CME 305: Discrete Mathematics and Algorithms

1 Basic Definitions and Concepts in Graph Theory

A **graph** $G(V, E)$ is a set V of **vertices** and a set E of **edges**. In an **undirected** graph, an edge is an unordered pair of vertices. An ordered pair of vertices is called a **directed** edge. If we allow **multi-sets** of edges, i.e. multiple edges between two vertices, we obtain a **multigraph**. A self-loop or **loop** is an edge between a vertex and itself. An undirected graph without loops or multiple edges is known as a **simple** graph. In this class we will assume graphs to be simple unless otherwise stated.

If vertices a and b are endpoints of an edge, we say that they are **adjacent** and write $a \sim b$. If vertex a is one of edge e's endpoints, a is **incident** to e and we write $a \in e$. The **degree** of a vertex is the number of edges incident to it.

A **walk** is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $\forall i \in 1, 2, \ldots, k-1$, $v_i \sim v_{i+1}$. A **path** is a walk where $v_i \neq v_j$, $\forall i \neq j$. In other words, a path is a walk that visits each vertex at most once. A **closed walk** is a walk where $v_1 = v_k$. A **cycle** is a closed path, i.e. a path combined with the edge (v_k, v_1). A graph is **connected** if there exists a path between each pair of vertices. A **tree** is a connected graph with no cycles. A **forest** is a graph where each connected component is a tree. A node in a forest with degree 1 is called a **leaf**.

The size of a graph is the number of vertices of that graph. We usually denote the number of vertices with n and the number edges with m.

Claim 1 *Every finite tree of size at least two has at least two leaves. Furthermore, the number of edges in a tree of size n is $n-1$.*

Proof: The first property can be seen by starting a path at an arbitrary node, walking to any neighbor that has not been visited before. Note that since the number of vertices is at least two, we can not have vertices of degree zero. After at most n steps, we must reach a vertex v with no unvisited neighbors. The degree of v can not be larger than one. Otherwise, the graph has a cycle and therefore it is not a tree. We can then traverse a path starting at v, and the vertex where the path stops must be a second leaf.

We can show the second property by induction. The base case $n = 1$ is trivial. Any tree of size $n > 1$ has at least one leaf. Removing a leaf results in a tree with one less node and one less edge. Therefore, a tree with n vertices has one more edge than a tree with $n-1$ vertices. ■

Proposition 1 *If a graph $G(V, E)$ has any two of the following three properties, it has all three.*

1. *G is connected.*
2. *G has no cycles.*
3. *$|E| = |V| - 1$.*

Therefore, any graph with any two of these properties is a tree.

Proof:

$(1), (2) \Rightarrow (3)$: Already proved in Claim 1.

$(1), (3) \Rightarrow (2)$: We prove this by contradiction; assume (1) and (3) hold, but (2) does not hold, i.e., graph G has a cycle. Consider cycle c in G: we remove one of the edges from cycle c. The graph remains connected.

We repeat this procedure until there is no cycle left. The resulting graph $G'(V, E')$ is still connected. Moreover, since we removed at least one edge, $E' \subset E$ and $|E'| < |E|$. On the other hand, G' is a connected graph with no cycle, hence by the first part of the proof, it has $|V| - 1$ edges, which is a contradiction.

$(2), (3) \Rightarrow (1)$: We prove this by contradiction; assume (2) and (3) hold, but (1) does not hold, i.e., graph G is not connected. Consider the connected components of G. In other words, partition V into $k > 1$ subsets $V_1, ..., V_k$ such that (i) there is no edge between V_i and V_j for $i \neq j$ and (ii) the graph induced by V_i, that is $G(V_i, E_i)$ where $E_i = \{\{u, v\} : \{u, v\} \in E, u, v \in V_i\}$ is connected. Each G_i is connected and has no cycles, therefore by the first part of the proof, $|E_i| = |V_i| - 1$. Since there is no edge between V_i and V_j, $i \neq j$, $|E| = \sum_{i=1}^{k} |E_i| = |V| - k < |V| - 1 = |E|$, which is a contradiction. ■

2 Eulerian Circuits

Definition: A closed walk (circuit) on graph $G(V, E)$ is an **Eulerian circuit** if it traverses each edge in E exactly once. We call a graph **Eulerian** if it has an Eulerian circuit.

