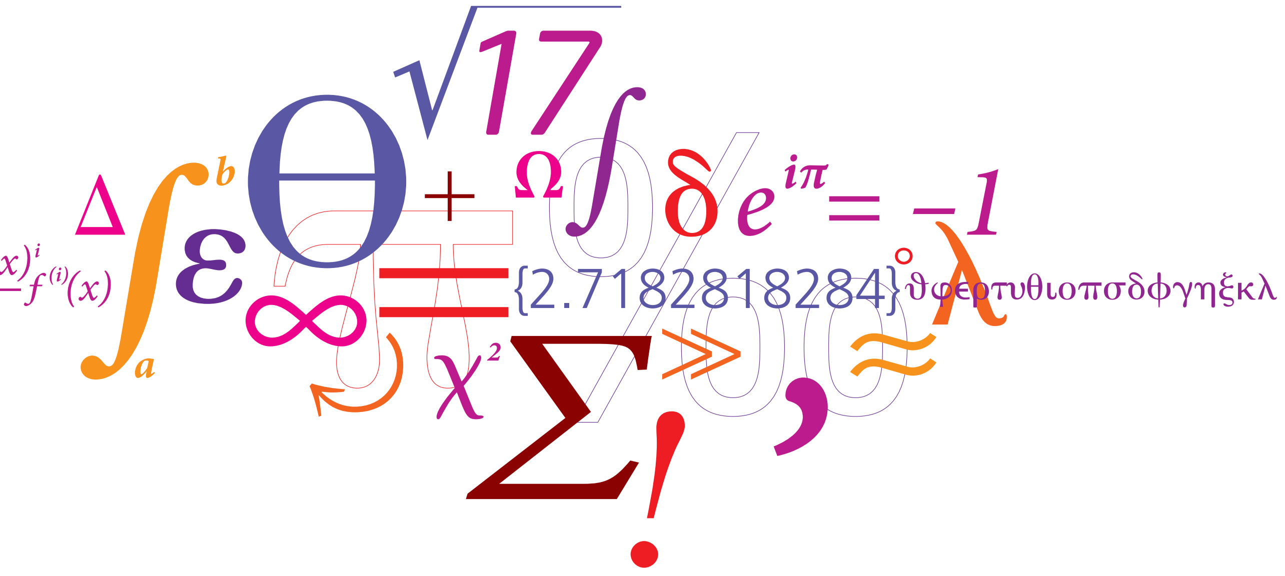
Technical University of Denmark

Bachelor Thesis, Spring 2011

Connected Graphs with Fewest Spanning Trees

A short survey on counting spanning trees and some new lower bounds on the number of spanning trees in cubic graphs.



DTU Mathematics
Department of Mathematics

Author
David Kofoed Wind s082951

Supervisor
Prof. Carsten Thomassen

Abstract

We present a known lower bound on the number of spanning trees in cubic graphs and prove a new simpler result, giving a lower bound on the number of spanning trees in cubic multigraphs. Then we give a new non-trivial lower bound on the number of spanning trees in cubic 2-connected graphs and finally we consider 3-connected graphs, and formulate some conjectures based on computational experiments.

Preface

This thesis was prepared at the Department of Mathematics at the Technical University of Denmark in the period from February 2011 to June 2011. It corresponds to 15 ECTS credits, and it was part of the requirements for acquiring an B.Sc. degree in engineering.

I would like to thank my supervisor Carsten Thomassen for careful guidance, patience and support, without him this thesis would not be what it is. I would also like to thank my parents for supporting my education, and my girlfriend for listening to my mathematical rambling.

I would like to thank Jesper Gielov, Christine Lindeblad, Rasmus Malthe Jørgensen and Pia Wind for proofreading. Additionally, I would like to thank Hjalte Wedel Vildhøj for working with me on this subject before the project, and I wish to thank Rasmus Malthe Jørgensen, Christine Lindeblad and Jakob Møller Andersen for a helping hand when the math got tricky.

David Kofoed Wind

Contents

Chapter 1

Introduction

Graph Theory is the study of *graphs*, a mathematical structure which models pairwise relations between objects, and is in that context, a very abstract and general setting. The focus in this thesis is on counting spanning trees in connected graphs, a subject in combinatorial graph theory. The number of spanning trees plays a crucial role in Kirchhoff's classical theory of electrical networks, for example in computing driving point resistance [**?**], and in random walks in graphs [**?**].

In the second chapter, some general graph theory is described, and methods for counting spanning trees are introduced.

In the third chapter, we present a known lower bound on the number of spanning trees in cubic graphs, originally presented by Kostochka [**?**] and prove a new slightly simpler result, giving a lower bound on the number of spanning trees in cubic multigraphs.

In the fourth chapter, we give a new non-trivial lower bound on the number of spanning trees in cubic 2-connected graphs.

Finally in the last chapter, we consider 3-connected graphs, and formulate some conjectures based on computational experiments.

Chapter 2

Notation and Theory

We will give a brief introduction to the notation of this thesis, and introduce some of the elementary results and theory. Some of this notation and theory is presented in the same way as in [**?**] and [**?**].

2.1 Notation and General Theory

We define a graph G as a pair (V, E) of sets V and E, where V is called the vertex set, and E is called the edge set. $E \subseteq V^2$, meaning that an edge $e \in E$ is a 2-element subset $\{v_i, v_j\}$ of V. When talking about the vertex set or edge set of a graph G, the sets will be denoted $V(G)$ and $E(G)$. We will denote the number of vertices in a graph G by $v(G)$ and the number of edges by $e(G)$. We let uv denote the edge joining vertex u and vertex v. In general, we will not allow multiple edges between a pair of vertices, but a graph where multiple edges are allowed, will be denoted a multigraph. We will ignore loops, that is edges from a vertex to itself.

A path P in a graph G is a sequence of vertices $v_1, v_2, \ldots, v_k$, such that $v_i v_{i+1} \in E(G)$ for $i = 1, \ldots, (k-1)$, and where $v_i \neq v_j$ when $i \neq j$. For a vertex v, we define its neighbours $N(v)$ as the vertices joined to v by an edge. For a subgraph $G' \subseteq G$, we define a multiset $N(G')$ of its neighbours in G as $\bigcup_{v \in V(G')} [N(v) - V(G')]$. For a subgraph $G' \subseteq G$, we define the neighbours of G' as the vertices in G joined to a vertex in G'. For a vertex v, we define the degree of v as the number of edges (again ignoring loops) having v as endpoint. A cycle C in a graph G, is a closed path $v_1, \ldots, v_i, \ldots, v_1$, where the vertices in C except v_1 are pairwise distinct. A cycle chord of a cycle C is an edge xy, such that $xy \notin E(C)$ and such that $x \in V(C)$ and $y \in V(C)$. An induced cycle C is a cycle with no cycle chord. A cycle C in a connected graph G is said to be non-separating if and only if $G - V(C)$ is connected.

For a vertex v with two neighbours x and y, we define *lifting* the edges of v, as replacing xv and yv by xy.

For a graph G, and a set S of vertices and edges in G, we will let $G - S$ denote the graph obtained by removing the elements of S from G. Removing a vertex from a graph, is defined as removing the vertex, and all edges incident to it. For a graph G, and a set of vertices and edges S, we will let $G + S$ denote the graph obtained by adding the elements of S to G. For a graph G, and an edge $e \in E(G)$, we denote a contraction of e by G/e. A contraction of an edge corresponds to glueing its ends together, and removing the loops created. For a graph G and a set of vertices $S \subseteq V(G)$, we denote a contraction of S as G/S, and define it as contracting every edge $e \in E(G)$ which is joining two vertices in S.

A graph G is said to be k-regular, if every vertex in G has degree k, that is, every vertex has k edges incident to it. A 3-regular graph is called cubic. A cubic graph has an even number of vertices.

A graph G is said to be k-connected, if the removal of any $(k - 1)$ vertices will not disconnect G, but there exist a set S of k vertices such that $G - S$ is disconnected.

An edge e in a graph G is called a bridge, if the removal of e increases the number of connected components of G. We define a block in a graph as either a bridge, or a maximally 2-connected subgraph.

A graph G is said to have girth k, if the smallest cycle in G has k vertices.

To describe the growth rate of a function, we will introduce the so-called **Big O-notation** [**?**]. We define Big-O-notation as follows

Definition 2.1 *We say that a function $g(n)$ is $O(f(n))$ if and only if there exists two constants c and n_0 such that $g(n) \leq cf(n)$ for all $n \geq n_0$.*

If we have two functions $f(n)$ and $g(n)$, and we want to compare their growth rates, we consider the limit

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

if this limit is equal to 0, then $g(n)$ grows asymptotically faster than $f(n)$. If the limit is equal to ∞, then $f(n)$ grows asymptotically faster than $g(n)$. If the limit is equal to a constant $c > 0$, then the two functions are defined to be asymptotically equal, since they only differ by a multiplicative factor.

2.2 Spanning Trees

A *spanning tree* in a graph G is a spanning subgraph of G, which is a tree. We define a spanning tree formally as follows.

Definition 2.2 *A spanning tree T of a graph $G = (V, E)$ is a graph $T = (V, E')$ such that T is a tree and $E' \subseteq E$.*

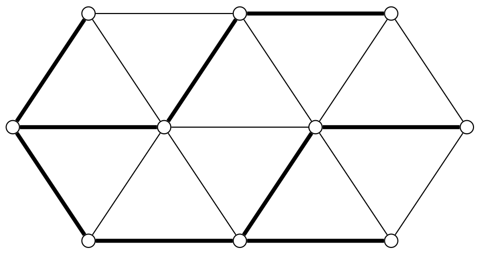


Figure 2.1: A spanning tree (emphasized with bold edges) in a graph.

2.2.1 Counting Spanning Trees

The number of different spanning trees in a graph G turns out to be a very useful number leading to interesting results and applications. We denote the number of spanning trees in a graph G by $\tau(G)$ and in this section, we will investigate different methods for calculating $\tau(G)$. If G itself is a tree then $\tau(G) = 1$, and if G is disconnected, then $\tau(G) = 0$. As an example, the 16 spanning trees of K_4 are shown in **??**.

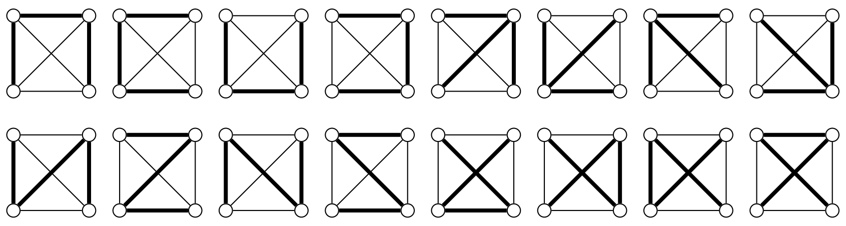


Figure 2.2: The 16 different spanning trees in K_4.

Theorem 2.1 *The number of spanning trees $\tau(G)$ in a graph G containing an edge e is equal to $\tau(G/e)$. The number of spanning trees in a graph G* **not** *containing an edge e is equal to $\tau(G - e)$.*

A useful Lemma is

Lemma 2.1 *If G is a connected graph, then for any $e \in E(G)$, there is a spanning tree containing e.*

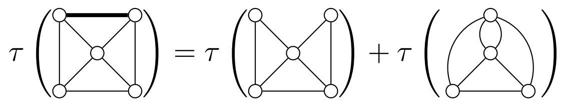
This follows easily from **??**, since contracting an edge in a connected graph does not disconnect the graph.

Contraction-Deletion

Theorem 2.2 (Contraction-Deletion) *The number of spanning trees in a graph G, is the number of spanning trees in G where some edge e is deleted, plus the number of spanning trees in G where e is contracted.*

$$\tau(G) = \tau(G - e) + \tau(G/e)$$

One way to compute the number of spanning trees in a graph, is to recursively degenerate the graph by removing and contracting edges. In this way, the number of spanning trees of a graph G is reduced to a sum of the number of spanning trees in small graphs which can be computed by hand. A Contraction-Deletion degeneration could look like this



The Matrix-Tree Theorem

Another way to compute the number of spanning trees in a graph is using linear algebra. By a theorem of Gustav Robert Kirchoff from 1847, the number of spanning trees can be computed by finding the cofactor of a special matrix called the Laplacian matrix.

Degree Matrix The degree matrix D of a graph, is a diagonal matrix with the vertex degrees in the diagonal. As an example, consider the labelled graph in **??**.

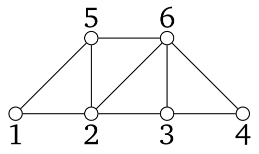


Figure 2.3: A graph with 6 vertices. We will compute the number of spanning trees of this graph by the matrix-tree theorem.

which has the degree matrix

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Adjacency Matrix The adjacency matrix A of a graph, is a $(0,1)$-matrix, where 1's represent adjacent vertices. A has a row for each vertex, and a column for each vertex. If a vertex v is connected to a vertex u, the entry in row v, column u is 1, and so is the entry in row u, column v. The graph in **??** has the adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Laplacian Matrix The Laplacian Matrix Q of a graph is the difference between its degree matrix D and its adjacency matrix A.

$$Q = D - A$$

Which for the graph in **??** gives

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 4 & -1 & 0 & -1 & -1 \\ 0 & -1 & 3 & -1 & 0 & -1 \\ 0 & 0 & -1 & 2 & 0 & -1 \\ -1 & -1 & 0 & 0 & 3 & -1 \\ 0 & -1 & -1 & -1 & -1 & 4 \end{bmatrix}$$

Now we can state the final theorem

Theorem 2.3 (Kirchoff's Matrix-Tree Theorem) *The number of spanning trees in a simple graph G, is the absolute value of any cofactor of the Laplacian Matrix of G.*

Where a cofactor of Q can be obtained by removing an arbitrary row, and an arbitrary column from Q and then computing the determinant. For our example, we find a cofactor by removing the last row and column,

$$\begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ -1 & -1 & 0 & 0 & 3 \end{bmatrix}$$

and then taking the determinant

$$\tau(G) = \det\left(\begin{bmatrix} 2 & -1 & 0 & 0 & -1 \\ -1 & 4 & -1 & 0 & -1 \\ 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 2 & 0 \\ -1 & -1 & 0 & 0 & 3 \end{bmatrix}\right) = 55$$

For multigraphs, the definition of Q is slightly different [**?**].

