Discrete seminar report

Trees:

A tree is simply defined as a connected, acyclic graph. That is, there exists a path between any two vertices in the graph, and said graph contains no cycles. A forest is then defined to be any acyclic graph, it follows that every connected component of a forest is itself a tree.

Labelled Trees:

A labelled tree is a tree in which every vertex is assigned a unique integer from 1 to n, where n is the amount of vertices in the tree. Cayley's formula states that there are n^{n-2} distinct labelled trees with n vertices. We can prove this using Kirchoff's Matrix Tree Theorem:

Kirchoff's Matrix Tree Theorem states that given a simple labelled graph $G = (V, E)$ with vertices $V = \{v_1, \dots, v_n\}$, define the matrix $M = (m)_{ij}$:

$$(m)_{ij} = \begin{cases} -1 & \text{if } v_i v_j \in E \\ 0 & \text{if } v_i v_j \notin E \\ d_G(v_i) & \text{if } i = j \end{cases}$$

Then the number of spanning trees of G is any unsigned cofactor of M. We want to find the number of spanning trees with n vertices, so applying this result to the complete graph K_n will prove our result. So we construct M. Note that the degree of any vertex is $n-1$, and $v_i v_j$ is always an edge, since we are considering K_n:

$$M = \begin{pmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{pmatrix}$$

Notice that for an undirected graph, this matrix is always symmetric. We will find the cofactor at $1, 1$ for convenience:

$$C_{11}(M) = \det(M_{(1,1)})$$

Where $M_{(1,1)}$ is the matrix obtained by deleting the first row and column of M. This will be an $(n-1) \times (n-1)$ matrix, denoted as M_1. So the answer to our question is equal to $\det(M_1)$. Note that the elementary row operation $A_{rs}(\mu)$ does not change the determinant of a matrix, where $A_{rs}(\mu)$ represents multiplying row r by μ and adding it to row s. This is because the determinant of the elementary matrix that represents this operation is 1, so the determinant of the matrix after an arbitrary amount of these elementary row operations is:
$\det(E_1 E_2 \dots E_n M_1) = \det(E_1) \dots \det(E_n) \det(M_1) = 1 \times \dots \times 1 \times \det(M_1) = \det(M_1)$, so we are free to manipulate M_1 in this way without changing its determinant. We choose elementary row

operations of the form $A_{1\,i}(-1)$ for $i \in \{2, 3, \dots, n-2, n-1\}$ so that our matrix is now of the form:

$$\begin{pmatrix} n-1 & -1 & -1 & \dots & -1 \\ -1 & n-1 & -1 & \dots & -1 \\ -1 & -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & \dots & n-1 \end{pmatrix} \mapsto \begin{pmatrix} n-1 & -1 & -1 & \dots & -1 \\ -n & n & 0 & \dots & 0 \\ -n & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -n & 0 & 0 & \dots & n \end{pmatrix}$$

Similarly, the elementary column operation of similar form does not change the determinant by the same line of reasoning, so we perform $A_{i\,1}(1)$ for $i \in \{2, 3, \dots, n-2, n-1\}$, adding each column i to the first, multiplied by 1:

$$\begin{pmatrix} n-1 & -1 & -1 & \dots & -1 \\ -n & n & 0 & \dots & 0 \\ -n & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -n & 0 & 0 & \dots & n \end{pmatrix} \mapsto \begin{pmatrix} 1 & -1 & -1 & \dots & -1 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & n \end{pmatrix}$$

Denote this new matrix as M_1'. We have that $\det(M_1) = \det(M_1')$, so expand M_1' along the first row. Notice that each $(M_1')_{(1,i)}$ for $i \in \{2, 3, \dots, n-2, n-1\}$ contains a row of zeros, so expanding along this row shows that each of these have a determinant of zero. Therefore we only need to worry about the term at $1, 1$, that is $(M_1')_{(1,1)} = M_2'$:

$$\det(M_1) = 1 \times \det(M_2')$$

Notice that M_2' is simply n times the $n-2$ dimensional identity matrix, so we have that:

$$\det(M_1) = \det(nI_{n-2}) = \det \begin{pmatrix} n & 0 & 0 & \dots & 0 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & n \end{pmatrix}$$