The problem of finding Eulerian circuits is perhaps the oldest problem in graph theory. It was originated by Euler in the 18th century; the problem was whether one could take a walk in Konigsberg and cross each of the four bridges exactly once. Motivated by this, Euler was able to prove the following theorem:

Theorem 1 *(Euler, 1736)*

Graph $G(V, E)$ is Eulerian iff G is connected (except for possible isolated vertices) and the degree of every vertex in G is even.

Proof:

"$\Rightarrow$": assume G has an Eulerian circuit. Clearly, G must be connected, otherwise we will be unable to traverse all the edges in a closed walk. Suppose that an Eulerian circuit starts at v_1; note that every time the walk enters vertex $v \neq v_1$, it must leave it by traversing a new edge. Hence, every time that the walk visits v it traverses two edges of v. Thus d_v is even. The same is true for v_1 except for the first step of the walk that leaves v_1 and the last step that it enters v_1. Again, we can pair the first and last edges in the circuit to conclude that d_{v_1} is also even.

"$\Leftarrow$": we prove this by strong induction on the number of nodes. For $n = 1$, the statement is trivial. Assume G has $k + 1$ vertices, it is connected, and all the nodes have even degrees. Pick any vertex $v \in V$ and start walking through the edges of the graph, traversing each edge at most once. Given that all the degrees of the nodes in G are even, we can only get stuck at v. Remove the circuit of visited edges, c_1, from G; Call this new graph G'. Node v is an isolated vertex in G' therefore any connected component of G' has at most k vertices. Moreover, it is easy to see that the degree of nodes in G' are even (we removed an even number of edges from each vertex). Thus by the induction assumption, all the connected components of G' are Eulerain. Now, we have a collection of circuits $c_1, \ldots, c_l$. We can merge them into one large circuit in the following way:

Traverse c_1 until it enters a vertex v that is part of a not-yet traversed circuit c_j. Then start traversing c_j, following the same rule. We resume traversing c_1 once all the edges in c_j has been visited. Note that since G was connected, we will be able to reach all circuits in this fashion. ■

Euler's theorem gives necessary and sufficient condition for whether a graph is Eulerian which can be easily checked in linear time. Note that this is a "constructive" proof: we explicitly defined an algorithm to find an Eulerian circuit in the proof. In addition, the algorithm is close to optimal, given that its running time is at most $2|E|$ and $|E|$ is an obvious lower bound on the number of operations any algorithm can achieve.

We can extend the result to find a necessary and sufficient condition for Eulerian paths, which is a walk (not necessarily closed) that visits each edge exactly once:

Claim 2 *G has an Eulerian path iff it is connected and only two of its vertices have odd degrees.*

We can also define Eulerian circuits of a directed graph.

Claim 3 *Let $G(V, E)$ be a directed graph. G has an Eulerian circuit iff*

- *G is connected*
- *$\forall v \in V$, $indegree(v) = outdegree(v)$*

A seemingly similar structure was defined by Sir William Rowan Hamilton in the 19th century, but concerns visiting every vertex exactly once rather than every edge.

Definition: A cycle C in G is **Hamiltonian** if it visits every vertex in V exactly once.

This was originally invented and marketed by Hamilton as a board game, where the object was to find a Hamiltonian cycle around the edges of a dodecahedron. The puzzle failed financially, but the concept lives on.

A famous example of a Hamiltonian cycle problem is the *Knight's tour*, which asks whether one can move a knight in a chessboard while visiting each vertex exactly once and returning to the starting vertex. The graph representation consists of 64 vertices corresponding to the squares of the board, with an edge between vertices if there is a legal knight move between the two squares. Is there a Hamiltonian path (cycle) in the chess board graph? The answer is yes. Gauss enumerated the number of different Hamiltonian cycles in such a graph.