2.3 Recurrence Relations

If we have a homogeneous linear recurrence relation of the form

$$a_n = \sum_{i=1}^{k} c_i a_{n-i}$$

involving k terms, and we want to find a closed form for a_n - that is, a function - we can solve the recurrence relation. We guess that a_n is on the form α^n, and substitute this into the recurrence relation to obtain

$$\alpha^n = \sum_{i=1}^{k} c_i \alpha^{n-i}$$

and we divide by α^{n-k}, giving us

$$\alpha^k = \sum_{i=1}^{k} c_i \alpha^{k-i}$$

This is a polynomial of degree k with distinct roots $\{\alpha_1, \ldots, \alpha_k\} \in \mathbb{R}^k$, and now a_n can be written as

$$a_n = \sum_{i=1}^{k} \beta_i \alpha_i^n$$

where β_1 to β_k is constants which can be found by using k different initial conditions for the recurrence.

In general, solving a recurrence relation can be much more difficult. We have only described homogeneous linear recurrence relations, and we assumed that the roots $\{\alpha_1, \ldots, \alpha_k\}$ were distinct real numbers - but this is all we need for the recurrence relations involved in this project.

Chapter 3

Cubic Graphs

In this chapter, we will consider cubic graphs, that is graphs where every vertex has degree 3. We will look at some graph classes, and find a lower bound for the number of spanning trees in graphs of each class.

3.1 Cubic Multigraphs

We consider cubic multigraphs, that is cubic graphs where we allow multiple edges between a pair of vertices. We introduce the following Lemma, which has previously been described, in a slightly different form, in [**?**].

Lemma 3.1 *Let G be a cubic connected graph, then G contains a non-separating induced cycle C. Additionally, if G has girth ≥ 5, C can be chosen such that all neighbours of C are pairwise distinct.*

Proof Choose C to be a cycle in G maximizing the size of the largest connective component of $G - V(C)$. We will start by showing that $G - V(C)$ is connected. Assume that $G - V(C)$ is disconnected and has connective components A and B (and maybe more), where B is the largest connective component. This is illustrated in **??**.

Since A and B are different components, there are no edges between them. Now we have two cases.

A **contains a cycle** C'**:** Consider $G - V(C')$. Since G is connected, at least one vertex v on C is joined to a vertex in B, so $G - V(C')$ has a connective component containing B and v, which contradicts that C was the cycle maximizing the size of the largest connective component of $G - V(C)$. Thus A can not contain a cycle.

A **is a forest:** Since A does not contain a cycle, it must be a forest. Now consider a leaf l in A. Since G is cubic and l is a leaf in A and there are no edges

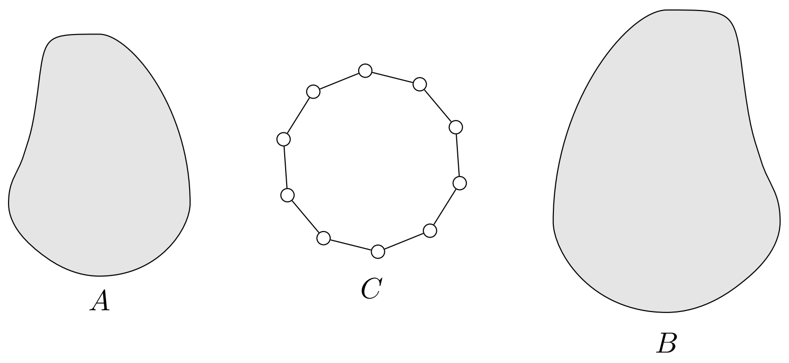


Figure 3.1: The two components A and B which would be created when removing $V(C)$ from G under the assumption that C is separating. All vertices on C have degree 3, so some of them are connected to A, some to B and maybe some of them to each other.

between the connective components of $G - V(C)$, l must have two edges going to vertices on C. Denote these vertices joined to l by x and y. At least one vertex on C is connected to B, denote this vertex u. This is illustrated in **??**.

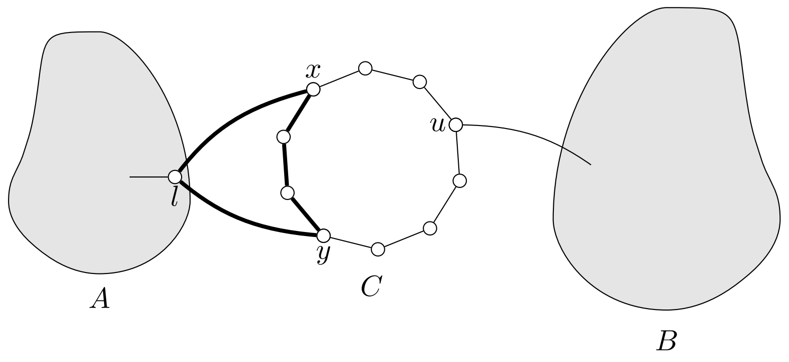


Figure 3.2: The leaf l in A, joined to x and y on C.

There are two paths from x to y on C. Consider the path from x to y on C not using u and extend it to a cycle by including lx and ly. Denote this cycle C', and look at $G - V(C')$, which has a connected component containing B and u. This contradicts that C was the cycle, which removal was maximizing the size of the largest connective component of $G - V(C)$. Thus C is non-separating.

Now we show that C can be chosen to be non-separating and induced. Consider a non-separating cycle C, such that $G - V(C)$ is maximal. Assume that C is not induced, thus C must have a chord xy. Singe G is cubic, there must be a vertex on C, different from x and y joined to a vertex in $G - V(C)$, denote this vertex u. This is illustrated in **??**.

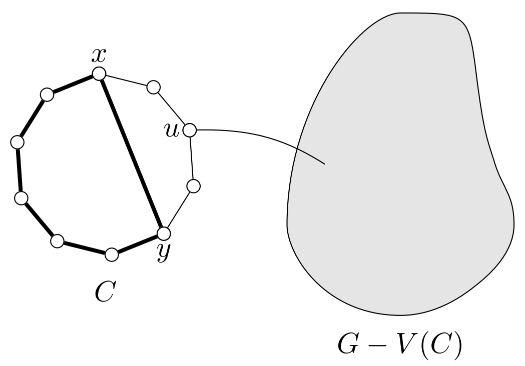


Figure 3.3: A chord xy in C, and a cycle (emphasized in bold) which removal leads to a contradiction.

Consider the cycle using xy and the path between x and y on C not containing u, denote this cycle C'. Since $V(C') \subset V(C)$, C' is non-separating. $G - V(C')$ has a connected component containing $G - V(C)$ and u, contradicting that C was the cycle, which removal maximizes the size of $G - V(C)$. Thus, C is induced.

Now assume G has girth ≥ 5, then we show that it is possible to choose a non-separating induced cycle C such that all neighbours of C are pairwise distinct. Consider a non-separating induced cycle C, such that $G - V(C)$ is maximal. Assume that C contains two vertices x and y which have a common neighbour z not on C. Since G has girth ≥ 5, one of the paths from x to y on C contains more than one vertex, denote two of the vertices on this path u, v. Since C is induced, both u and v are joined to vertices in $G - V(C)$. This is illustrated in **??**.

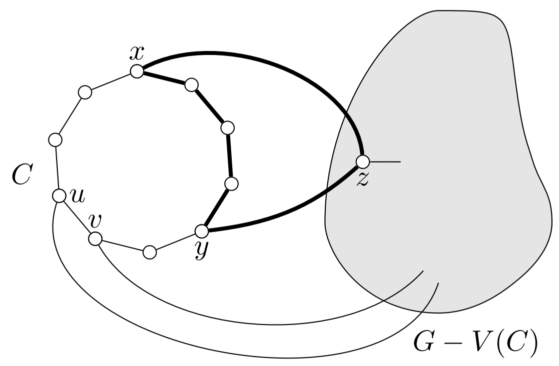


Figure 3.4: Since G has girth ≥ 5, one of the paths from x to y on C contains more than one vertex, in this case u and v.

Consider the cycle using xz, yz and the path from x to y on C not containing either u or v, denote this cycle C'. Since C is induced and G is cubic, C' is also induced.

Since z is a leaf in $G - V(C)$, and since every vertex on C not contained in C' is connected to vertices in $G - V(C')$ through u and v, C' is non-separating. Since both u and v are joined to vertices in $G - V(C)$, this contradicts that C was the cycle, which removal maximizes the size of $G - V(C)$. Thus, all neighbours of C are different. ■

We will use **??** in the proof of **??**.

Theorem 3.1 *For any cubic, connected multigraph G with $n > 2$ vertices, $\tau(G) \geq 3 \cdot 2^{n/2}$.*

Proof We do the proof by induction over the number of vertices n.

Basis of the induction All cubic graphs on ≤ 6 vertices is used as basis of the induction. These are listen in **??**.

Intuition We will assume that all graphs on k vertices, where $k < n$, have at least $3 \cdot 2^{k/2}$ spanning trees. Then we will consider some cubic graph G on n vertices. Now we want to show that G has at least $3 \cdot 2^{n/2}$ spanning trees. To show this, we will perform some operation on G to create G', reducing the number of vertices. Now we have, by induction, a lower bound on the number of spanning trees in G'. Then we take every spanning tree in G', and extend it to a number of different spanning trees in G, thus showing that G has at least $3 \cdot 2^{n/2}$ spanning trees.

Induction step Let G have n vertices. Our induction hypothesis states, that all graphs on $k < n$ vertices, have at least $3 \cdot 2^{k/2}$ spanning trees. Now choose a cycle C in G in the following way (possible by **??**).

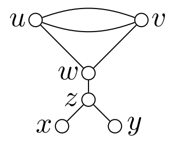
G **has girth 2:** Let C be any 2-cycle

G **has girth 3:** Let C be any 3-cycle

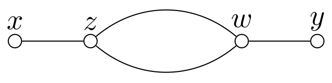
G **has girth 4:** Let C be a non-separating induced 4-cycle

G **has girth $\geq$ 5:** Let C be a non-separating induced cycle such that all neighbours of C are different.

Case 1 - $v(C) = 2$: Then C looks like one of the two cases in **??**.



(a) Subcase 1.1



(b) Subcase 1.2

Figure 3.5: The two different types of 2-cycles

If we have Subcase 1.1, we create $G' = G - \{u, v, w, z\} + xy$, this operation is shown in **??**, and is denoted Operation 1.

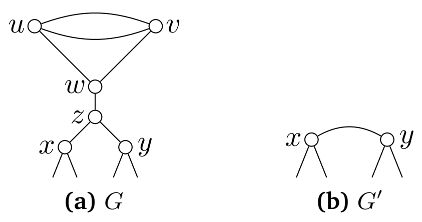
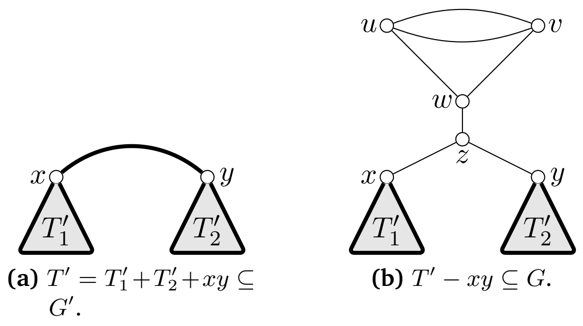


Figure 3.6: Operation 1, used when G has a 2-cycle as in Subcase 1.1.

Operation 1 reduces the number of vertices in G by 4, so we want to show, that if T' is a spanning tree in G', and we use Operation 1 backwards, then we can extend T' to at least 4 spanning trees in G, since this implies

$$\tau(G) \geq 4\tau(G') \geq 4 \cdot 3 \cdot 2^{\frac{n-4}{2}} = 4 \cdot 3 \cdot \frac{2^{n/2}}{4} = 3 \cdot 2^{n/2}$$

We need to consider the case where $xy \in E(T')$ illustrated in **??**, and the case where $xy \notin E(T')$ illustrated in **??**. When $xy \in E(T')$, we apply Operation 1 in reverse, removing xy from T'. To extend T' to a spanning tree T in G, we need both xz and zy to make T connected. There are two ways of choosing an edge in C, and for each of those, we have to chose either uw or vw, thus giving the 4 needed spanning trees.



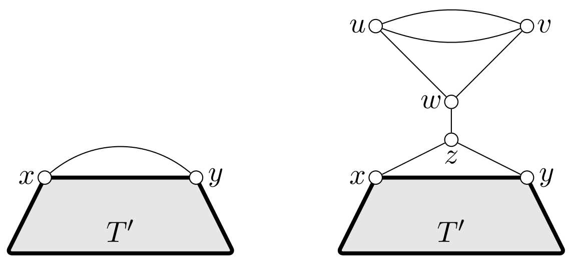
x
y
T_1'
T_2'
(a) $T' = T_1' + T_2' + xy \subseteq G'$.
u
v
w
z
x
y
T_1'
T_2'
(b) $T' - xy \subseteq G$.

Figure 3.7: The case when $xy \in T'$. When $xy \in T'$, note that $T' - xy$ will be disconnected with connective components T_1' and T_2', and that xz and zy need to be included to make T connected.

When $xy \notin E(T')$, and we apply Operation 1 in reverse, we do not disconnect T'. To extend T' to a spanning tree T in G, we can only use one of xz and zy. There are two ways of choosing an edge in C, a choice between uw and vw, and a choice between zx and zy. This gives 8 spanning trees, but only 4 is needed.

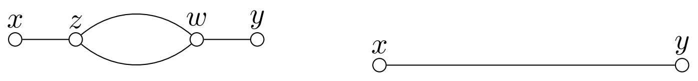
If we have Subcase 1.2, we create $G' = G - \{z, w\} + xy.$, this operation is shown in **??**, and is denoted Operation 2.

Operation 2 reduces the number of vertices in G by 2, so we want to show, that if T' is a spanning tree in G', and we use Operation 2 backwards, then we can extend T'



(a) $T' \subseteq G'$. **(b)** $T' - xy \subseteq G$.

Figure 3.8: The case when $xy \notin T'$. When $xy \in T'$, note that $T' - xy$ will be connected, and that using both xz and yz would yield a cycle.



(a) G **(b)** G'

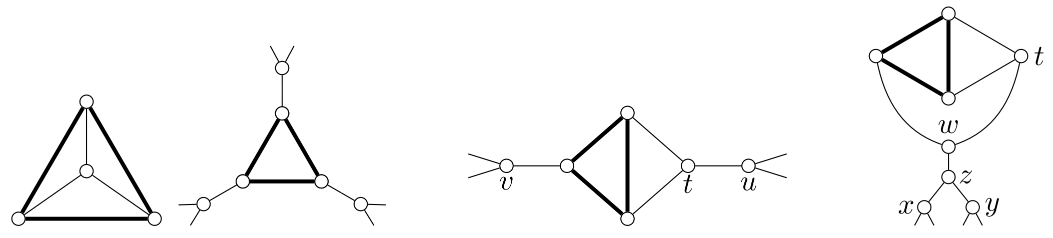
Figure 3.9: Operation 2, used when G has a 2-cycle as in Subcase 1.2.

to 2 spanning trees in G since this implies

$$\tau(G) \geq 2 \cdot \tau(G') \geq 2 \cdot 3 \cdot 2^{\frac{n-2}{2}} = 2 \cdot 3 \cdot \frac{2^{n/2}}{2} = 3 \cdot 2^{n/2}$$

When $xy \in E(T')$, we can extend T' to a spanning tree in G in two ways since we have a choice for one of the edges joining z to w. When $xy \notin E(T')$, we can extend T' to a spanning tree in G in at least 2 ways, since we can choose an edge joining z to w, and either xz or wy.