This is simply equal to the product of the diagonal elements, that is:

$$\det(M_1) = \prod_{i=1}^{n-2} nI_{n-2} = \prod_{i=1}^{n-2} n = n^{n-2}$$

So we have proven the result as required, using Kirchoff's Matrix Tree theorem, that there indeed are n^{n-2} distinct labelled trees with n vertices.

Spanning trees and MSTs:

The spanning tree of a graph is a subgraph that shares the same vertex set as the graph, which is also a tree. That is, every vertex in the graph is in the tree, the tree spans the graph. For any weighted graph, the minimum spanning tree (MST) is defined to be the spanning tree of least total weight.

Prim's Algorithm:

One way to find the MST of a weighted graph is through Prim's algorithm, the procedure of which is as follows:

> From any weighted, connected graph $G = (V, E, \Phi_G)$, choose a vertex $v_1 \in V$ and include this in $T_1 = (\{v_1, \emptyset\})$
> Find the edge of least weight in G that connects any vertex in $T_k = (V_{T_k}, E_{T_k})$ to a vertex not currently in T_k, but in G. That is the least weighted edge $e \in E \setminus E_{T_k}$ such that $e = v_i v_j$ where $v_i \in V_{T_k}$ and $v_j \in V \setminus V_{T_k}$
> Add such edge, e, and such vertex v_j to the tree to construct T_{k+1}
> Repeat the previous two steps until $V_{T_k} = V$

This algorithm naturally avoids creating cycles, since edges are only added if they connect a vertex in T_k to a vertex not in T_k. Since this algorithm only terminates when $V_{T_k} = V$, both G and T share a vertex set, so the resulting tree must span.

Suppose that the tree produced by this algorithm, T, is not optimal and let T' be the MST. Then we have $w(T) \geq w(T')$ and that there must exist some edge e in T that is not in T' because if they shared the edge set they must be the same tree (since we know trees from Prim's algorithm span). Let V_e be the set of vertices added before e was added, then there must exist an edge f in T' that connects a vertex in V_e to a vertex in $V \setminus V_e$. Form the tree $T'' = T' - f + e$, that is the tree formed when replacing edge f with edge e in T'. Note that $w(f) \geq w(e)$ since e was the edge used to make T' optimal and T not optimal. Therefore T'' is also an MST. So we have $w(T'') = w(T') - w(f) + w(e) \leq w(T')$, and since both are MSTs we must have $w(T'') = w(T')$. One can repeat this process for all edges in T that are not in T'. Once this has been done for all of such edges, the tree "T''" is the same as T, giving the result $w(T) \leq \cdots \leq w(T'') \leq w(T')$. This implies $w(T) \leq w(T')$, but since we have $w(T) \geq w(T')$, we can conclude $w(T) = w(T')$, therefore T is indeed a minimum spanning tree.

Kruskal's algorithm:

Another way to find the MST of a connected graph is by applying Kruskal's algorithm. The procedure for this algorithm is much simpler than Prim's with the only rule being:

> Add the edge of least weight that does not cause a cycle

This will terminate when it is not possible to add an edge without creating a cycle. Assume that the algorithm terminates without adding a vertex. Since the original graph was connected, there exists a path between any two vertices and therefore there must be a an edge connecting any vertex to the rest of the graph. That is, $d_G(e) \geq 1$ for all $e \in E$. Since the algorithm terminated without adding a vertex, adding the edge that connects the vertex must create a cycle. In this scenario, the vertex would be connected to some other vertex in the tree, which is a contradiction. Therefore this

process always produces a tree, since we have shown connectedness is preserved, and the algorithm preserves acyclicity by nature. Due to the simpler algorithm, Kruskal's algorithm is often preferred when constructing a computer model of graphs.