As opposed to the Eulerian circuits, Hamiltonian paths are remarkably difficult to find. We will show later in the class that there exists no polynomial running time algorithm to find a Hamiltonian cycle in a general graph unless $P = NP$. For this particular problem, the best known running time is super-polynomial (it is not bounded above by any polynomial).

3 Application: DNA Sequencing

Recall that DNA consists of two complementary strands of nucleotides. There are 4 different nucleotides, namely A, G, C and T (for those wondering, U only appears in RNA, replacing T). Our goal is to be able to read a DNA sequence. What's the best way to do it? One possible way (one that we will not study), due to Frederick Sanger, is called the Dideoxy termination method, or chain termination method. For this he was awarded the 1980 Nobel price in Chemistry.

A simpler technique, invented by Professor Patrick Brown from Stanford University, is based on *DNA arrays*. Basically the original DNA sequence is cut into pieces of a given length, tagged with a fluorescent agent, and exposed to an array of known sequences of the same length, so that the DNA pieces will hybridize with complementary sequences in the array. We can detect the presence of particular sequences based on the strength of florescence in each square of the array. Here we treat the presence or absence as a binary variable, but in reality the florescence will be graded.

Given a DNA sequence, s, of length n, *DNA arrays* detects all the length l subsequence of s, $\{\sigma_1, \sigma_2, \ldots, \sigma_{n-l+1}\}$, which is called the *Spectrum* of s, $Spectrum(s, l)$. However, it does not provide any information about the position of a substring in s.

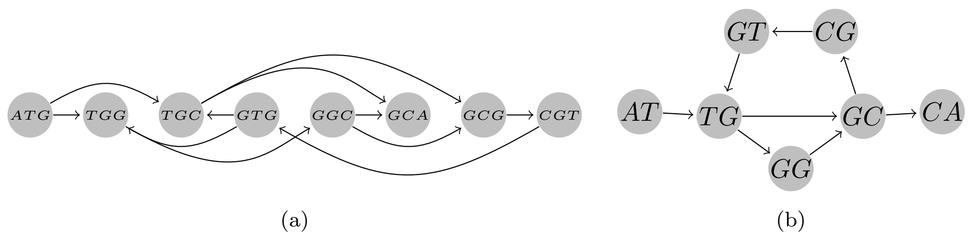

Figure 1: (a) Hamiltonian Based Reconstruction Graph (b) Eulerian Based Reconstruction Graph

Question: Given $Spectrum(s, l)$, can we reconstruct s?

Answer: (Hamiltonian Based) An ordered pair of substrings (σ_i, σ_j) *overlap* if the last $l-1$ letters of σ_i overlap with the first $l-1$ letters of σ_j. Given Spectrum(s,l), we construct a directed graph $G_H(s, l)$ in the following way: we introduce vertex v_i for σ_i, and put a directed edge from v_i to v_j if (σ_i, σ_j) overlap. An example is illustrated in Fig.1(a). It is not hard to see that there is a one-to-one mapping from every Hamiltonian path of $G_H(s, l)$ and a sequence with $Spectrum(s, l)$. Note that a graph can have more than one Hamiltonian paths; or equivalently there may exist $s_1 \neq s_2$ such that $Spectrum(s_1, l) = Spectrum(s_2, l)$. For example spectrum of Fig. 1(a) corresponds to two possible sequences; $ATGCGTGGCA$ and $ATGGCGTGCA$.

As we noted above finding a Hamiltonian path is an NP-Hard problem. Hence, we need to formulate the problem in a way that we can solve it more efficiently; we reduce the problem to an Eulerain path problem by constructing a graph whose edges correspond to σ_i's. Thus by traversing all the edges exactly once - or equivalently finding an Eulerain circuit- we can reconstruct s; We define a directed graph $G_E(s, l)$ in the following way: we introduce a vertex v for each $l-1$ substring of s. We connect v to w if there exists σ_i such that the fist $l-1$ letters of σ_i coincide with v and its last $l-1$ letter coincide with w. As an example, $G_E(s, l)$ of Spectrum $\{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT\}$ is shown in Fig.1(b). It is easy to see that there is a one-to-one mapping from every Eulerian path of G_H and a sequence with $Spectrum(s, l)$. For example an Eulerain path of $G_E(s, l)$ in Fig.1(b) is $ATGGCGTGCA$.