Case 2 - $v(C) = 3$**:** Then we have one of the four cases shown in **??**.



(a) Subcase 2.1 **(b)** Subcase 2.2 **(c)** Subcase 2.3 **(d)** Subcase 2.4

Figure 3.10: The four different types of 3-cycles

Subcase 2.1 is K_4, which has 16 spanning trees, and this is greater than $3 \cdot 2^2$. When we have Subcase 2.2, we construct $G' = G/V(C)$. This operation is shown in **??** and is denoted Operation 3.

Operation 3 reduces the number of vertices by 2, so we want to show, that if we have a spanning tree T' in G', and use Operation 3 backwards, then we can extend

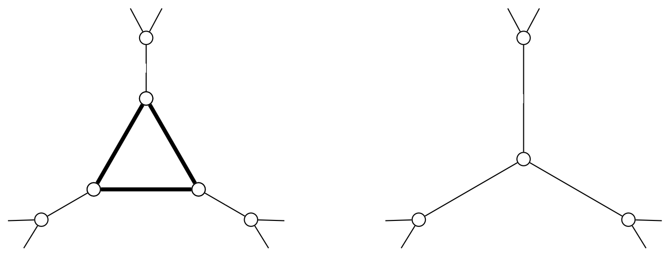


Figure 3.11: Operation 3, used when we have Subcase 2.2.

T' to 2 spanning trees in G. No matter which edges T' uses, we can pick some spanning tree in the triangle, and some additional edges. Since the triangle has 3 spanning trees, this gives at least 3 spanning trees in G for each spanning tree in G'.

When we have Subcase 2.3, we construct $G' = G - \{V(C), t\} + vu$. This operation is shown in **??** and is denoted Operation 4.

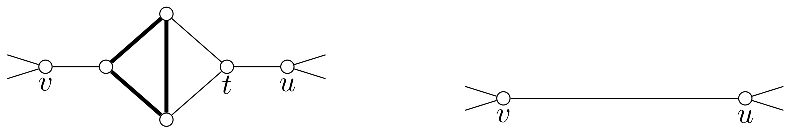


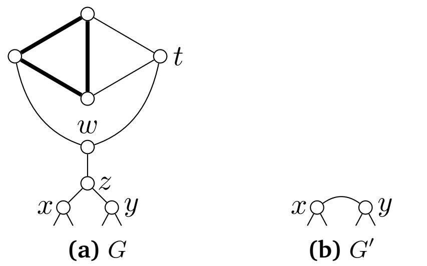
Figure 3.12: Operation 4, used when we have Subcase 2.3.

Operation 4 reduces the number of vertices in G by 4, so we want to show, that if we have a spanning tree T' in G', and use Operation 4 backwards, then we can extend T' to 4 spanning trees in G since $2^{\frac{n-4}{2}} = \frac{2^{n/2}}{4}$. We need to consider the case where vu is in T' and the case where vu is not in T'. Note that the Diamond Graph ($K_4 - e$) has 8 spanning trees. In either case, we can choose some spanning tree in the diamond, and some additional edges. Since the diamond has 8 spanning trees, this gives more than 8 spanning trees for G in all cases.

When we have Subcase 2.4, we construct $G' = G - \{V(C), t, w, z\} + xy$. This operation is shown in **??** and is denoted Operation 5.

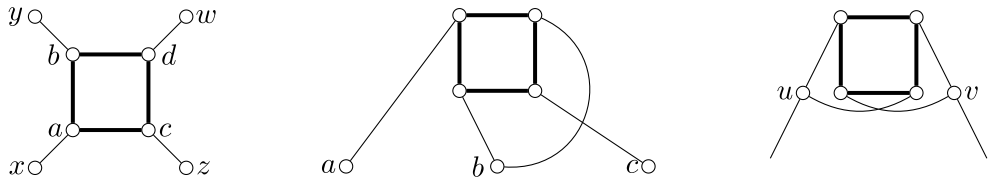
Operation 5 reduces the number of vertices in G by 6, so we want to show, that if we have a spanning tree T' in G', and use Operation 5 backwards, then we can extend T' to 8 spanning trees in G. We need to consider the case where xy is in T' and the case where xy is not in T'. In either of the cases, choose some spanning tree in the diamond, and some additional edges making the tree connected. This gives more than 8 spanning trees in both cases.

Case 3 - $v(C) = 4$**:** Since C is induced, we can have the 3 cases shown in **??**.



(a) G (b) G'

Figure 3.13: Operation 5, used when we have Subcase 2.4.



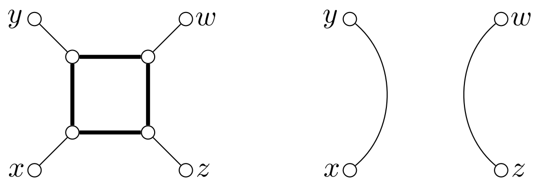
(a) Subcase 3.1: C has four distinct neighbours x, y, z, w.

(b) Subcase 3.2: C has three distinct neighbours a, b, c.

(c) Subcase 3.3: C has two distinct neighbours u, v.

Figure 3.14: The three different types of 4-cycles

When we have Subcase 3.1, we construct $G' = G - V(C) + \{xy, zw\}$. This operation is shown in **??** and is denoted Operation 6.

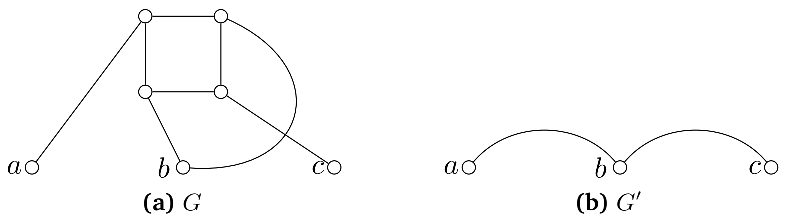


(a) G (b) G'

Figure 3.15: Operation 6, used when we have Subcase 3.1.

Since C is non-separating, G' is connected. Operation 6 reduces the number of vertices in G by 4, so we want to show, that if we have a spanning tree T' in G', and use Operation 6 backwards, then we can extend T' to 4 spanning trees in G. T' can use none of the edges xy, zw, one of them or both of them. In either case, we can choose one of the four spanning trees in C, and some additional edges. This gives at least 4 spanning trees in G for each spanning tree in G'.

When we have Subcase 3.2, we construct $G' = G - V(C) + \{ab, bc\}$. This operation is shown in **??**, and is denoted Operation 7.



(a) G **(b)** G'

Figure 3.16: Operation 7, used when we have Subcase 3.2.

Operation 7 reduces the number of vertices in G by 4, so we want to show, that if we have a spanning tree T' in G' using some combination of the edges ab, bc, and use Operation 7 backwards, then we can extend T' to 4 spanning trees in G. In either of the cases for ab and bc, we can pick one of the four spanning trees in C, and some additional edges. This gives us at least 4 spanning trees in G for each spanning tree in G'.

When we have Subcase 3.3, consider the cycle shown in **??**. This cycle has three distinct neighbours, and thus we have Subcase 3.2.

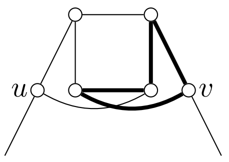


Figure 3.17: When we have Subcase 3.3, we also have Subcase 3.2 and consider that cycle instead.

Case 4 - $v(C) \geq 5$: Now C is a non-separating, induced cycle in G, such that all neighbours of C are distinct. Let k denote the number of vertices in C, that is $k = v(C)$. Now we construct G' by removing $V(C)$, and by lifting the vertices of degree 2. This is shown in **??**. We will let $x_1, x_2, \ldots, x_k$ denote the vertices on C, and let y_i denote the neighbour of x_i not on C. We let $e_1, e_2, \ldots, e_k$ denote the new edges we create by lifting the vertices of degree 2.

G' is a cubic, connected graph on $n - 2k$ vertices. By the induction hypothesis, $\tau(G') \geq 3 \cdot 2^{\frac{n-2k}{2}}$. We will show, that for any spanning tree in T' in G', we can extend T' to at least 2^k spanning trees in G.

We start by considering the case where T' contains all of the edges $e_1, e_2, \ldots, e_k$. In this case we can extend T' to a spanning tree in G, by using some of the edges $x_1y_1, x_2y_2, \ldots, x_ky_k$, and some additional edges from C. We can choose to have 1 to k of the edges $x_1y_1, x_2y_2, \ldots, x_ky_k$ in T, and for each number of edges, we have $\binom{k}{i}$

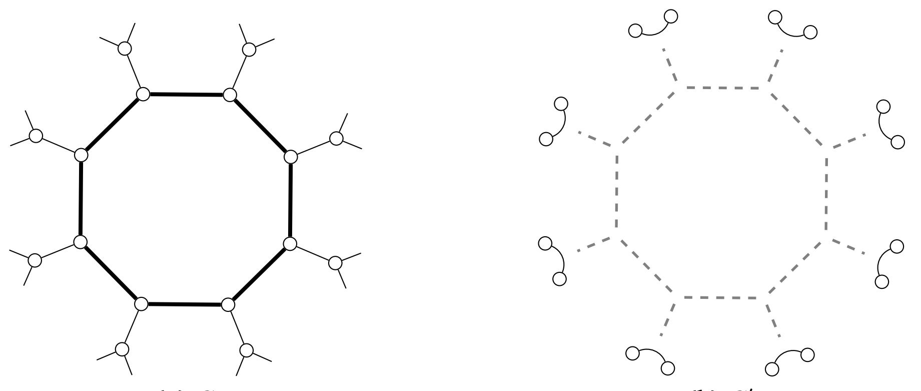

(a) G **(b)** G'

Figure 3.18: Construction of G' by removal of C.

ways to choose them. So in total we have

$$\sum_{i=1}^{k}\binom{k}{i}$$

possible choices of the edges $x_1y_1, x_2y_2, \ldots, x_ky_k$. When i of the k edges are chosen, we need to choose some of the edges on C to be in T, and some not to be in T. The case which gives us the fewest possible choices, is when the i edges we choose from $x_1y_1, x_2y_2, \ldots, x_ky_k$ are mutually adjacent to each other on C. In this case, we can only choose the edges on C in $(k-(i-1))$ ways, since this is the number of ways to choose an edge to exclude from T. Since this holds for any i, we get the inequality

$$\tau(G) \geq \sum_{i=1}^{k}\left[\binom{k}{i}(k-i+1)\right]\cdot\tau(G') \geq \sum_{i=1}^{k}\left[\binom{k}{i}(k-i)\right]\cdot\tau(G')$$

We use combinatorial identities from [?], and reverse the order of summation to obtain

$$\tau(G) \geq \sum_{i=1}^{k}\left[\binom{k}{k-i}i\right]\cdot\tau(G') = \sum_{i=1}^{k}\left[\binom{k}{i}i\right]\cdot\tau(G') = k2^{k-1}\cdot\tau(G')$$

where $k2^{k-1} \geq 2^k$ since $k \geq 5$. Now we have that

$$\tau(G) \geq 2^k\tau(G') = 2^k \cdot 3 \cdot 2^{\frac{n-2k}{2}} = 3 \cdot 2^k$$

Now assume T' does not use all of the edges $x_1y_1, x_2y_2, \ldots, x_ky_k$. Then we have the same choices as before, but after that, we still have to choose some extra edges to make T span all the vertices. These extra choices increases the number of ways to extend T' to a spanning tree in G, thus also satisfying the inequality.

For all even $n \geq 6$, we can create a graph with $\frac{25}{8} \cdot 2^{n/2}$ spanning trees by taking a path of $n-3$ edges, where the first and the last edge are replaced by a triangle with a double edge, and where every second edge is replaced by a double edge. This construction can be seen in **??**.

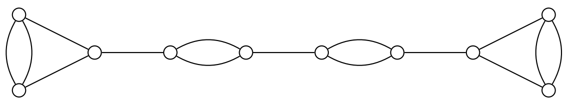


Figure 3.19: A construction of graphs with $\frac{25}{8} \cdot 2^{n/2}$ spanning trees.

The number of spanning trees in this construction has the same asymptotic growth as the lower bound on $3 \cdot 2^{n/2}$. This shows that our bound is asymptotically tight.

3.2 Cubic Simple Graphs

Now we consider simple graphs (no multiple edges), and find a lower bound for the number of spanning trees in simple cubic graphs. This result has been given by [**?**], and we repeat their proof with slight moderation.

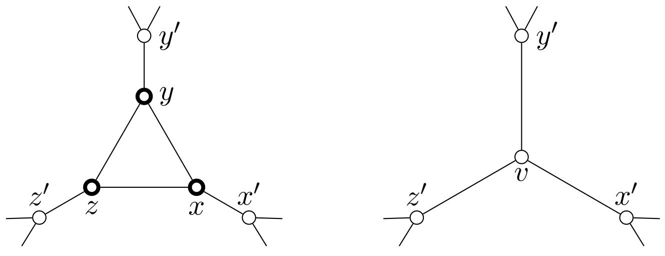
Theorem 3.2 *If G is cubic, connected, simple and $G \neq K_4$, then $\tau(G) \geq \frac{9}{\sqrt{8}} 8^{n/4}$.*

Proof We do induction on the number of vertices n.

Basis of the induction In **??**, it is tested and documented, that all connected, cubic, simple graphs with 6 to 12 vertices have at least $\frac{9}{\sqrt{8}} 8^{n/4}$ spanning trees.

Induction step Let G have n vertices, $n > 12$. Assume the induction hypothesis, that is, all graphs on $k < n$ vertices have at least $\frac{9}{\sqrt{8}} 8^{k/4}$ spanning trees.

Case 1 - G has a triangle: If G has a triangle $\{x, y, z\}$ with neighbours $\{x', y', z'\}$, where x', y', z' are pairwise distinct, construct $G' = G/\{x, y, z\}$, thus reducing the number of vertices by 2. This reduction is shown in **??**.



(a) G **(b)** G'

Figure 3.20: The contraction of a triangle.

