In this lecture we will consider the Subgraph Isomorphism problem (SI): Given graphs $H = (V_H, E_H)$, $G = (V, E)$, determine whether G contains a copy of H.

There are two versions of the problem:

1. In the **induced** version of SI, one is looking for an "induced copy", i.e. a one-to-one mapping $f : V_H \to V$ so that $(f(u), f(v)) \in E$ if and only if $(u, v) \in E_H$.

2. In the **non-induced** version of SI, one is looking for a not necessarily induced copy, i.e. a one-to-one mapping $f : V_H \to V$ so that if $(u, v) \in E_H$, then $(f(u), f(v)) \in E$. However, if $(u, v) \notin E_H$, then $(f(u), f(v))$ may or may not be an edge in G, so the copy $f(H)$ is actually a supergraph of H.

In general, SI is NP-Complete, since the case when H is a clique is already NP-Complete. Today we will consider the case when the number of vertices of H is constant, $|V_H| = k = O(1)$, and the running time is measured in terms of the number of vertices n of G. This case is often referred to as Graph Pattern Detection.

The **brute-force** algorithm for finding a pattern graph H in G is to try all n^k ordered k-tuples $(v_1, \ldots, v_k)$ of vertices of G and to check if mapping the ith vertex of H to v_i for each $i \in [k]$ is an isomorphism (induced or non-induced, whichever one we care about). The running time is $O(n^k k^2)$ and since k is constant, it is just $O(n^k)$.

Induced vs non-induced. Here we will show that if there is a fast algorithm for detecting induced copies of H, then there is also a fast algorithm for detecting non-induced copies of H. Thus the induced version of SI is at least as hard as the non-induced version. In fact, for many patterns the non-induced version is easier.

Theorem 0.1. *Suppose that there is an $O(n^c)$ time algorithm that can determine if an n-node graph contains an induced copy of a k-node graph $H = (V_H, E_H)$, then there is also an $\tilde{O}(n^c)$ time algorithm that can determine if an n-node graph contains a non-induced copy of H.*

Let's begin the proof of the theorem. Let $V_H = \{h_1, \ldots, h_k\}$.

Let $G = (V, E)$ be an n-node graph in which we want to detect a non-induced H, and suppose we have an $O(n^c)$ time algorithm that can detect an induced H in any n-node graph. We will create an n-node graph $G' = (V', E')$ so that if G does not have a non-induced H, then G' does not have any induced H, and if G has an H, then G' has an induced H with constant probability.

For each vertex $v \in V$, let $c(v)$ be a color selected independently and uniformly at random from $\{1, \ldots, k\}$ (each with probability $1/k$). Let for each $c \in \{1, \ldots, k\}$, V_c be the vertices v with $c(v) = c$.

G' will be a k-partite graph with partitions $V_1, \ldots, V_k$, so there will be no edges between two vertices in the same V_i.

For every edge (u, v) of G, if $(h_{c(u)}, h_{c(v)})$ is an edge of H, then place edge (u, v) into G'. These are all the edges of G'.

Notice that the construction means that for $i \neq j$ where $(h_i, h_j) \in E_H$, the pairs in $V_i \times V_j$ have edges between them iff they had edges in G, whereas when $i = j$ or $(h_i, h_j) \notin E_H$, there are no edges between any pair in $V_i \times V_j$.

The construction also ensures that G' is just a subgraph of G. Thus, if G' contains an induced copy of H, then G contains a non-induced copy of H.

On the other hand, suppose that G contains a non-induced copy of H, $v_1, \ldots, v_k$ where if $(h_i, h_j) \in E_H$, then $(v_i, v_j) \in E_H$. Then with probability $1/k$, for each i, $c(v_i) = i$. If this happens, then for every i, j such that $(h_i, h_j) \in E_H$, we will also have that $(v_i, v_j) \in E'$, and that for $(h_i, h_j) \notin E_H$, $(v_i, v_j) \notin E'$, and so G' will have an induced copy of H with probability $1/k$, which is constant.

We can repeat this construction $O(k^k \log n)$ times and run the induced H detection algorithm on each G' that we construct. If G has a copy of H, then the probability that none of the G's have an induced copy is at most $1/\text{poly}(n)$. Thus, with probability at least $1 - 1/\text{poly}(n)$ we will solve the non-induced problem on G.

This reduction is an example of *color-coding*, a method used quite often for subgraph detection algorithms. The reduction as presented is randomized, but there are known techniques to derandomize color-coding without real loss in the running time. Thus there is also a deterministic reduction from the non-induced to the induced version of SI.

Cliques are the hardest patterns. On the problem set you will prove the following theorem:

Theorem 0.2. *Suppose there is an $O(n^c)$ time algorithm that can determine if an n-node graph contains a k-clique. Then there is also an $O(n^c)$ time algorithm that can determine if an n-node graph contains an induced copy of H, for any fixed k-node pattern H.*

We thus get that cliques are the hardest patterns.