The Travelling Salesman Problem:

The Travelling Salesman Problem (TSP) is a classic problem in decision mathematics. This involves finding the path of least weight in a weighted network that starts at a given vertex, traversing to all other vertices in the graph and returning to the starting vertex. There does not exist an efficient algorithm to find the optimal solution for the TSP, so upper and lower bounds are often used to find an interval that the optimal solution lies in:

$$w(\text{lower bound}) \leq w(\text{optimal solution}) \leq w(\text{upper bound})$$

From this, it is obvious that a lower bound is better the larger it is, and an upper bound is better the lower it is. This is because the interval on which the optimal solution lies will be smaller. MSTs are very useful when finding these upper and lower bounds:

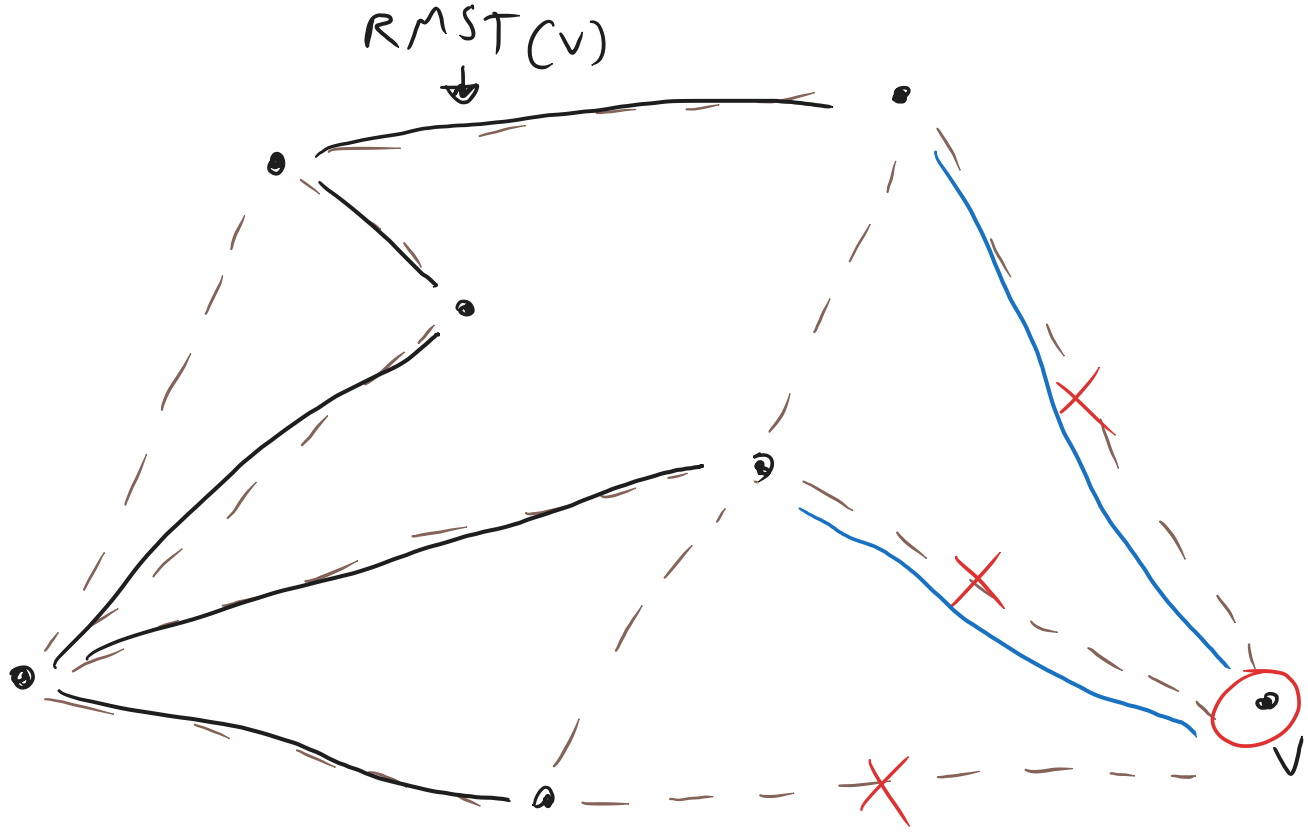
TSP-lower bound:

Since the MST of a weighted, connected graph does not contain a cycle, and is defined to be the spanning tree of least weight, the weight of a graph's MST is itself a lower bound to the TSP. However this can be refined by using the Minimum Connector / Edge removal Residual Minimum Spanning Tree algorithms. This algorithm is as follows:

> Remove any vertex $v \in V$ from the connected, weighted graph $G = (V, E)$, including any edges in which it appears
> Find the Residual Minimum Spanning Tree, $RMST(v)$, having done this
> Add the removed vertex back, via the two edges of least weight,
> $e_1 = \min\{v\,\Phi(v)\}, e_2 = \min\{v, \Phi(v)\} \setminus \{e_1\}$
> Take the lower bound to be the sum of the weights of the RMST and the two edges,
> $w(LB_v) = w(RMST(v)) + w(e_1) + w(e_2)$
> Repeat this for every vertex in V, taking the best lower bound to be the greatest lower bound found

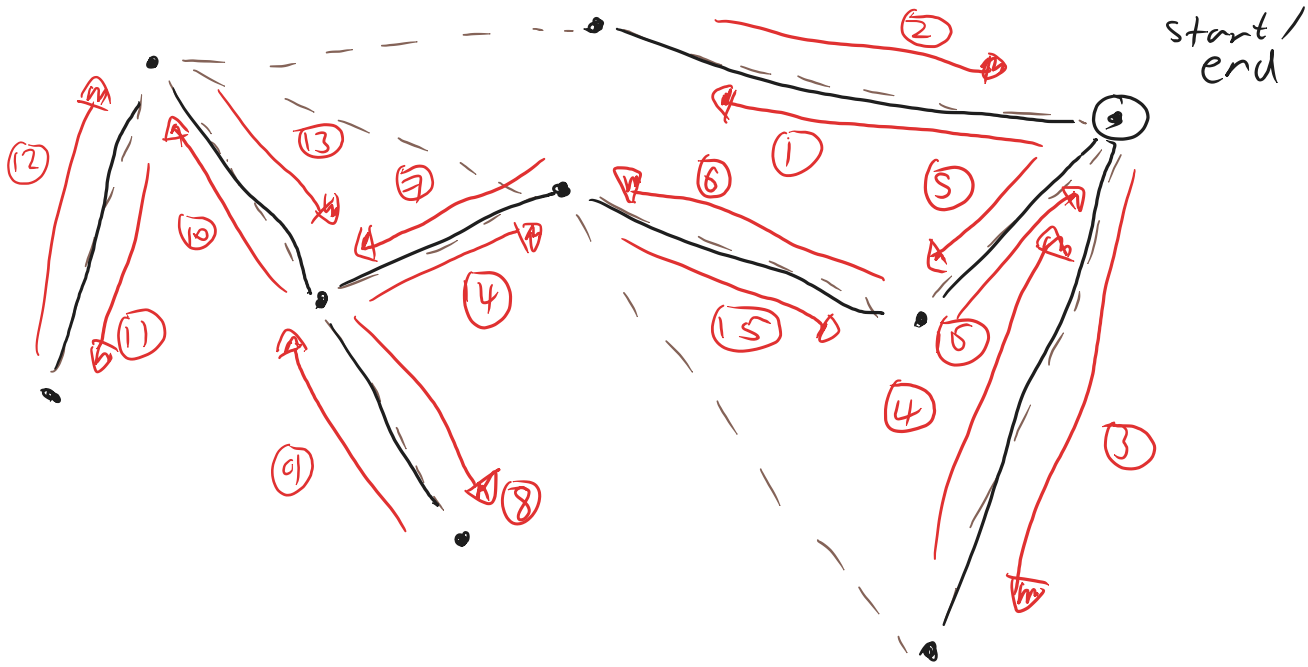
Take the optimal solution, OS, and a lower bound, LB. Now remove the vertex from OS that was removed in LB, including the edges in which it appears. These edges removed correspond to the minimum two edges we add when calculating LB. Since in our algorithm, we consider the two edges of least weight, these two edges make $w(LB)$ less than or equal to $w(OS)$. Therefore we have that the lower bounds produced by this algorithm are indeed lower bounds.