By the induction hypothesis, $\tau(G') \geq \frac{9}{\sqrt{8}}8^{(n-2)/4}$, so we want to show that for every spanning tree T' in G', we can extend this to at least $\lceil\sqrt{8}\rceil = 3$ spanning trees in G since $\frac{9}{\sqrt{8}}8^{(n-2)/4} = \frac{9}{\sqrt{8}}\frac{8^{n/4}}{\sqrt{8}}$. T' can use some of the edges vx', vy', vz', and in any of the cases, we can extend T' by picking two of the three edges xy, yz, zx, and some of the edges xx', yy', zz' without creating a cycle, thus giving more than 3 possible ways to extend each spanning tree in G' to a spanning tree in G.

Case 2 - G has a diamond: If G does not have a triangle with distinct neighbours, but still have girth 3, it has a diamond $\{x, y, z, w\}$ with neighbours $\{x', y'\}$.

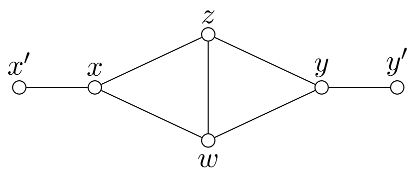


Figure 3.21: Case 2 when G contains a diamond. Note that it might be the case that $x' = y'$.

Now there are two possibilities for x' and y'.

Subcase 2.1 - $x' \neq y'$: We construct $G' = G - \{x, y, z, w\} + \{x'y'\}$. By the induction hypothesis, $\tau(G') \geq \frac{9}{\sqrt{8}}8^{(n-4)/4}$, so we need to extend each spanning tree T' in G' to 8 different spanning trees in G. T' can use $x'y'$ or not. If T' uses the edge $x'y'$, we can choose any spanning tree in the diamond (recall that the diamond graph has 8 spanning trees) for T, and both of the edges xx' and yy'. If T' does not use $x'y'$, we can again choose any spanning tree in the diamond, and one of the edges xx' and yy'. Both cases give us at least 8 ways to extend every spanning tree.

Subcase 2.2 - $x' = y'$: In the case that $x' = y'$, G looks like **??**.

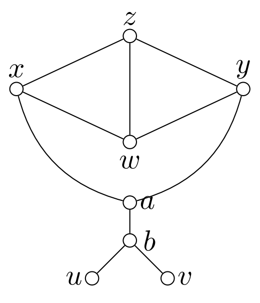


Figure 3.22: Case 2 when $x' = y'$.

Now there are again two cases, either $uv \in E(G)$ or $uv \notin E(G)$.

Subcase 2.2.1 - $uv \notin E(G)$: If this is the case, we construct $G' = G - \{x, y, z, w, a, b\} + \{uv\}$. By the induction hypothesis, $\tau(G') \geq \frac{9}{\sqrt{8}}8^{(n-6)/4}$, so we need to extend each spanning tree T' in G' to at least 23 different spanning trees in G. Let H denote

K_4 with one edge subdivided, which have 24 spanning trees. H is isomorphic to the subgraph with vertices $\{x, y, z, w, a\}$ and edges $\{xz, xw, zw, zy, wy, xa, ya\}$ in G. We can choose any spanning tree in H and some of ub and vb, for T. This gives us at least 24 ways to extend every spanning tree.

Subcase 2.2.2 - $uv \in E(G)$: If this is the case, G must look like **??**

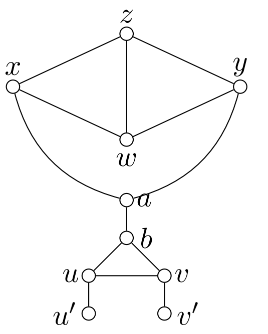


Figure 3.23: G when $uv \in E(G)$.

Now there are two cases, either $u' = v'$ or $u' \neq v'$. If $u' = v'$, we have Subcase 2.1. If $u' \neq v'$, we have Case 1.

Case 3 - G has girth ≥ 4: Let $e = uv$ be a non-cut-edge, i.e. e is contained in some cycle, where $N(u) = \{v, x, y\}$ and $N(v) = \{u, z, w\}$. Since G has girth ≥ 4, xy and zw are not in $E(G)$. By the same argument, $\{x, y, z, w\}$ are pairwise distinct. We now construct $G' = G - \{u, v\} + \{xy, zw\}$, as illustrated in **??**.

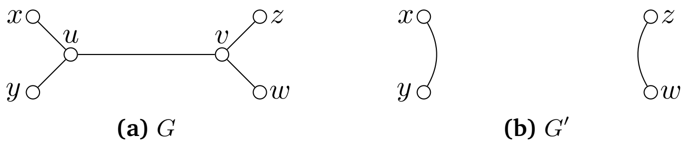


Figure 3.24: The construction of G' when G has girth ≥ 4.

By the induction hypothesis, $\tau(G') \geq \frac{9}{\sqrt{8}} 8^{(n-2)/4}$, so we need to extend each spanning tree T' in G' to at least 3 different spanning trees in G.

There are three possibilities for T', it can use none of the edges xy, zw, it can use one of them, or it can use both.

Subcase 3.1 - T' uses none of the edges xy, zw: Since T' does not use xy or zw, the vertices x, y, z, w are connected when considering T' in G. This is illustrated in **??**. We can extend T' in four ways, by using one of the edges xu, yu, and one of the edges zv, wv. It is possible to extend T' to more than 4 trees, but that is all we need.

Subcase 3.2 - T' uses xy but not zw: When considering T' in G, it consists of 2 connective components. Since xy was in T' when considering G', and was then removed, x and y are in different components when considering T' in G. This is

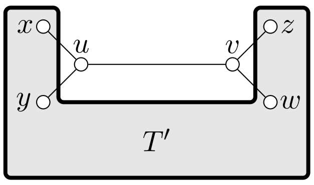


Figure 3.25: When T' does not contain xy or zw, the vertices x, y, z, w are connected in T' after reversing the construction of G'.

illustrated in **??**. Since u is adjacent to both components, we extend T' in three ways by including xu and yu and one of the three edges incident to v.

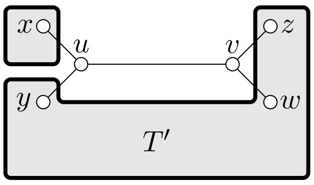


Figure 3.26: The vertex u is adjacent to two connective components of T' after reversing the construction of G'.

Subcase 3.3 - T' uses both xy and zw: Since T' is connected in G', there must exist a path from one of x, y to one of z, w, assume there is a path from y to w. This is illustrated in **??**. Now we extend T' by adding either $\{xu, yu, zv, wv\}$, $\{xu, uv, zv, wv\}$ or $\{xu, yu, uv, zv\}$. Thus giving us three different spanning trees in G by extending T'.

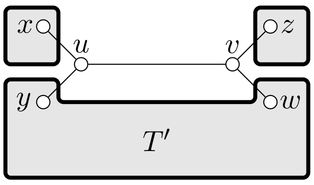


Figure 3.27: The vertices u and v are adjacent to two connective components of T' after reversing the construction of G'.

In each of the subcases, we can extend a spanning tree T' in G', to at least 3 spanning trees in G, concluding the proof.

For all even $n \geq 10$, we can create a graph with $\frac{576}{8^{5/2}} \cdot 8^{n/4}$ spanning trees by taking a path of $\frac{n}{2} - 2$ edges, where the first and the last edge are replaced by K_4 with an edge subdivided, and where every second edge is replaced by a diamond. This construction can be seen in **??**.

The number of spanning trees in this construction has the same asymptotic growth as the lower bound on $\frac{9}{\sqrt{8}} \cdot 8^{n/4}$. This shows that our bound is asymptotically tight.

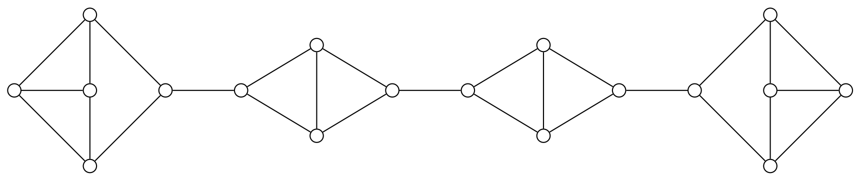


Figure 3.28: A construction of graphs with $\frac{576}{8^{5/2}} \cdot 8^{n/4}$ spanning trees.

Chapter 4

2-Connected Graphs

4.1 Simple 2-Connected Graphs

We use a Lemma from [**?**] stating that

Lemma 4.1 *Let G be a 2-connected graph with at least 4 vertices, and let $e \in E(G)$. Then either $G - e$ is 2-connected or G/e is 2-connected.*

We will use **??** to show a trivial lower bound on the number of spanning trees in 2-connected graphs. When we have a 2-connected graph, we can keep deleting and contracting edges until we reach a 2-connected graph with 3 vertices, namely C_3.

Theorem 4.1 *If G is 2-connected, $\tau(G) \geq e(G)$.*

Proof We do induction on the number of edges m.

Basis of the induction: We will start the induction with the triangle C_3, that is, a cycle with 3 edges. $\tau(C_3) = 3 = e(C_3)$. The only 2-connected graph on ≤ 3 vertices is C_3.

Induction step: Let G be a 2-connected simple graph with m edges. Now we assume the induction hypothesis, that all graphs on k edges, where $k < m$, have at least k spanning trees. Let e be any edge in G. Since G is 2-connected, we know by **??** that either $G - e$ or G/e is 2-connected.

$G - e$ **is 2-connected:** By the induction hypothesis, $\tau(G - e) \geq e(G - e)$. Inserting e in G does not disconnect G, thus every spanning tree in $G - e$ is also a spanning tree in G (avoiding e). Since G is connected, we know by **??**, that there is a spanning tree T in G using e, which therefore can not be a spanning tree in $G - e$. Since we can find at least one more spanning tree in G than in $G - e$, we know that

$$\tau(G) \geq \tau(G - e) + 1 \geq e(G - e) + 1 = e(G)$$

G/e **is 2-connected:** By the induction hypothesis, $\tau(G/e) \geq e(G/e)$. Splitting a vertex to create e does not disconnect G, thus every spanning tree in G/e is also a spanning tree in G (using e). Since $G - e$ is connected, we know by **??**, that there is a spanning tree T in G not using e, which therefore can not be a spanning tree in G/e. Since we can find at least one more spanning tree in G than in G/e, we know that

$$\tau(G) \geq \tau(G/e) + 1 \geq e(G/e) + 1 = e(G)$$

We can construct 2-connected graphs with exactly this number of spanning trees by considering the cycles, so the bound is best possible.

4.2 Simple Cubic 2-Connected Graphs

We show a lower bound on the number of spanning trees in cubic 2-connected graphs. To achieve this, we define a *diamond path*.

A diamond path is a set of diamonds, joined in a path-like structure. More formally, a diamond path is a path $v_1, v_2, \ldots, v_k$ where k is even, and where every second edge $v_1v_2, v_3v_4, \ldots, v_{k-1}v_k$ is replaced by a diamond. In **??** the bold vertices and edges show a diamond path. When we talk about diamond paths, we will only consider maximal (non extendable) diamond paths, that is diamond paths where the neighbouring vertices at the ends (a and b in **??**) are not part of a diamond.

We denote the removal of a diamond path $\mathcal{D}$ from G by $G - \mathcal{D}$, and define it as removing $V(\mathcal{D})$ from G and lifting the vertices getting degree 2. An example can be seen in **??**.

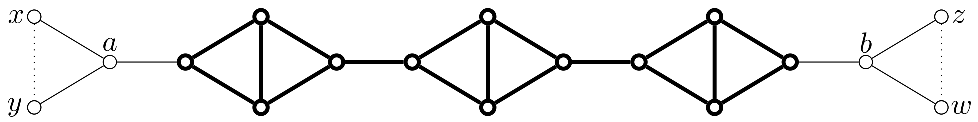
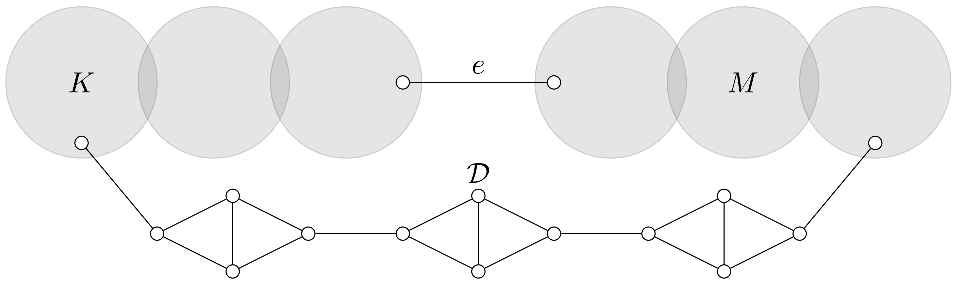


Figure 4.1: A maximal diamond path with 3 diamonds. When removing this diamond path, we remove all inner vertices and a, b and join x to y and z to w.

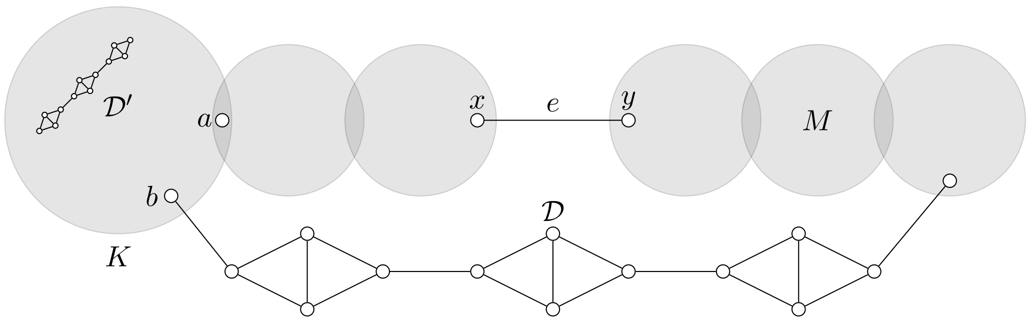
To achieve a lower bound, we will remove diamond paths. **??** shows that we can find removable diamond paths in special graphs.

Lemma 4.2 *If G is simple, cubic, 2-connected, and if G has at least one vertex which is not in a diamond, and if every edge of G has an end vertex which is in a diamond, then G contains a diamond path $\mathcal{D}$, such that $G - \mathcal{D}$ is also 2-connected. Additionally, $\mathcal{D}$ can be chosen such that $v(G - \mathcal{D}) \geq v(\mathcal{D})$.*

Proof We first prove that the diamond path $\mathcal{D}$ can be chosen such that $G - \mathcal{D}$ is 2-connected. Consider therefore any diamond path $\mathcal{D}$ and assume that $G - \mathcal{D}$ has a bridge e (otherwise we have finished). Choose $\mathcal{D}$ such that the largest block M of $G - \mathcal{D}$ is as large as possible. Let K denote the block on the opposite side of e, joined to D by an edge.