To prove the theorem you will take an n-node graph G in which you want to detect a copy of H and you will create an kn-node k-partite graph G' that contains a k-clique if and only if G contains an induced copy of H. The reduction will be similar in spirit to the one from non-induced H-detection to induced H-detection. However, it will be simpler in that you will not need randomization.

Fastest known algorithm for k-clique. We have already seen the fastest 3-clique algorithm for n-node graphs. It runs in $O(n^\omega)$ time and involves squaring the adjacency matrix.

Here we will show how to find k-cliques faster than the brute-force algorithm, basically by reducing the problem to triangle detection in a huge new graph.

Let $G = (V, E)$ be the given host graph in which we want to find/detect a k-clique.

First let's consider the divisibility of k by 3. Let $k = 3\ell + r$, where $r \in \{0, 1, 2\}$.

If $r \neq 0$, we will reduce to finding a 3ℓ-clique in the following simple way. If $r = 1$, then for every node x of G, take the subgraph of G induced by the neighbors of x and try to find a $k - 1 = 3\ell$-clique in the neighborhood of x. If a $k-1$-clique is detected in the neighborhood of some x, then together with x, we get a k-clique. Otherwise, if no x has a $k-1$-clique in its neighborhood, then there is no k-clique in G.

If $r = 2$ we basically do the same thing, except that we try all edges $e = (x, y)$ of G and consider the subgraph of G induced by the intersection of the neighborhoods of x and y. Search for a $k-2$-clique in this graph.

For both $r = 1, 2$ the running time becomes n^r times the time to detect a $k - r = 3\ell$-clique in an $\leq n$-node graph.

When k is divisible by 3, Nešetril and Poljak showed how to solve k-clique faster via a reduction to triangle detection.

Theorem 0.3. *If k is divisible by 3, then there is an $O(n^{\omega k/3})$ time algorithm that can detect if an n-node graph has a k-clique.*

To prove the theorem, let $G = (V, E)$ be our given graph. We will create a huge graph $G' = (V', E')$ on $O(n^{k/3})$ vertices which will have a triangle if and only if G has a k-clique.

For every $k/3$-tuple of vertices of G, $(v_1, \ldots, v_{k/3})$, we check if $(v_1, \ldots, v_{k/3})$ is a $k/3$ clique, and if so, add $(v_1, \ldots, v_{k/3})$ as a vertex in G'. Now G' has at most $n^{k/3}$ vertices.

For every pair of vertices of G', $(v_1, \ldots, v_{k/3})$ and $(u_1, \ldots, u_{k/3})$, check whether their tuples are disjoint, and that together $(v_1, \ldots, v_{k/3}, u_1, \ldots, u_{k/3})$ form a $2k/3$-clique in G. That is, we check if for every $i, j \in [k/3]$, $v_i \neq u_j$ and $(v_i, u_j) \in E$.

If this is the case, then we add an edge between $(v_1, \ldots, v_{k/3})$ and $(u_1, \ldots, u_{k/3})$ in G'.

The construction of G' takes $O(n^{2k/3})$ time.

Now if G' contains a triangle, through $(x_1, \ldots, x_{k/3})$, $(v_1, \ldots, v_{k/3})$ and $(u_1, \ldots, u_{k/3})$, then the k-tuple $(x_1, \ldots, x_{k/3}, v_1, \ldots, v_{k/3}, u_1, \ldots, u_{k/3})$ must be a k-clique in G since every pair of G vertices in the k-tuple is connected by an edge in G by construction.

On the other hand, if G has a k-clique $x_1, \ldots, x_k$, then the three nodes of G' $(x_1, \ldots, x_{k/3})$, $(x_{k/3+1}, \ldots, x_{2k/3})$, $(x_{2k/3+1}, \ldots, x_k)$ exist and form a triangle in G'.

Thus G' contains a triangle if and only if G has a k-clique.

Finding a triangle in G' takes $O(n^{\omega k/3})$ time since G' has $O(n^{k/3})$ nodes.

Some H are faster than k-clique. We have seen that k-clique is the hardest pattern, and we have seen the fastest known algorithm for k-clique. How much easier are the other patterns?

If we are considering the non-induced version of SI, many H on k-nodes are much easier than k-clique. The simplest case is finding a k-independent set – it is in basically constant time, as there is a non-induced k-independent set, if and only if the graph has at least k-nodes. For a slightly more interesting case, it is known that for any constant k, a k-path can be detected in *linear* time! In a later lecture we will see how to find k-cycles in $O(n^2)$ time for any constant k, if k is even. (When k is odd, k-cycle detection is equivalent to triangle finding, and can be solved in $O(n^\omega)$ time.)

Here we will focus on the induced version of the problem. While detecting k-independent sets is a trivial problem for the non-induced version of SI, the induced k-independent set problem is equivalent to finding a k-clique (just consider the complement of the given graph). So induced H-detection can be a hard problem even for simple looking H.