Here is an example of a lower bound found using this algorithm, where edge weights are omitted. The RMST is shown with black edges, and the two edges of least weight used to reconnect v are shown in blue:



TSP-upper bound:

A simple upper bound for the TSP is found by doubling the weight of the graph's MST. While this tends not to be a very good upper bound, it is one nonetheless and will always work. This can be visualised in the example, where edge weights are omitted:

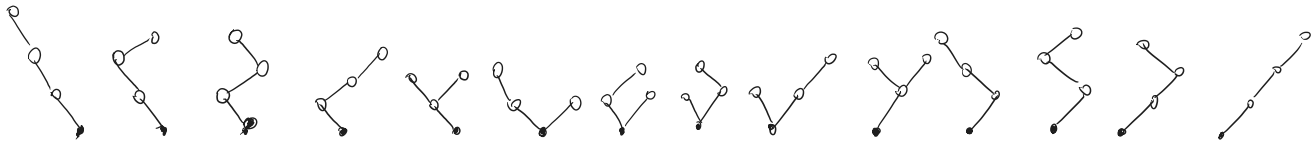


Upper bounds found in this manner can be improved by looking for shortcuts.

Rooted Binary trees:

A rooted binary tree is an unlabelled tree with one unique vertex called the root. From the root, vertices branch upwards either to the left or the right. Subsequent vertices can branch in a similar

manner, with the degree of any vertex being at most 3. The distinct binary trees with 4 vertices can be seen here:



Observe that each branch from the root is either empty, or can itself be considered a binary tree (ignoring any vertices below it). It is therefore possible to formulate the number of distinct binary trees with n vertices:

$$b_n = b_0 b_{n-1} + b_1 b_{n-2} + \cdots + b_{n-2} b_1 + b_{n-1} b_0$$

With $n \geq 1$ and $b_0 = 1$. Each $b_i b_j$ term represents the product of the number of ways to create a binary tree from i vertices, initially branching to the left of the root and the number of ways to create a binary tree from j vertices, initially branching to the right of the root. It follows that $n = i + j$.

To determine the number b_n explicitly, we define the generating function:

$$b(x) = b_0 + b_1 x + b_2 x^2 + \cdots = \sum_{n=0}^{\infty} b_n x^n$$

Consider the square of this function:

$$b(x)^2 = (b_0 + b_1 x + b_2 x^2 + \ldots)^2 = b_0 b_0 + (b_0 b_1 + b_1 b_0) x + \ldots$$

We can see that each coeffient of x_n here is equal to b_{n-1}, except for b_0, so lets fix that:

$$x b(x)^2 + b_0 = b_0 + b_1 x + b_2 x^2 + \cdots = b(x)$$

This gives us a quadratic in $b(x)^2$:

$$b(x)^2 - \frac{1}{x} b(x) + \frac{1}{x} b_0 = 0$$

Solving the quadratic:

$$\left(b(x) - \frac{1}{2x}\right)^2 - \frac{1}{4x^2} + \frac{1}{x} b_0 = 0 \implies \left(b(x) - \frac{1}{2x}\right)^2 = \frac{1}{4x^2} - \frac{1}{x} b_0$$