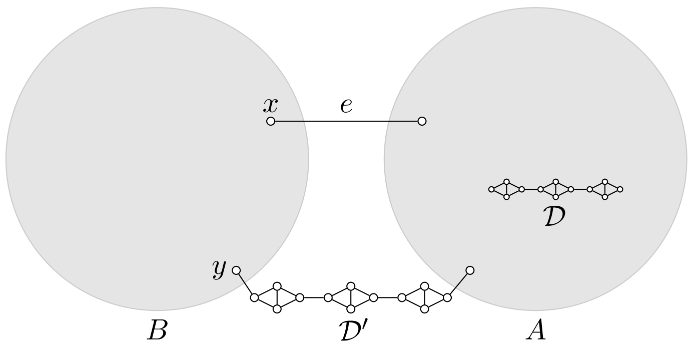
Take a diamond path $\mathcal{D}'$ in K. Such a diamond path exists because G has a vertex not in a diamond (in fact G has at least 3 diamond paths). Since K is a block, there is no bridge in K, and thus $K - \mathcal{D}'$ is still connected. If we can find a cycle in $G - \mathcal{D}'$, that goes through inner vertices of M and K, then we come to a contradiction since we assumed that the largest block M of $G - \mathcal{D}$ was as large as possible. Denote the endpoints of e by x and y and let a and b denote the cut-vertices of $G - e$ insinde K.



Since G is 2-connected, it does not contain a bridge, and thus $G - e$ is connected. Now look at $G - \mathcal{D}' - e$. Take the path P_1 from x to a. Since $K - \mathcal{D}'$ is connected, there is a path P_2 from a to b inside of $K - \mathcal{D}'$. Now take the path P_3 from b to y (this path goes through M and $\mathcal{D}$). Together $P_1 + P_2 + P_3$ forms a path from x to y in $G - \mathcal{D}' - e$, and $P_1 + P_2 + P_3 + e$ forms a cycle in $G - \mathcal{D}'$ using vertices from M, and additional other. This cycle contradicts that the largest block of $G - \mathcal{D}$ was as large as possible, since we can choose $\mathcal{D}'$ instead to achieve larger block.

Now we prove that it is possible to choose $\mathcal{D}$ such that $v(G - \mathcal{D}) \geq v(\mathcal{D})$. Let $\mathcal{D}$ be a diamond path in G, such that $G - \mathcal{D}$ is 2-connected, and such that $v(G - \mathcal{D})$ is as large as possible. If $v(G - \mathcal{D}) \geq v(\mathcal{D})$, then we are finished. Otherwise, consider another diamond path $\mathcal{D}'$ in G, such that the block in $G - \mathcal{D}'$ containing $\mathcal{D}$ is as large as possible. Since $v(G - \mathcal{D}) < v(\mathcal{D})$, we know that $v(\mathcal{D}') < v(\mathcal{D})$. If $G - \mathcal{D}'$ is 2-connected, then we have a contradiction since we chose $\mathcal{D}$ in such a way, that $G - \mathcal{D}$ was as large as possible. If not, then $G - \mathcal{D}'$ must contain a bridge e, and since $G - \mathcal{D}$ is 2-connected, e can not be in $\mathcal{D}$. Let A denote the component in $G - \mathcal{D}' - e$

containing $\mathcal{D}$, and let B denote the block on the opposite side of e. Let x denote the endpoint of e in B. Let y denote the endpoint of $\mathcal{D}'$ in B.



Since G is cubic, two different diamond paths $\mathcal{D}_1$ and $\mathcal{D}_2$ (different from $\mathcal{D}'$) must leave y inside B. Since G is 2-connected, there must exist a cycle through every vertex of $\mathcal{D}_1$ and $\mathcal{D}_2$. Now $G - \mathcal{D}_1$ contradicts that the block in $G - \mathcal{D}'$ containing $\mathcal{D}$ was as large as possible, since removing $\mathcal{D}_1$ instead of $\mathcal{D}'$ would result in a larger block containing both A and $\mathcal{D}_2$. ■

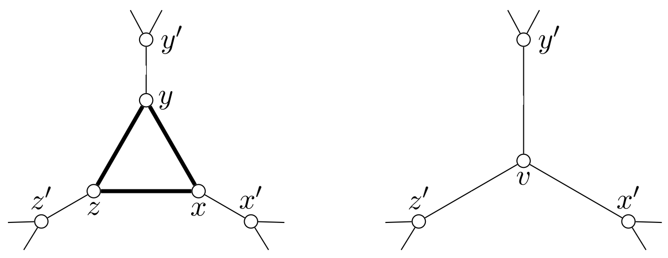
Theorem 4.2 *If G is 2-connected, cubic and simple, $\tau(G) \geq \frac{1}{11}n \cdot 8^{n/4}$.*

Proof We do induction on the number of vertices n.

Basis of the induction Since $\frac{9}{\sqrt{8}}8^{n/4} > \frac{1}{11}n \cdot 8^{n/4}$ for $n \leq 35$, we know from **??** that $\tau(G) \geq \frac{1}{11}n \cdot 8^{n/4}$ is true for all cases where $n \leq 35$.

Induction step Let G have n vertices, $n > 35$. Assume the induction hypothesis, that is, all graphs on $k < n$ vertices have at least $\frac{1}{11}k \cdot 8^{k/4}$ spanning trees.

Case 1 - G has a triangle: If G has a triangle $\{x, y, z\}$ with neighbours $\{x', y', z'\}$, where x', y', z' are pairwise distinct, construct $G' = G/\{x, y, z\}$, reducing the number of vertices by 2.



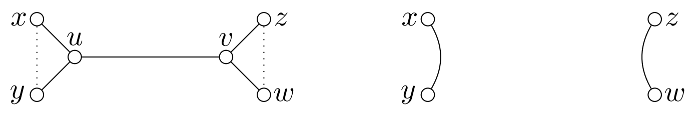
(a) G **(b)** G'

For every spanning tree T' in G', we can extend it to three spanning trees in G by the argument given in Case 1 of **??**. Since we can extend each spanning tree in G' in three ways, we know that $\tau(G) \geq 3\tau(G')$, and now we have

$$\tau(G) \geq 3 \cdot \tau(G') \geq 3 \cdot \frac{1}{11}(n-2)8^{(n-2)/4} \geq \frac{1}{11}n8^{n/4}$$

where the last inequality is true for $n \geq 35$. Since we assumed, $n > 35$, this is fine.

Case 2 - G has an edge, not adjacent to a triangle: If G has an edge uv not incident to a triangle, where the neighbours of u are x, y, v and where the neighbours of v are z, w, u, construct $G' = G - \{u, v\} + \{xy, zw\}$. Since uv is not incident to a triangle, x, y, z, w are pairwise distinct.



(a) G **(b)** G'

We remove 2 vertices when constructing G', so we need (by the same argument as in Case 1) to extend each spanning tree in G' to three spanning trees in G. Since G is 2-connected, this is possible by the argument given in Case 3 of the proof of **??**.

Case 3 - All edges in G are incident to a vertex in a diamond: If G does not contain any triangle not part of a diamond, and does not contain any edges not incident to a vertex in a triangle, G must consist of diamonds.

Subcase 3.1 - G is a closed diamond cycle: If every vertex of G is in a diamond, then G is a closed cycle of diamonds as illustrated in **??**.

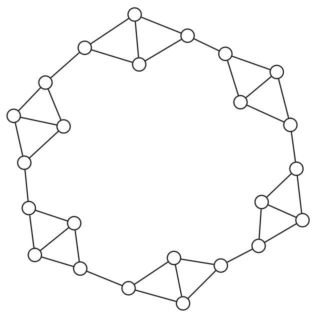


Figure 4.2: A closed diamond cycle.

In this case, the number of spanning trees can be counted in the following way. Let q be the number of diamonds in G, that is $v(G) = 4q$. To obtain a spanning tree in G, we need to decide which edges to exclude to avoid creating a cycle. We can either exclude one of the edges joining two diamonds or we can exclude some edges inside a diamond. An edge between to diamonds can be chosen in q ways, and for each choice, we can choose the rest of the spanning tree in 8^q ways since we have 8 possibilities for each of the q diamonds. Likewise, we can pick a diamond in q ways, and choose the rest of the spanning tree in 8^q ways, since we can *break* the diamond in 8 ways (use **??**, and count the number of spanning trees in K_4 with an edge contracted), and have 8^{q-1} choices for the rest of the diamonds. In total, this gives $\tau(G) = 2(q \cdot 8^q) = \frac{n}{2} \cdot 8^{n/4}$, which is greater than the desired bound.

Subcase 3.2 - G is multiple joined diamond cycles: If G contains vertices which are not in a diamond, G consists of diamond cycles, joined together like illustrated in **??**.

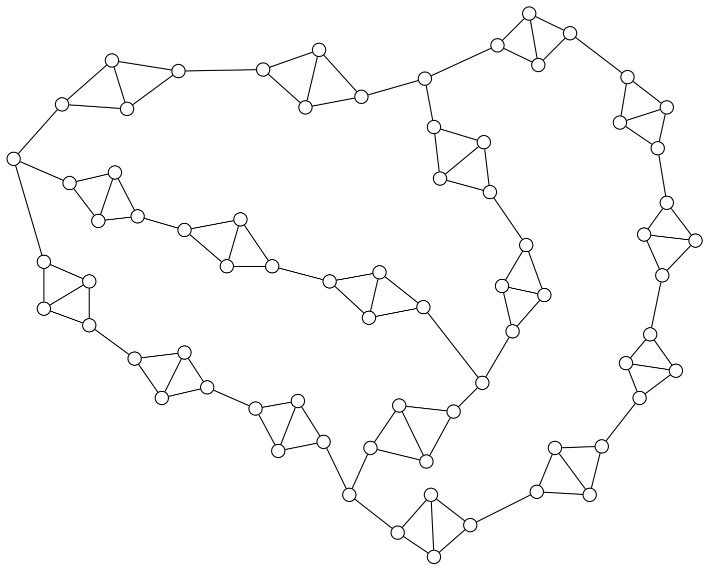


Figure 4.3: Multiple diamond cycles joined together.

In this case, G has a diamond path $\mathcal{D}$. Now construct G' by removing $\mathcal{D}$. By **??**, we can choose $\mathcal{D}$ in such a way that $G - \mathcal{D}$ is 2-connected and such that $v(G - \mathcal{D}) > v(\mathcal{D})$. We denote the number of diamonds on $\mathcal{D}$ by q, and thus the number of vertices in G' is $v(G) - 4q - 2$. For every spanning tree T' in G', we extend it to a number of spanning trees in G. Let a be the left neighbour of $\mathcal{D}$ and b be the right neighbour of $\mathcal{D}$. Let a have neighbours x, y outside of $\mathcal{D}$ and let b have neighbours z, w outside of $\mathcal{D}$. The way we extend a spanning tree in this case is like Case 3 in **??**.

Subcase 3.2.1 - T' uses both xy and zw: Since $G - \mathcal{D}$ is 2-connected, there is a path from one of x, y to one of z, w in $G - \mathcal{D}$, assume x to z. Now a spanning tree in G' can be extended to a spanning tree in G in the following ways. Include ay and bw but avoid one of xa and zb, and choose a spanning tree for $\mathcal{D}$, this results in $2 \cdot 8^q$ spanning trees. Otherwise, choose a spanning tree for $\mathcal{D}$ excluding a single edge between two diamonds. This gives $(q + 1) \cdot 8^q$ spanning trees. At last, choose a spanning tree for $\mathcal{D}$ which is disconnected inside a diamond on $\mathcal{D}$. This gives additionally $q \cdot 8^q$ spanning trees. In total, this gives

$$2 \cdot 8^q + q \cdot 8^q + (q + 1) \cdot 8^q$$

spanning trees in G for each spanning tree in G'. Now we need to show that

$$\begin{aligned}(2\cdot 8^q + q\cdot 8^q + (q+1)\cdot 8^q)\cdot\tau(G') &\geq \tau(G)\\ (3+2q)\,8^q\cdot\frac{1}{11}(n-4q-2)8^{\frac{n-4q-2}{4}} &\geq \frac{1}{11}n\cdot 8^{\frac{n}{4}}\\ (3+2q)\,8^q\cdot(n-4q-2)8^{\frac{n-4q-2}{4}} &\geq n\cdot 8^{\frac{n}{4}}\\ (3+2q)\,8^q\cdot(n-4q-2)\frac{8^{n/4}}{8^q\sqrt{8}} &\geq n\cdot 8^{\frac{n}{4}}\\ (3+2q)\cdot(n-4q-2)\frac{1}{\sqrt{8}} &\geq n\end{aligned}$$

By **??**, we can pick $\mathcal{D}$ in such a way, that $n \geq 8q$, and such that $G - V(\mathcal{D})$ is still 2-connected. When this is the case, the inequality holds.

Subcase 3.2.2 - T' uses neither xy nor zw: In this case, choose one of the four edges xa, ya, zb, wb, and for each of these, we can choose 8^q spanning trees for the q diamonds on $\mathcal{D}$, this gives us $4\cdot 8^q$ spanning trees. We can also choose one of xa, ya and one of zb, wb, and then choose a spanning tree in the diamond path excluding one edge, this can be done in $(q+1)8^q$ ways, giving us $4(q+1)8^q$ spanning trees. At last, we can again take one of xa, ya and one of zb, wb and choose a spanning tree in $\mathcal{D}$ where one of the diamonds is disconnected in the spanning tree. This can be done in $4q8^q$ ways. This gives us a total of

$$4\cdot 8^q + 4(q+1)\cdot 8^q + 4\cdot q\cdot 8^q$$

spanning trees in G for each spanning tree in G'. Since this is more than in Subcase 3.2.1, we have enough spanning trees in G.

Subcase 3.2.3 - T' uses xy but not zw: In this case, we can extend a spanning tree in the following ways. We can choose a spanning tree for $\mathcal{D}$, excluding both zb and wb, this can be done in 8^q ways. We can use zb or wb and exclude ax, and then choose a spanning tree for $\mathcal{D}$, giving additionally $2\cdot 8^q$ spanning trees. We can use zb or wb, and then choose a spanning tree for $\mathcal{D}$ excluding a single edge between two diamonds, giving $2(q+1)8^q$ spanning trees. At last, we can use zb or wb, and then choose a spanning tree for $\mathcal{D}$ which is disconnected inside a diamond on $\mathcal{D}$ giving additionally $2q8^q$ trees. In total, this gives

$$3\cdot 8^q + 2\cdot q\cdot 8^q + 2\cdot(q+1)\cdot 8^q$$

ways to extend each spanning tree in G' to a spanning tree in G. Since this is more than in Subcase 3.2.1, we have enough spanning trees in G.

