checks whether it meets the criteria given in the statement of Graph Isomorphism (which can clearly be done in polynomial time):

$$(u, v) \in E_1 \iff (\beta(u), \beta(v)) \in E_2$$

(b) Graph Isomorphism is in co-NP

This is unknown.

No way of determining whether two graphs are *not* isomorphic has been given. We have been told that Graph Isomorphism is in NP, is not NP-complete but is not known to be in P. This is not sufficient to conclude that Graph Isomorphism is in co-NP.

(c) GI $\subseteq$ NP

This is true.

Consider an arbitrary $G \in$ GI. Since $G \leq_P$Graph Isomorphism (by definition), we know that G is decidable in polynomial time by a nondeterministic Turing machine by reducing G to an instance of Graph Isomorphism and testing whether the resulting graphs are isomorphic. So we have $\forall G \in \text{GI}. G \in \text{NP}$. This implies GI $\subseteq$ NP.

(d) NP $\subseteq$ GI

This is unknown.

A language L is NP-complete if it is both in NP and is NP-hard.

Since Graph Isomorphism is in NP and not known to be NP-complete, we can conclude that Graph Isomorphism must not be known to be NP-hard. Let G denote Graph Isomorphism.

By the definition of NP-hardness:

If Graph Isomorphism is not NP-hard:

$$\begin{aligned} \neg(\forall A \in \text{NP}. A \leq_P \mathsf{G}) &\implies \\ \exists A \in \text{NP}. A \not\leq_P \mathsf{G} &\implies \\ \exists A \in \text{NP}. A \notin \text{GI} &\implies \\ \text{NP} \not\subseteq \text{GI} & \end{aligned}$$

If Graph Isomorphism is NP-hard:

$$\begin{aligned} \forall A \in \text{NP}. A \leq_P \mathsf{G} &\implies \\ \forall A \in \text{NP}. A \in \text{GI} &\implies \\ \text{NP} \subseteq \text{GI} & \end{aligned}$$

Therefore, it is not known whether this is true or false.

(e) P $\subseteq$ GI

This is true.

Any language $L_1 \in$ P can be polynomially reduced to *any* language L_2 (except $\emptyset$ or Σ^*) by the algorithm which solves the problem and maps it onto an instance of L_2 with the same truth.

Therefore, $\forall L \in \text{P}. L \leq_P$ Graph Isomorphism. By the definition of GI, $\forall L \in \text{P}. L \in \text{GI}$. Therefore P $\subseteq$ GI.

(f) GI $\subseteq$ P

This is unknown.

If Graph Isomorphism is in P, then this holds. However, if Graph Isomorphism is *not* in P then this does not hold.

3 2011 Paper 6 Question 1

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2011p6q1.pdf

The following is a quotation from an Internet forum on cryptography.

Cracking RSA is NP-complete so nothing better than brute force is possible.

Your task is to evaluate to what extent (if any) this statement is true. For full marks, you will consider the following questions.

- What would it mean, precisely, for "cracking RSA" to be NP-complete? In particular, what is the decision problem involved and what is meant by saying it is NP-complete?

  Cracking RSA can be defined as the decision problem: "given a public key e and an value y which was encrypted using it, find an x such that $R(e, x) = y$".

  A language L is NP-complete if and only if it is both in NP and NP-hard.

  A language is in NP if and only if it is solvable in polynomial time by a nondeterministic algorithm.

  A language L is NP-hard if and only if all languages A in NP are polynomially reducible to it. In maths: $\forall A \in \text{NP}. A \leq_P L$.

  Saying that RSA is NP-complete means that it is both in NP and for every problem in NP, there exists a reduction from it to an instance of decrypting a RSA encryption given the public key and encrypted value (but no private key).

- Is the problem, in fact, NP-complete? Why or why not?

  Whether RSA is NP-complete is **unknown**.

  RSA is polynomially verifiable by the certificate of the x which can be encrypted to the digest. Since the set of polynomially verifiable languages is equal to the set NP, we have that RSA is in NP.

  However, RSA is not known to be NP-hard. Therefore, RSA is not known to be NP-complete.

- What is meant, precisely, by the conclusion, "nothing better than brute force is possible"?

  "Nothing better than brute force is possible" means "the only algorithms for cracking RSA perform by testing every possible value which could hash".

- Assuming the premise is correct, i.e "cracking RSA is NP-complete", does the conclusion follow? Why or why not?

  The conclusion *does not follow.*

  P = NP is an *open problem.* If RSA was NP-complete and P was **not** equal to NP, then the author would be partially correct. There would be no deterministic polynomial-time algorithm for cracking RSA. However, randomised algorithms may be able to find solutions in polynomial time with very high probability. Therefore, the statement still does not follow in this case.

  In the case that P = NP, since cracking RSA is in the complexity class NP; there would exist a polynomial-time algorithm for cracking RSA and therefore the conclusion would *also* not follow.

- What is the relationship, more generally, between encryption systems and NP-completeness?

  No encryption algorithm is known to be NP-complete.

"cracking RSA" is believed to be the inverse of a one-way function. The properties of a one-way function f are as follows:

- f is one-to-one

- $f \in \mathsf{FP}$
- $f^{-1} \notin \mathsf{FP}$
- $\exists k \in \mathbb{R}. \forall x. |x|^{\frac{1}{k}} \leq |f(x)| \leq |x|^k$

Where FP is defined as the set of functions which are computable in polynomial time.

UP (unambiguous polynomial) is the set of languages which are computable by an unambiguous nondeterministic Turing machine in polynomial time. In maths, if L is a language in UP, then there must exists some nondeterministic polynomial-time verifier R such that:

$$L = \{x | \exists ! y. R(x, y)\}$$

P $\neq$ UP $\iff$ there exists a one-way function.

Note that P $\neq$ UP is a stronger statement than P $\neq$ NP.

By the definition of UP, we have UP $\subseteq$ NP. Therefore, if P $\neq$ UP, we have P $\neq$ NP.

There are some encryption algorithms which are provably secure (such as the one-time pad), where the only way to decrypt an encrypted string is to know the decryption key.