The following theorem was proven by Vassilevska Williams, Wang and Williams for four node subgraphs:

Theorem 0.4. *Let H be any 4-node graph that is not the 4-clique or 4-independent set. Then detecting H in an n-node graph can be done in $\tilde{O}(n^\omega)$ time.*

Thus, all 4-node Hs that are not the clique or independent set can be found in the same time as the fastest 3-clique algorithm. The theorem was later extended by Dalirrooyfard, Vassilevska Williams and Vuong to show that for $k = 5$ or 6, any k-node H that is not the k-clique or k-independent set can be detected in the same time as the best known algorithm for $k-1$-clique. Extending this result for larger values of k is an open problem.

Here we will focus on $k = 4$ and the special case where H is a "diamond", i.e. a 4-clique missing an edge.

Let $G = (V, E)$ be a given n node graph and let A be its $n \times n$ adjacency matrix. Let $\#Dia$ denote the number of diamonds in G, and let $\#Cli$ be the number of 4-cliques in G.

First consider the quantity $A^2[u, v]$ for an edge (u, v). This is the number of triangles going through edge (u, v).

Now, every diamond is formed by an edge (u, v) and two triangles using this edge, (x, u, v) and (y, u, v). An induced diamond means that (x, y) is not an edge. If (x, y) is an edge then we instead have a 4-clique.

Then the quantity

$$\binom{A^2[u,v]}{2}$$

is the number of induced diamonds through edge (u, v), plus the number of 4-cliques through edge (u, v).

The quantity

$$\Gamma = \sum_{(u,v)\in E} \binom{A^2[u,v]}{2}$$

equals the number of induced diamonds plus the 6 times the number of 4-cliques. This is because every diamond appears once, for the edge (u, v) that has two triangles attached to it that share no other edges, while every 4-clique appears once for each of its edges.

We have $\Gamma = \#Dia + 6\#Cli$, and we can compute Γ in $O(n^\omega)$ time by multiplying A by itself and computing the sum over all edges in additional $O(n^2)$ time.

Notice that then $\Gamma \equiv \#Dia(\mod 6)$. Thus, if the number of diamonds of G is actually nonzero mod 6, then we can determine that G has a diamond in $O(n^\omega)$ time.

Next we will show how to take a graph G and obtain a subgraph G' of G such that if G has a diamond, then the number of diamonds in its subgraph G' is nonzero mod 6 with constant probability.

The sampling is very simple: for every vertex of G, include it in G' with probability 1/2. We only need to show that this sampling works.

Think of the n vertices of G as $[n]$ (they each have an integer name from 1 to n). We say that $a, b, c, d \in V$ induce a diamond in G if there is some permutation π of a, b, c, d so that $(\pi(a), \pi(b))$, $(\pi(a), \pi(c))$, $(\pi(a), \pi(d))$, $(\pi(b), \pi(c))$, $(\pi(b), \pi(d))$ are edges in G and $(\pi(c), \pi(d))$ is not an edge in G.

Consider the following polynomial over the n variables $x_1, \ldots, x_n$.

$$P(x_1, \ldots, x_n) = \sum_{\substack{i_1<i_2<i_3<i_4, \\ (i_1,i_2,i_3,i_4)\text{induce a diamond in } G}} x_{i_1} x_{i_2} x_{i_3} x_{i_4}.$$

Evaluating P on the length n all 1s vector, gives the number of diamonds in G. Moreover, for every subset $S \subseteq V$, if we evaluate P on the vector $\mathbf{x_S} = (x_1, \ldots, x_n)$ where $x_i = 1$ if $i \in S$ and $x_i = 0$ otherwise, then $P(\mathbf{x_S})$ is the number of diamonds in the subgraph of G induced by S.

What we want to know is, what is the probability that $P(\mathbf{x_S})$ is $\neq 0 \mod 6$ if S is a random subset of V where each vertex is picked with probability 1/2. This is the same as the probability that we get nonzero mod 6 when we evaluate P on a uniformly random vector from $\{0, 1\}^n$.

With the lemma below (applied with $m = 6, d = 4$) we will get that this probability is constant, at least 1/16. Thus detecting if G has a diamond can be done whp in time $\tilde{O}(n^\omega)$.

Recall that a polynomial over variables $x_1, \ldots, x_n$ is multilinear over $\mathbb{Z}_m$ if it is of the form $P(x_1, \ldots, x_n) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i$, where the coefficients $c_S \in \{0, \ldots, m-1\}$ are elements of $\mathbb{Z}_m$ for every choice of $S \subseteq [n]$. The degree of a multilinear polynomial is the largest size of S such that $c_S \neq 0$.

Lemma 0.1. *Let $m \geq 2$ be an integer. Let $P = (x_1, \ldots, x_n)$ be a non-zero multilinear polynomial over $\mathbb{Z}_m$ of degree d. Then*

$$Pr_{(a_1,\ldots,a_n)\in\{0,1\}^n}[P(a_1, \ldots, a_n) \neq 0 \mod m] \geq 1/2^d.$$

You'll prove this lemma on the problem set.