We have shown that, if G is 2-connected, cubic and simple, $\tau(G) \geq \frac{n}{11}8^{n/4}$, but it seems like the correct bound is $\tau(G) \geq \frac{n}{2}8^{n/4}$. It is possible to construct graphs with this exact number of spanning trees, by taking a cycle, and replacing every second edge with a diamond, illustrated in **??**. The number of spanning trees in such a *closed diamond cycle* is $\frac{n}{2}8^{n/4}$, which shows that our bound is asymptotically tight. We believe that the closed diamond cycles are the 2-connected cubic graphs with the fewest possible spanning trees.

Chapter 5

3-Connected Graphs

5.1 Another Counter Example to a Conjecture by Tutte

W. T. Tutte conjectured [1] the following

Conjecture 5.1 *Among all 3-connected planar graphs with $2m$ edges, the graph with the smallest number of spanning trees is the wheel W_{m+1} on $m+1$ vertices.*

But in 1976, [**?**] found an infinite series of graphs for which the number of spanning trees is smaller than the wheels Tutte conjectured. The smallest graph in their construction which was a counter example to Tutte's Conjecture had 12 vertices and 30 edges.

5.1.1 The Counter Example

I present a graph (shown in **??**) on 20 vertices and 30 edges, which has $1.755.600$ spanning trees, this is less than W_{16} which also has 30 edges, but has $1.860.496$ spanning trees, and it is less than the smallest counter example presented in [**?**] which has $1.815.792$ spanning trees.

5.1.2 A Construction

We describe the construction from [**?**]. Let P_k be a path on k vertices. Define A_n (where $n \geq 3$) to be $P_2 \circ P_{n-2}$, where $G \circ H$ is the graph consisting of G and H, where every vertex in G is joined to every vertex in H. Thus, A_n has n vertices and $3n-6$ edges and is illustrated in **??**.

To show that A_n is a counter example to Conjecture **??**, we need to determine $\tau(A_n)$. We use **??** on A_n by contracting and deleting the edge in P_2 which we will denote e.

[1]This conjecture is not stated in any article published by Tutte. In [**?**], it is claimed that Tutte has confirmed his conjecture in private communication with the authors.

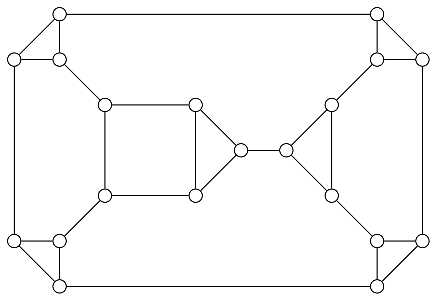


Figure 5.1: A 3-connected graph with 30 edges and $1.755.600$ spanning trees.

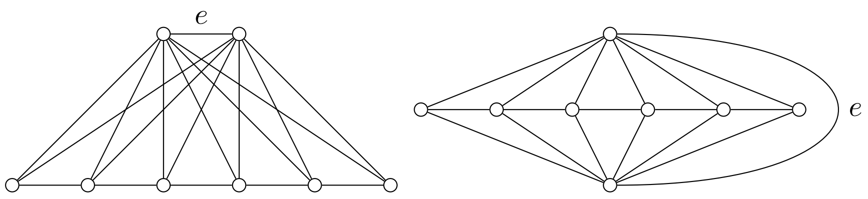
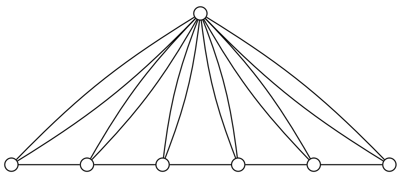
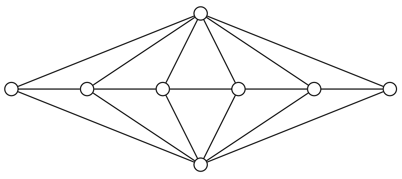


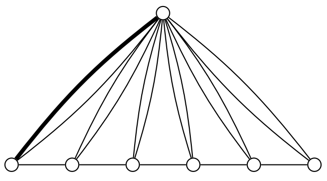
Figure 5.2: Different embeddings of A_8 constructed by P_2 and P_6

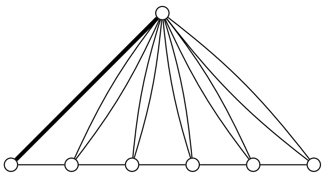
Deleting e from A_n, gives us a graph we will denote S_n, while contracting e in A_n gives us a graph we will denote H_{n-1}. Thus we split the problem of determining $\tau(A_n)$ into two cases, namely finding $\tau(S_n)$ and $\tau(H_n)$.

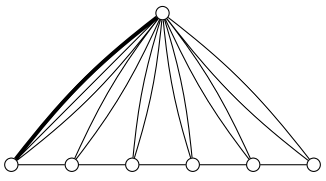

(a) $H_{n-1} = A_n/e$


(b) $S_n = A_n - e$

To find $\tau(H_n)$, we will find a set of recurrence relations and solve them. We consider the following three graphs H_n, B_n and C_n where n denote the number of vertices.


(a) H_7


(b) B_7


(c) C_7

By using **??** on H_n, we can - by contracting and deleting the bold edge - get the following expression $\tau(H_n) = \tau(B_n) + \tau(C_{n-1})$. In the same way, we can get expressions for $\tau(B_n)$ and $\tau(C_n)$. In total, we get the following system of coupled linear recurrence relations.

$$\begin{aligned}\tau(H_n) &= \tau(B_n) + \tau(C_{n-1})\\ \tau(B_n) &= \tau(H_{n-1}) + \tau(C_{n-1})\\ \tau(C_n) &= \tau(H_n) + \tau(C_{n-1})\end{aligned}$$

Our goal is to find a single reccurence relation for H_n, which can be found by solving the system of recurrence relations using regular substitution. We solve for $\tau(C_n)$ in the first relation, and plug this into the other equations

$$\begin{aligned}\tau(B_n) &= \tau(H_{n-1}) + \tau(H_n) - \tau(B_n)\\ \tau(H_{n+1}) - \tau(B_{n+1}) &= 2\tau(H_n) - \tau(B_n)\end{aligned}$$

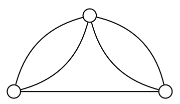
Then we solve for $\tau(B_n)$ in the first of these relations, and plug it into the other to obtain a relation involving only H_n.

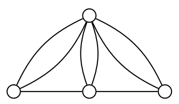
$$\begin{aligned}\tau(B_n) &= \frac{\tau(H_n) + \tau(H_{n-1})}{2}\\ \tau(H_{n+1}) - 2\tau(H_n) &= \frac{\tau(H_{n+1}) + \tau(H_n)}{2} - \frac{\tau(H_n) + \tau(H_{n-1})}{2}\end{aligned}$$

This gives $\tau(H_n) = 4\tau(H_{n-1}) - \tau(H_{n-2})$. We now have a single linear recurrence relation for $\tau(H_n)$ which we can solve using the theory described in **??**. We guess that $\tau(H_n)$ is on the form α^n so we get the equation

$$\alpha^n = 4\alpha^{n-1} - \alpha^{n-2}$$

Let α_1 and α_2 be the two roots of this equation, then the general solution is on the form $A\alpha_1^n + B\alpha_2^n$. We divide by α^{n-2}, so we get $\alpha^2 = 4\alpha - 1$, which we can solve to get $\alpha = 2 \pm \sqrt{3}$. Now we know that $\tau(H_n) = A(2+\sqrt{3})^n + B(2-\sqrt{3})^n$, where we can solve for A and B by computing $\tau(H_3)$ and $\tau(H_4)$.


(a) $\tau(H_3) = 8$


(b) $\tau(H_4) = 30$

Since $\tau(H_3) = 8$ and $\tau(H_4) = 30$, we get $A = \frac{2}{3}\sqrt{3} - 1$ and $B = -\frac{2}{3}\sqrt{3} - 1$. Thus, we get the following expression for $\tau(H_n)$

$$\tau(H_n) = \left(\frac{2}{3}\sqrt{3} - 1\right)(2+\sqrt{3})^n + \left(-\frac{2}{3}\sqrt{3} - 1\right)(2-\sqrt{3})^n$$

We want to compute $\tau(A_n)$, this can be done using electrical network theory [**?**], and the same approach is used in [**?**]. If we look at A_n as a two-terminal electrical network with the edges as branches of unit resistance, the vertices as nodes, and the vertices in P_2 (during the construction of A_n) denoted s and t as terminal nodes, then the quotient R_{st} called the driving point resistance between s and t, can be computed as

$$R_{st} = \frac{\tau(A_n/st)}{\tau(A_n)} = \frac{\tau(H_{n-1})}{\tau(A_n)}$$

So if we can compute this quotient R_{st}, then we can also find $\tau(A_n)$ by the formula

$$\tau(A_n) = \frac{\tau(H_{n-1})}{R_{st}}$$

This network can be visualized as shown in **??**.

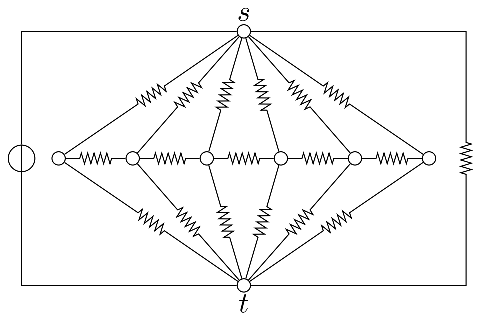


Figure 5.3: A visualization of the electrical network used to compute the quotient R_{st}. The resistors are of unit resistance and the edge which is not a resistor is a voltage generator.

When we have this network, we can compute R_{st} by the methods described in [**?**, **?**]. We start by inserting a voltage generator from s to t. Now the node-voltage in s is 1, and the node-voltage in t is 0. Since the edge st is joined directly to s and t, the voltage drop over this edge must be $1 - 0 = 1$. If we consider one of the vertices in the middle of the network (on P_{n-2}), then we will compute the node-voltage in this vertex. By [**?**], the node-voltage in a vertex v, can be found as the probability that a random walk starting in v, reaches s before t. Now we can use the symmetry of the network, to see that the node voltage must be $1/2$ in all vertices except s and t.

The voltage drop over an edge, is the difference between the node-voltage in the endpoints of the edge. This gives us the edge-voltages shown in **??**.
Since all resistors in the network are of unit resistance, Ohm's law gives us that $I = V$. So every edge in the network, has the same current going through it, as the voltage drop over it. Now we will calculate the current going from s to t. One unit of current is flowing over st, and $1/2$ unit over each of the other paths of length 2 going out of s. There are $n - 2$ edges from s not going directly to t, so the total current is

$$I = \frac{1}{2}(n-2) + 1 = \frac{n}{2}$$

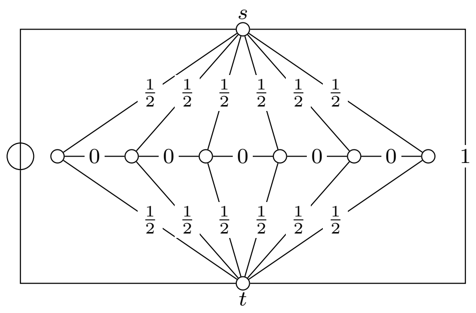


Figure 5.4: The label on the edges show the voltage drop over the corresponding resistor. Since $V = R \cdot I$, and since all resistors are of unit resistance, the current through an edge is equal to the voltage drop over the edge.

So we can find that $R = \frac{1}{n/2} = \frac{2}{n}$ and thus

$$\tau(A_n) = \frac{\tau(H_{n-1})}{2/n} = n \cdot \frac{\tau(H_{n-1})}{2} = \frac{n}{2}\tau(H_{n-1})$$

where we can plug in our formula for $\tau(H_{n-1})$

$$\tau(A_n) = n\left(\frac{1}{6}\sqrt{3}(2+\sqrt{3})^{n-1} - \frac{1}{6}\sqrt{3}(2-\sqrt{3})^{n-1}\right)$$

To show that the graphs constructed in [**?**] has fewer spanning trees than the wheels, we need a closed form for $\tau(W_n)$. This has been approached by first [**?**], and later by [**?**, **?**], but we will do the calculations ourselves. We consider 5 different graphs, denoted w_n, a_n, b_n, d_n, e_n where n denote the number of vertices, shown in **??**

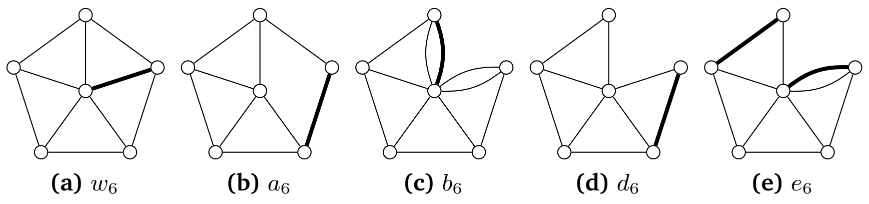


Figure 5.5: The five families of graphs we use to find an explicit formula for the number of spanning trees in the wheel. We use the Contraction-Deletion Theorem on the emphasized edges, to find a system of recurrence relations.