We can use the fact that $b_0 = 1$ to give:

$$b(x) = \frac{1}{2x} \pm \sqrt{\frac{1}{4x^2} - \frac{1}{x}} = \frac{1}{2x} \pm \sqrt{\frac{1 - 4x}{4x^2}} = \frac{1 \pm \sqrt{1 - 4x}}{2x}$$

From here, we use the extended binomial coefficients to rewrite the square root:

$$b(x) = \frac{1 \pm \sum_{k=0}^{\infty} \binom{\frac{1}{2}}{k} (-4x)^k}{2x} = \frac{1 \pm (1 - 2x + \ldots)}{2x}$$

It only makes sense to have positive coefficients of each x, so we take the negative multiple of the

sum, as this works out to give positive coefficients:

$$b(x) = \frac{1 - (1 - 2x + \dots)}{2x} = 1 + x + 2x^2 + \dots = b_0 + b_1 x + b_2 x^2 + \dots$$

Making our closed form expression:

$$b(x) = \frac{1}{2x} - \frac{1}{2x} \sum_{k=0}^{\infty} \binom{\frac{1}{2}}{k} (-4x)^k$$

Now since the $k = 0$ term is 1 in the summand, the $\frac{1}{2x}$ out the front is cancelled, now rewriting the summand and putting the factor of $-\frac{1}{2x}$ inside the summand gives:

$$b(x) = -\frac{1}{2x} \sum_{k=1}^{\infty} \binom{\frac{1}{2}}{k} (-4)^k x^k = \sum_{k=1}^{\infty} \binom{\frac{1}{2}}{k} (-1)^{k+1} 2^{2k-1} x^{k-1}$$

Now we can adjust the summing variable so that the powers of x are of form x^k, which also makes the sum start at zero which is always nice:

$$b(x) = \sum_{k=0}^{\infty} (-1)^{k+2} 2^{2k+1} \binom{\frac{1}{2}}{k+1} x^k$$

Now, to find b_n, look at the coefficient of x^n in this sum:

$$b_n = (-1)^n 2^{2n+1} \binom{\frac{1}{2}}{n+1} = (-1)^n 2^{2n+1} \frac{\frac{1}{2} \times (\frac{1}{2} - 1) \times (\frac{1}{2} - 2) \times \dots \times (\frac{1}{2}}{(n+1)!}$$

Now we borrow 2^{n+1} from the 2^{2n+1} term to multiply every term in the numerator by 2 to give:

$$b_n = (-1)^n 2^n \frac{1 \times (1 - 2) \times (1 - 4) \times \dots \times (1 - 2n)}{(n+1)!}$$

We can also multiple every term in the numerator by -1 to get rid of the $(-1)^n$ term:

$$b_n = 2^n \frac{(2n - 1) \times (2n - 3) \times \dots \times (2 - 1)}{(n+1)!}$$

Notice that the numerator is similar to $2n!$, missing all of the even terms however, so we can reintroduce these, making sure to balance them by dividing by every even term also:

$$b_n = 2^n \frac{(2n) \times (2n - 1) \times \dots \times (2 - 1)}{(2n) \times (2n - 2) \times \dots \times (3 - 1)(n+1)!}$$

Now factoring a 2 from all n terms in the denominator cancels the 2^n, as well as making the denominator the same as $n!$:

$$b_n = \frac{2^n}{2^n} \frac{(2n)!}{n \times (n-1) \times \dots \times (1)(n+1)!} = \frac{(2n)!}{n!(n+1)!} = \frac{1}{n+1} \frac{(2n)!}{n!n!}$$

We can manipulate the last term here to give the result:

$$b_n = \frac{1}{n+1}\frac{(2n)!}{(2n-n)!n!} = \frac{1}{n+1}\binom{2n}{n}$$

This is the number of rooted binary trees with n vertices, also known as the Catalan numbers.

Declaration.

I have read and understood these guidelines. I also understand that any plagiarism could have serious consequences for my mark, and for my academic career.

Name: Oscar Pearce

Signed:  Date 16/02/24