Homework on Graph Isomorphism
CSCI 6114 Fall 2021

Joshua A. Grochow

Due Thursday Oct 7, 2021

1. A multigraph allows self-loops and parallel edges (two edges are parallel if they have the same start as one another and the same end as one another). In a multigraph, the set E of edges is no longer thus a subset of $V \times V$, but a sub-*multi*set. Two multigraphs G, H are isomorphic if there is a bijection $\pi \colon V(G) \to V(H)$ and a bijection $\rho \colon E(G) \to E(H)$ such that for every edge $e = (u, v) \in E(G)$, we have $\rho(e) = (\pi(u), \pi(v))$. Show that testing isomorphism of directed multigraphs is GI-complete.

2. A hypergraph consists of a vertex set V and set of hyperedges E, where each hyperedge e is a nonempty subset of vertices. Two hypergraphs G, H are isomorphic if there is a bijection π between their vertices such that for each subset $S \subseteq V(G)$, we have $S \in E(G) \Leftrightarrow \pi(S) \in E(H)$, where $\pi(S) = \{\pi(v) : v \in S\}$. Show that hypergraph isomorphism is GI-complete.

3. Subgraph Isomorphism is the problem

   SUBGRAPH ISOMORPHISM
   Input: Two graphs G, H
   Decide: Is there an injective function $\pi \colon V(G) \to V(H)$ such that, for all $(u, v) \in V(G)$, $(u, v) \in E(G) \Rightarrow (\pi(u), \pi(v)) \in E(H)$?

   Note that here G can be smaller than H, and also note that the arrow only goes in one direction (so we are finding G as a general subgraph, not an induced subgraph, of H). INDUCED SUBGRAPH ISOMORPHISM is similar but where the arrow goes both ways ($(u, v) \in E(G) \Leftrightarrow (\pi(u), \pi(v)) \in E(H)$).

(a) Show that Subgraph Isomorphism is NP-complete even when G, H have the same number of vertices. (Compare to Induced Subgraph Isomorphism when G, H have the same number of vertices, which is precisely GI.)

(b) Show that Induced Subgraph Isomorphism is NP-complete when $|V(G)| \sim c|V(H)|$ for any $c > 1$.

4. The Exponential Time Hypothesis is a strengthening of $\mathsf{P} \neq \mathsf{NP}$, useful for understanding more fine-grained complexity of problems:

   Exponential Time Hypothesis (ETH): Let δ be the infimum of the numbers such that 3SAT on n variables can be solved by an algorithm running in $O(2^{\delta n})$ time. Then $\delta > 0$.[1]

   (a) Using the fact that GI can be solved in quasi-polynomial ($2^{\mathrm{poly}(\log n)}$) time on n-vertex graphs (Babai, STOC '16), show that ETH implies that GI is not NP-complete.

   (b) Show that ETH implies that there is no polynomial-time many-one reduction from 3SAT to Induced Subgraph Isomorphism that maps n-variable 3SAT instances to instances of Induced Subgraph Isomorphism with $|V(H)| \leq |V(G)| + o(n)$. (Thus, assuming ETH, there is a pretty sharp threshold (in terms of $|V(H)| - |V(G)|$) between which instances are NP-complete and which are not.)

5. An *automorphism* of a graph G is an isomorphism from G to itself; an automorphism is non-trivial if it is a non-identity bijection $\pi \colon V(G) \to V(G)$. A graph is *rigid* if it has no nontrivial automorphisms. (In a precise sense, almost all graphs are rigid.)

   (a) Show that IsRigid,[2] the problem of testing whether a given input graph is rigid, reduces to GI in polynomial time. *Hint:* Search-to-decision reduction.

   (b) Rigid Graph Isomorphism is the following *promise* problem:

[1]Sometimes this is defined as 3SAT cannot be solved in $2^{o(n)}$ time, but there is a subtle difference between the two. Do you see what it is?

[2]The set of non-rigid graphs is the language corresponding to the decision problem GA or Graph Automorphism, of deciding whether a graph has a nontrivial automorphism. We mention this because it is often studied under the name GA in the literature.

Rigid Graph Isomorphism
Input: Two graphs G, H
Promise: G, H are both rigid
Decide: Are G and H isomorphic?

We say an algorithm solves a promise problem if, *on inputs satisfying the promise*, it correctly solves the decision problem. On inputs that don't satisfy the promise, the algorithm may have arbitrary behavior (including making no output, incorrect output, or running forever).

Show that RigidGI is polynomial-time Turing equivalent to IsRigid. That is, each can be solved in polynomial time with an oracle for the other (in symbols, we'd write $\textsc{RigidGI} \in \mathsf{PromiseP}^{\textsc{IsRigid}}$ and $\textsc{IsRigid} \in \mathsf{P}^{\textsc{RigidGI}}$, or $\textsc{IsRigid} \equiv_T^p \textsc{RigidGI}$).