Now we find the following set of recurrence relations by using the Contraction-Deletion theorem on the bold edges, marked in the graphs.

$$\begin{aligned}
\tau(w_n) &= \tau(a_n) + \tau(b_{n-1}) \\
\tau(a_n) &= \tau(d_{n-1}) + \tau(w_{n-1}) \\
\tau(b_n) &= \tau(e_n) + \tau(b_{n-1}) \\
\tau(d_n) &= \tau(d_{n-1}) + \tau(e_{n-1}) \\
\tau(e_n) &= \tau(d_n) + \tau(e_{n-1}) = \tau(e_{n-1}) + \tau(b_{n-1})
\end{aligned}$$

Our idea is, to show that $\tau(a_n)$ and $\tau(d_n)$ follow the same recurrence relation, and then use that their sum $\tau(w_n)$ must also follow that recurrence relation. We start by considering the two last equations for $\tau(d_n)$ and $\tau(e_n)$.

$$\begin{aligned}\tau(d_n) &= \tau(d_{n-1}) + \tau(e_{n-1})\\ \tau(e_n) &= \tau(d_n) + \tau(e_{n-1})\end{aligned}$$

Now we isolate $\tau(e_{n-1})$ in the first equation, and plug it into the second equation to obtain

$$(\tau(d_{n+1}) - \tau(d_n)) = \tau(d_n) + (\tau(d_n) - \tau(d_{n-1}))$$

which reduces to

$$\tau(d_{n+1}) = 3\tau(d_n) - \tau(d_{n-1})$$

When we have this, we change the index and get the following equation

$$\tau(d_n) = 3\tau(d_{n-1}) - \tau(d_{n-2}) \Rightarrow 0 = \tau(d_n) - 3\tau(d_{n-1}) + \tau(d_{n-2})$$

we change the index again, to get

$$0 = \tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})$$

and then we subtract these two equations with different index from each other to get

$$\begin{aligned}0 - 0 &= (\tau(d_n) - 3\tau(d_{n-1}) + \tau(d_{n-2})) - (\tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3}))\\ 0 &= \tau(d_n) - 4\tau(d_{n-1}) + 4\tau(d_{n-2}) - \tau(d_{n-3})\end{aligned}$$

which is our final recurrence relation for $\tau(d_n)$. From the last relation, we can see that $\tau(b_{n-1}) = \tau(d_n)$. Which we plug into the first relation $\tau(w_n) = \tau(a_n) + \tau(b_{n-1})$

$$\tau(w_{n-1}) = \tau(a_{n-1}) + \tau(d_{n-1})$$

Now, we plug this into the second relation $\tau(a_n) = \tau(d_{n-1}) + \tau(w_{n-1})$ to obtain

$$\tau(a_n) = \tau(d_{n-1}) + \tau(w_{n-1}) = \tau(d_{n-1}) + (\tau(a_{n-1}) + \tau(d_{n-1})) = \tau(a_{n-1}) + 2\tau(d_{n-1})$$

thus giving

$$\tau(a_n) - \tau(a_{n-1}) = 2\tau(d_{n-1})$$

We showed that $0 = \tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3})$, and from this we obtain

$$\begin{aligned}2 \cdot 0 &= 2(\tau(d_{n-1}) - 3\tau(d_{n-2}) + \tau(d_{n-3}))\\ 0 &= 2\tau(d_{n-1}) - 2 \cdot 3\tau(d_{n-2}) + 2\tau(d_{n-3})\\ 0 &= (\tau(a_n) - \tau(a_{n-1})) - 3(\tau(a_{n-1}) - \tau(a_{n-2})) + (\tau(a_{n-2}) - \tau(a_{n-3}))\\ 0 &= \tau(a_n) - 4\tau(a_{n-1}) + 4\tau(a_{n-2}) - \tau(a_{n-3})\end{aligned}$$

which is the final recurrence relation for $\tau(a_n)$. Now we have that both $\tau(a_n)$ and $\tau(d_n)$ follow the recurrence relation $0 = x_n - 4x_{n-1} + 4x_{n-2} - x_{n-3}$, and therefore, their sum $\tau(w_n)$ must follow the same recurrence relation. Now we guess for a solution $\tau(w_n) = \alpha^n$.

$$0 = \alpha^n - 4\alpha^{n-1} + 4\alpha^{n-2} - \alpha^{n-3}$$

and divide by α^{n-3} to get

$$0 = \alpha^3 - 4\alpha^2 + 4\alpha - 1$$

Now, if $\alpha_1, \alpha_2, \alpha_3$ are the three roots of this equation, then

$$A\alpha_1^n + B\alpha_2^n + C\alpha_3^n$$

is the general solution to the recurrence relation. We see that $\alpha_1 = 1$ is a root in the equation, and isolate $(\alpha - 1)$ to get

$$0 = (\alpha - 1)(\alpha^2 - 3\alpha + 1)$$

where the last factor has roots

$$\alpha = \frac{3 \pm \sqrt{9-4}}{2} = \frac{3 \pm \sqrt{5}}{2}$$

So we have the general solution

$$A\left(\frac{3+\sqrt{5}}{2}\right)^n + B\left(\frac{3-\sqrt{5}}{2}\right)^n + C$$

We compute the initial conditions and get

$$\tau(W_3) = 16 \quad \tau(W_4) = 45 \quad \tau(W_5) = 121$$

Now we solve with these initial conditions, to obtain the values $A = 1, B = 1, C = -2$, giving the final equation for the number of spanning trees in the wheel on n vertices

$$\tau(W_n) = \left(\frac{3+\sqrt{5}}{2}\right)^n + \left(\frac{3-\sqrt{5}}{2}\right)^n - 2$$

If we consider A_{12} which has 30 edges, and the wheel W_{16} on 30 edges, we get the following number of spanning trees $\tau(A_{12}) = 1815792$, and $\tau(W_{16}) = 1860496$. This shows, that A_n is a counter example to **??**.

Not only is the graphs constructed in [**?**] a counter example to the conjecture by Tutte, but they actually have a smaller order of growth than the wheel graphs. We want to compare

$$\tau(W_n) = \left(\frac{3+\sqrt{5}}{2}\right)^n + \left(\frac{3-\sqrt{5}}{2}\right)^n - 2$$

to

$$\tau(A_n) = n\left(\frac{1}{6}\sqrt{3}(2+\sqrt{3})^{n-1} - \frac{1}{6}\sqrt{3}(2-\sqrt{3})^{n-1}\right)$$

We start by determining their individual asymptotic growth

$$\tau(W_n) = O\left(\left(\frac{3+\sqrt{5}}{2}\right)^n\right)$$

$$\tau(A_n) = O\left(n(2+\sqrt{3})^{n-1}\right)$$

Since Tutte's Conjecture is based on the number of edges in the graph, we will rewrite $\tau(A_n)$ and $\tau(W_n)$ to depend on the number of edges m. The wheel graph on m edges, has $\frac{m+2}{2}$ vertices, and the graphs constructed by [?] with m edges has $\frac{m+6}{3}$ vertices. So we will compare

$$\tau(W_m) = O\left(\left(\frac{3+\sqrt{5}}{2}\right)^{\frac{m+2}{2}}\right) = O\left(\left(\frac{3+\sqrt{5}}{2}\right)^{\frac{m}{2}}\right)$$

with

$$\tau(A_n) = O\left(\frac{m+6}{3}(2+\sqrt{3})^{\frac{m+6}{3}}\right) = O\left(m(2+\sqrt{3})^{\frac{m}{3}}\right)$$

We consider the limit

$$\lim_{m\to\infty}\frac{\tau(W_m)}{\tau(A_m)} = \lim_{m\to\infty}\frac{\left(\frac{3+\sqrt{5}}{2}\right)^{\frac{m}{2}}}{m(2+\sqrt{3})^{\frac{m}{3}}} = \lim_{m\to\infty}\frac{1}{m}\left(\frac{\left(\frac{3+\sqrt{5}}{2}\right)^{\frac{1}{2}}}{(2+\sqrt{3})^{\frac{1}{3}}}\right)^m = \infty$$

since

$$\frac{\left(\frac{3+\sqrt{5}}{2}\right)^{\frac{1}{2}}}{(2+\sqrt{3})^{\frac{1}{3}}} > 1.04 > 1$$

this implies that A_m grows asymptotically slower than W_m.

5.2 Generating All Minimal 3-Connected Graphs

To achieve a lower bound on the number of spanning trees in 3-connected graphs, one can generate all 3-connected graphs on n vertices for small n. Unfortunately, the number of 3-connected graphs on n vertices, grows very fast, so even for relatively small n, generating them all is impossible.

One key observation to make, is that the 3-connected graph on n vertices with the fewest possible spanning trees is *minimally 3-connected*. Here minimally 3-connected means that the graph is 3-connected, but that the removal of any edge would reduce the connectivity. If it was not the case that the graph with fewest spanning trees was minimal, there would - by definition - be an edge which could be removed without losing 3-connectivity. Removing an edge in a 3-connected graph will reduce the number of spanning trees (can easily be seen by the contraction-deletion theorem). The number of 3-connected graphs is given by the sequence A006290 in OEIS[2] and the number of minimally 3-connected graphs, is based on our computations.

Vertices	3-connected graphs	Minimally 3-connected graphs
4	1	1
5	3	1
6	17	3
7	136	5
8	2.388	18
9	80.890	57
10	5.114.079	285
11	573.273.505	1.513

So instead of generating all 3-connected graphs, one can create all minimal 3-connected graphs on small n, compute the number of spanning trees in each, and in that way, find the 3-connected graphs with fewest spanning trees. The naive way to generate the minimally 3-connected graphs, is to create all 3-connected graphs, and then check minimality for all of them. A better way to approach the problem, is to only generate the minimally 3-connected graphs. A constructive characterization of the minimally 3-connected graphs has been given by [**?**]. The construction uses three operations, and only need K_4 as the starting set. We will describe the construction, and use a slight variation of it to generate the minimally 3-connected graphs for small n.

5.2.1 The Construction

To describe the construction, we need some definitions.

Definition 5.1 (Chording path) *In a graph G, a path P is said to be a chording path, if some edge $e \in E(P)$ chords a cycle C in G, such that C has no intersection with P other than at the end-vertices of e.*

The construction uses 3 operations, and to perform these operations, the following definition of a 3-compatible set is important.

Definition 5.2 (3-compatible set) *Let G be a graph. A set S of vertices and/or edges in G is 3-compatible if it conforms to one of the following three types*

Type 1: *$S = \{x, ab\}$ where x is a vertex of G, ab is an edge of G, $x \neq a$ and $x \neq b$, and no xa-path or xb-path is a chording path of $G - ab$.*

Type 2: *$S = \{ab, cd\}$ where ab and cd are distinct edges of G, and no ac-path, bc-path, ad-path or bd-path is a chording path of $G - ab - cd$.*

[2]The On-Line Encyclopedia of Integer Sequences: `http://oeis.org/A006290`

Type 3: $S = \{x, y, z\}$ *where* x, y *and* z *are distinct vertices of* G *and no* xy*-path,* xz*-path or* yz*-path is a chording path of* G.

The starting set of the construction, is the smallest 3-connected graph, K_4. From this starting set, the three operations are used to construct precisely the minimally 3-connected graphs.

Operation 1 Let x be a vertex and ab be a non adjacent edge. Operation 1 subdivides ab with a new vertex y and joins x to y.

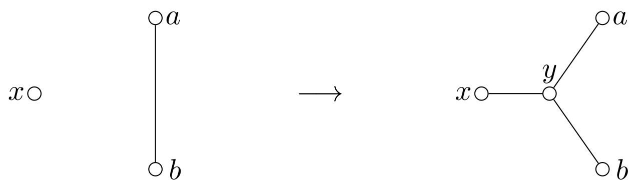


Figure 5.6: Operation 1

Operation 2 Let ab and cd be edges. Operation 2 subdivides ab with a vertex x and subdivides cd with a vertex y and then joins x and y.

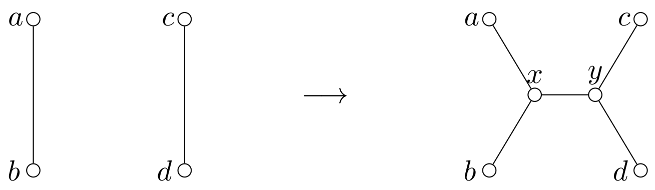


Figure 5.7: Operation 2

Operation 3 Let x, y, z be distinct vertices. Operation 3 adds a new vertex w and joins x to w, y to w and z to w.

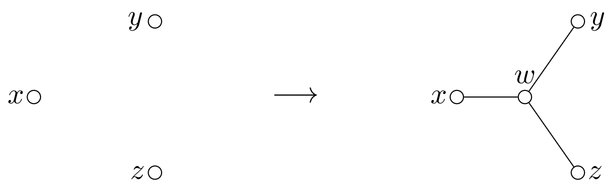


Figure 5.8: Operation 3

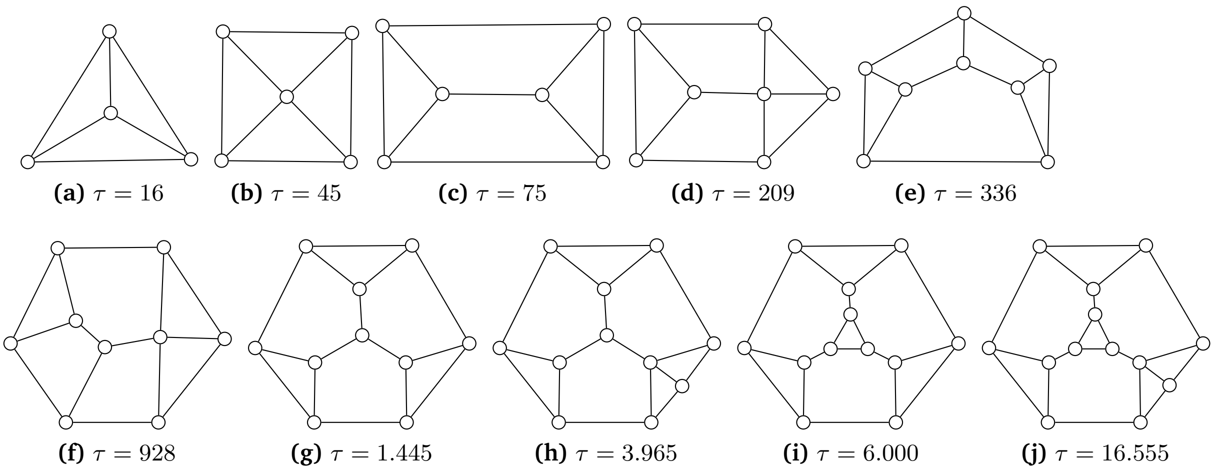
If one wishes to construct exactly the minimal 3-connected graphs, then the 3 operations can only be applied to 3-compatible sets. Let H be a 3-connected graph, and assume we want to construct G from H. If we construct G by applying an operation to any set S in H, then G is also 3-connected. But if H was minimally 3-connected, and the set S was not 3-compatible, then G is not guaranteed to be minimally 3-connected.

5.2.2 *Implementation*

We want to use the construction to generate the minimally 3-connected graphs. In practice, determining if a set S is 3-compatible can be done in polynomial time, but is not simple to do. We will use a simpler - yet not as effective - way to find the minimally 3-connected graphs.

Suppose we have all minimally 3-connected graphs on n vertices, and want to construct all minimally 3-connected graphs on $n+1$ vertices. Instead of only applying the three operations on 3-compatible sets, we will apply them to all possible sets. This results in all the minimally 3-connected graphs $n+1$ vertices, but also results in some 3-connected graphs on $n+1$ vertices which are not minimal. To reduce the set to only contain the minimally 3-connected graphs, we will perform a minimality check on each of them. Determining whether a graph is minimally 3-connected, can be done in polynomial time, and is simple to implement (using Mengers Theorem and Network Flow [**?**]). When the non minimally 3-connected graphs have been removed, we are left with all the minimally 3-connected graphs. If we compare all the minimally 3-connected graphs for isomorphism, we will reduce the set to consist of exactly the non-isomorphic minimally 3-connected graphs on $n+1$ vertices.