4 Minimum Spanning Trees

We are given a connected graph $G(V, E)$, each edge e has a cost (wight) of $c(e) > 0$. Let the cost of G to be $\sum_{e \in E} c(e)$.

Question: What is cheapest connected subgraph of G?

Clearly, the solution G' is a tree. Otherwise, suppose G' has at least one cycle. Pick an edge from that cycle and delete it. The resulting graph is still connected and has a lower cost, thus G' could not be the cheapest connected subgraph.

Definition: For a connected graph $G(V, E)$, a *spanning tree* of G, $T(G)$ is a subgraph of G that is a tree and has vertex set V. Given the cost function $c(\cdot)$, the *minimum spanning tree* of G, $MST(G)$, is the cheapest spanning tree of G.

To find the cheapest subgraph of G - or equivalently $MST(G)$ - one way would be to enumerate all the spanning trees and find the cheapest one. However, the number of spanning trees can be exponentially large. We need to come up with a more efficient way to find $MST(G)$.

Joseph Kruskal proposed the following greedy algorithm to produce an MST. For simplicity assume that the $c(\cdot)$ is a one-to-one function.

The Greedy Algorithm: Greedy-MST

1. Given graph $G(V,E)$, initialize $E(T) = \{\}$ and $V(T) = V$.
2. Re-index the edges $e_1, ..., e_m$ so that $c(e_{i_1}) < c(e_{i_2}) < ... < c(e_{i_m})$
3. While $|E(T)| < |V| - 1$, add the cheapest unused e_{i_k} that does not create a cycle.

Claim 4 *Greedy-MST produces $T^* = MST(G)$.*

Proof: Let T be the tree that Greedy-MST outputs. Clearly, $e_{i_1} \in E(T)$. We prove that $e_{i_1} \in E(T^*)$; assume $e_{i_1} \notin E(T^*)$, adding e_{i_1} to T^* results in creating a cycle in the new graph. Remove any edge from such cycle (except e_{i_1}); we have a new spanning tree whose weight is strictly smaller than that of T^*, which is contradicting with the assumption that T^* is $MST(G)$.

We prove the rest by induction on the steps of the progress of the algorithm. Let $i_{p(1)}, i_{p(2)}, \ldots, i_{p(k)}$ be the indices of the first k edges that Greedy-MST picks. Assume they all belong to T^*. Suppose $e_{i_{p(k+1)}}$ does not belong to T^*. Adding $e_{i_{p(k+1)}}$ to T^* results in having a graph with a cycle, c. If we can show that c contains at leat one edge with index $l > p(k+1)$ - or equivalently $c(e_{i_l}) > c(e_{i_{p(k+1)}})$ - then by removing e_{i_l}, we have a new spanning tree whose weight is strictly smaller than the cost of T^*, which is contradicting with the assumption that T^* is $MST(G)$.

Now we prove that c contains at leat one edge with index $l > p(k+1)$; suppose c does not have such an edge, hence all the edges of c have indices smaller than or equal to $p(k+1)$. Looking back at Greedy-MST algorithm, c must have an edge with index $x < p(k+1)$ where $x \neq p(j)$, $1 \leq j \leq k+1$, otherwise $e_{i_{p(k+1)}}$ would have created a cycle with the edges that have already been picked by the algorithm, however the algorithm does not select such an edge. For $x \neq p(j)$, $1 \leq j \leq k$, assume $p(j) < x < p(j+1)$ the algorithm did not pick i_x because adding e_{i_x} to $\{e_{i_{p(1)}}, e_{i_{p(2)}}, \ldots, e_{i_{p(j)}}\}$ would create a cycle. However, since $\{e_{i_{p(1)}}, e_{i_{p(2)}}, \ldots, e_{i_{p(j)}}\}$ and e_{i_x} belong to T^*, T^* contains a cycle which is contradicting with T^* being a tree. ∎

5 Application: Clustering in Bio-informatics

Biologists commonly use DNA arrays to analyze gene function. This technique allows the measurement of "expression level" (amounts of mRNA produced) in genes. The result is a $n \times m$ matrix recording in each row the expression levels of a gene.

Clustering algorithms allow for genes with similar expression levels to be linked together in hope that genes placed in common clusters share common functions. To achieve this, an $n \times n$ distance matrix d is constructed. $d(i,j)$ records how close, or similar, genes i and j are based on their expression levels.

Given n nodes (genes), a distance function, and $l < n$, the clustering problem aims to partition the nodes into l groups so that the nodes in the same group are "close" and the nodes in different groups are "far apart". One way to formulate the problem and address these goals is as follows:

Let $V = \{v_1, v_2, dots, v_n\}$ be the set of nodes, and $d : V \mapsto \mathbb{R}^+$ be the corresponding distance function that satisfies the following conditions:

1. $d(v_i, v_i) = 0$.
2. $d(v_i, v_j) = d(v_j, v_i)$.

Let $C = \{C_1, C_2, \ldots, C_l\}$ be a partition of V where $C_i \neq \emptyset$. We define $D(C_i, C_j)$ to be:

$$D(C_i, C_j) = \min_{v \in C_i, w \in C_j} d(v, w),$$

and $D^*(C) = min_{i,j} D(C_i, C_j)$.

The problem of *clusterings of maximum spacing* is to find the partition that maximizes $D^*(C)$.

An efficient way to solve this problem is to consider the complete graph on V and for $e = (i, j)$, let $c(e) = d(i, j)$; ignore all the self loops and run Greedy-MST; delete the $l-1$ most expensive edges - or equivalently, the last $l-1$ edges added in the run of the algorithm - more formally, let $\{i_{p(1)}, i_{p(2)}, \ldots, i_{p(n-1)}\}$ be the indices of edges of the tree selected by Greedy-MST, T. It is easy to see that by deleting one edge of T we will have two trees. Similarly, by deleting $\{i_{p(n-l)}, \ldots, i_{p(n-1)}, \}$ we will have l connected components, $C_1, C_2, \ldots, C_l$.

Claim 5 $C = \{C_1, C_2, \ldots, C_l\}$ *is the solution of the l-clusterings of maximum spacing problem.*

Proof: First note that $D^*(C)$ is $c(e_{i_{p(n-l)}})$ or the $(l-1)$st most expensive edge of T. We prove the claim by contradiction, assume that $C' = \{C'_1, C'_2, \ldots, C'_l\}$ is another partition in which all C'_i's are non-empty and $D^*(C') > D^*(C)$. Since $C \neq C'$, there exists a cluster C_r with nodes $v_i \in C_r$ and $v_j \in C_r$ that are in two different clusters of C'. Say $v_i \in C'_s$ and $v_j \in C'_t$. Since v_i and v_j belong to the connected component C_r there must be a path P between them in T. Moreover, since all the edges in $\{i_{p(1)}, i_{p(2)}, \ldots, i_{p(n-l-1)}\}$ have cost smaller than the cost of $i_{p(n-l)}$, the cost of any edge in P is smaller than $D^*(C)$.

Consider path P in C', let u be the first node of P that does not belong to C'_s and w be the node comes just before u in P. Suppose $u \in C'_q$, clearly $D^*(C') \leq D(C'_s, C'_q) \leq c(w, u) < D^*(C)$ which is contradicting with the assumption that C' is the l-clusterings of maximum spacing.

■

We can also use MST to create a hierarchical clustering of a set. In biology, this tool is used to classify species, i.e. to build the "tree of life" or the phylogenetic tree. Solving this problem exactly is very hard but several approximations heuristics are used, one of which relies on minimum spanning trees. To build a "maximum likelihood" phylogenetic tree in this way, one can use the "closeness" of DNA sequences between different species as weights of a graph and build the MST of this graph. The phylogenetic tree can then be reconstructed by looking at where the nodes are located in the MST.