A simple Python program was implemented to generate the set of minimally 3-connected graphs on n vertices, and to find the 3-connected graphs with the fewest spanning trees. The program is in **??**. The 3-connected graphs with fewest spanning trees on $4-13$ vertices are shown in **??**.


(a) $\tau = 16$ **(b)** $\tau = 45$ **(c)** $\tau = 75$ **(d)** $\tau = 209$ **(e)** $\tau = 336$

(f) $\tau = 928$ **(g)** $\tau = 1.445$ **(h)** $\tau = 3.965$ **(i)** $\tau = 6.000$ **(j)** $\tau = 16.555$

Figure 5.9: The 3-connected graphs with fewest spanning trees for $n = 4 \ldots 13$, and for each, its number of spanning trees.

5.3 Open Problems

Here we present some open problems and conjectures. It seems intuitively correct, that the 3-connected graphs with fewest spanning trees, have as few edges as possible. The computational experiments in **??** also suggest this.

Conjecture 5.2 *The 3-connected graphs on n vertices with the fewest spanning trees, has $\lceil \frac{3}{2}n \rceil$ edges.*

The connected graphs with fewest spanning trees are trees, and the 2-connected graphs with fewest spanning trees are cycles. It is not clear what we can say about 3-connected graphs, but planarity seems plausible.

Conjecture 5.3 *The 3-connected graphs on n vertices with the fewest spanning trees are planar.*

It seems like the bound given in **??** is too low.

Problem If G is 2-connected, cubic and simple, is it possible to find a better lower bound than $\tau(G) \geq \frac{1}{11}n \cdot 8^{n/4}$?

Chapter 6

Concluding Remarks

The bound given on the number of spanning trees in 2-connected cubic graphs does not seem as tight as possible, and perhaps it is possible to choose a different set of degenerative operations, leading to a better bound.

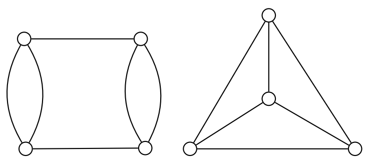
Many classes of graphs exist for which we have no tight lower bounds on the number of spanning trees, and the 3-connected cubic graphs is one of the simplest which gives rise to some interesting complex questions. The structural properties of a graph leading to a large number of spanning trees is very difficult to get an intuition about, and for 3-connected graphs, it was not even possible to state a conjecture about which graphs give the lower bound. It is possible that further numerical experiments might help formulate a conjecture for the 3-connected graphs, but it is also possible that the behaviour is so complex that this will never happen.

We employed an inductive proof technique to show the lower bounds, but there might be more suitable methods for proving lower bounds in more complex classes of graphs. One example is the Laplacian Matrix, which utilizes linear algebra to count spanning trees.

Chapter A

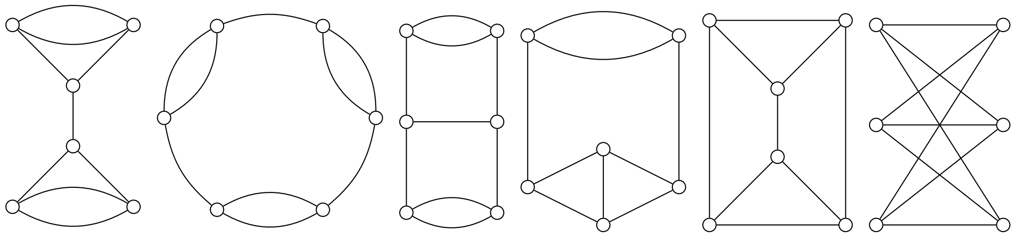
Small Connected Cubic Multigraphs

We show the cubic connected multigraphs on 4 and 6 vertices. These were found by an exhaustive search by hand.



(a) $\tau = 12$ **(b)** $\tau = 16$

Figure A.1: Connected cubic multigraphs on 4 vertices



(a) $\tau = 25$ **(b)** $\tau = 36$ **(c)** $\tau = 45$ **(d)** $\tau = 56$ **(e)** $\tau = 75$ **(f)** $\tau = 81$

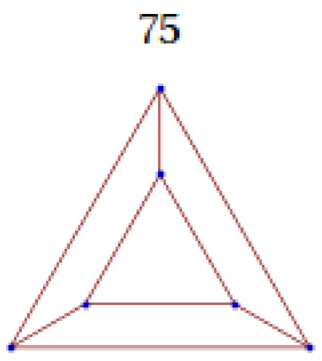
Figure A.2: Connected cubic multigraphs on 6 vertices

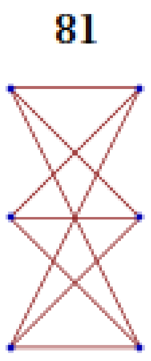
Chapter B

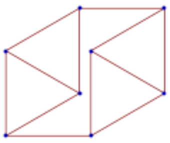
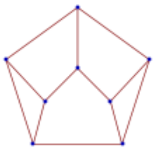
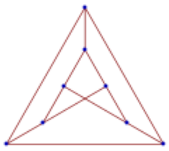
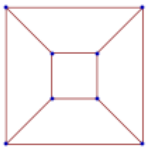
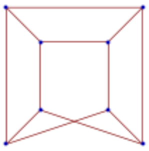
Small Connected Cubic Simple Graphs

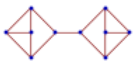
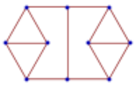
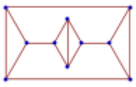
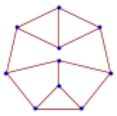
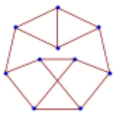
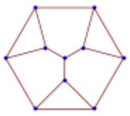
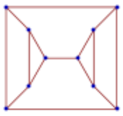
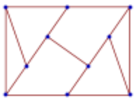
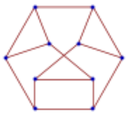
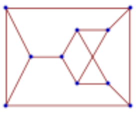
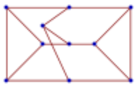
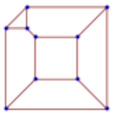
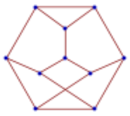
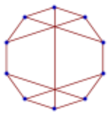
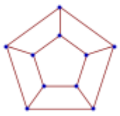
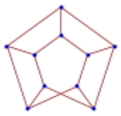
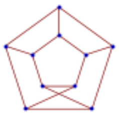
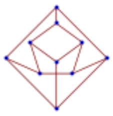
Here we show all simple, connected, cubic graphs on 6 to 12 vertices, and their number of spanning trees. These have been found using Mathematica. Here is the Mathematica code for finding the graphs on 8 vertices.

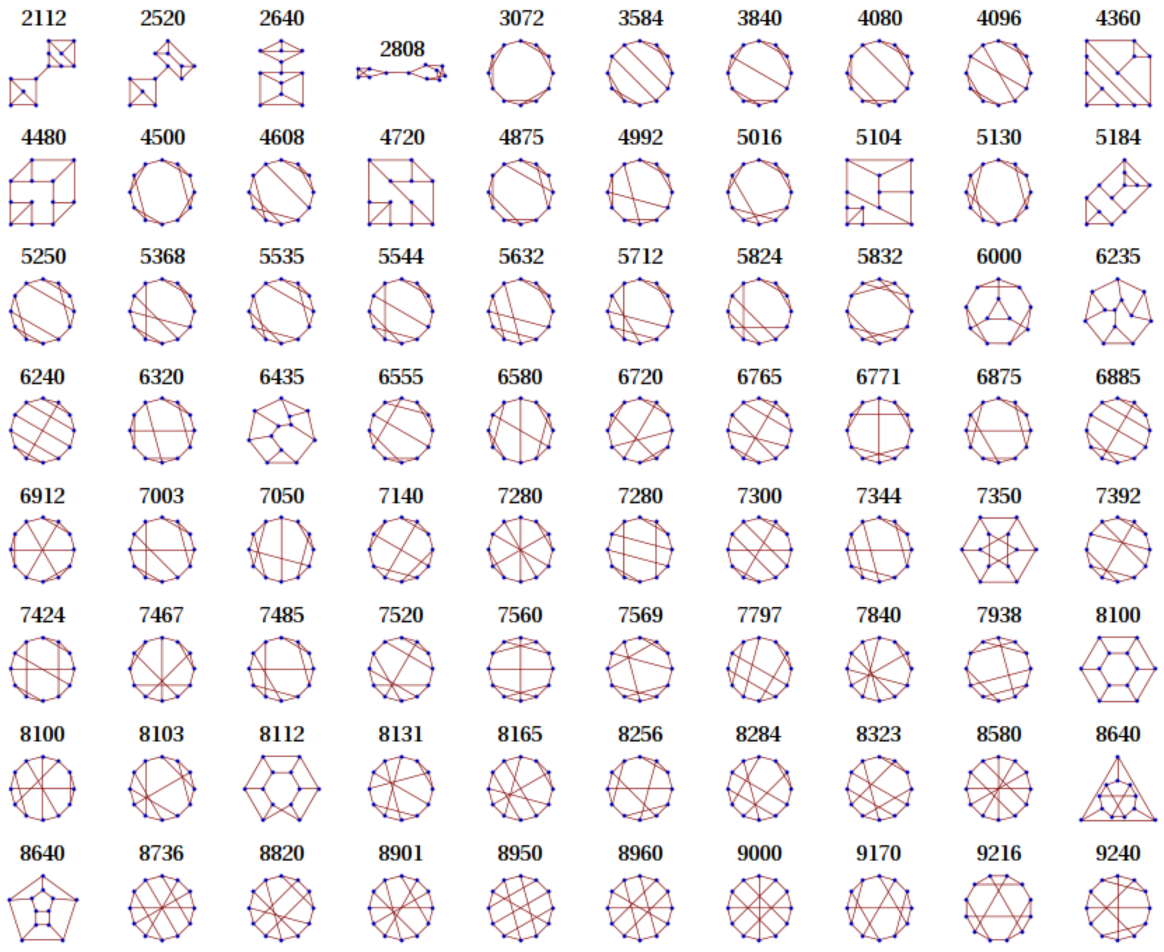
```
count[name_] := GraphData[name, "SpanningTreeCount"];
showCount[name_]:=Show[GraphData@name, PlotLabel -> count@name];
showLabeledCount[name_] := Tooltip[showCount@name,name];
graphs = GraphData[{"Cubic","Connected"}, 8];
pics = ArrayPad[pics, {0,2}, Graphics[]];
GraphicsGrid@Partition[pics,3]
```





256

336

363

384

392


576

960

1080

1160

1224

1445

1463

1488

1573

1599

1600

1682

1717

1728

1805

1815

1881

1920



2112 2520 2640 2808 3072 3584 3840 4080 4096 4360
4480 4500 4608 4720 4875 4992 5016 5104 5130 5184
5250 5368 5535 5544 5632 5712 5824 5832 6000 6235
6240 6320 6435 6555 6580 6720 6765 6771 6875 6885
6912 7003 7050 7140 7280 7280 7300 7344 7350 7392
7424 7467 7485 7520 7560 7569 7797 7840 7938 8100
8100 8103 8112 8131 8165 8256 8284 8323 8580 8640
8640 8736 8820 8901 8950 8960 9000 9170 9216 9240

Chapter C

Python Code

The following code was implemented to generate the small minimally 3-connected graphs. It uses the construction described in ??.

C.1 specialGraphs.py

```
from igraph import *

def CompleteGraph(n):
    g = Graph(n)

    for i in range(n):
        for j in range(i+1,n):
            g.add_edges([(i,j)])

    return g

def Wheel(n):
    g = Graph(n+1)

    # Create spokes
    for i in range(1,n+1):
        g.add_edges([(0,i)])

    # Create circle
    for i in range(1,n):
        g.add_edges([(i,i+1)])
    g.add_edges([(1,n)])

    return g

def Prism(n):
    g = Graph(2*n)
    # Create outer circle
    for i in range(n-1):
        g.add_edges([(i,i+1)])
    g.add_edges([(0,n-1)])

    # Create inner circle
    for i in range(n,2*n-1):
        g.add_edges([(i,i+1)])
    g.add_edges([(n,2*n-1)])

    # Connect circles
    for i in range(n):
        g.add_edges([(i,i+n)])

    return g
```

C.2 spanningTree.py

```
from igraph import *
import numpy

def numberOfSpanningTrees(g):
    # Get Laplacian Matrix
    la = g.laplacian()

    # Get Cofactor
    del la[0]
    for r in la:
        del r[0]

    # Compute determinant
    return numpy.linalg.det(la)
```

C.3 minimal.py

```
from igraph import *
import os
from specialGraphs import *
from spanningTree import *
import numpy
import cProfile

visual_style = {}
visual_style["vertex_size"] = 10
visual_style["vertex_color"] = "white"
visual_style["edge_width"] = 1
visual_style["layout"] = "kk"
visual_style["margin"] = 20
visual_style["vertex_label"] = None
```

```
minNumberOfSpanningTrees = [0 for j in range(20)]
minGraphs = [0 for j in range(20)]

treemap = {}

def saveGraph(g):
    found = False

    numspan = int(round(numberOfSpanningTrees(g))+0.1)

    if treemap.has_key(numspan):
        for h in treemap[numspan]:
            if g.isomorphic(h):
                found = True
                return

    if not isMinimal(g):
        return

    if found == False:
        if treemap.has_key(numspan):
            treemap[numspan].append(g)
        else:
            treemap[numspan] = [g]

def operation1(G,x,a,b):
    G.add_vertices(1)
    y = G.vcount()-1
    G.add_edges(((x,y),(a,y),(b,y)))
    G.delete_edges(G.get_eid(a,b))
    return G

def operation2(G,ab,cd):
    G.add_vertices(2)
    y = G.vcount()-1
    x = y-1
    a = ab[0]
    b = ab[1]
    c = cd[0]
    d = cd[1]
    G.delete_edges((G.get_eid(a,b),G.get_eid(c,d)))
    G.add_edges(((a,x),(b,x),(c,y),(d,y),(x,y)))
    return G

def operation3(G,x,y,z):
    G.add_vertices(1)
    w = G.vcount()-1
    G.add_edges(((x,w),(y,w),(z,w)))
    return G
```

```
def buildGraphsOfSize(n):
    print "Starting", n
    global treemap
    treemap = {}

    if not os.path.isdir("./" + str(n) + "-vertices"):
        os.mkdir("./" + str(n) + "-vertices")

    numn = len(os.listdir("./" + str(n) + "-vertices"))
    if n>4:
        numnminus1 = len(os.listdir("./" + str(n-1) + "-vertices
           "))
    if n>5:
        numnminus2 = len(os.listdir("./" + str(n-2) + "-vertices
           "))

    if numn != 0:
        print "Already computed on ",n,"vertices"
        return

    if n > 5:
        print "Building on", numnminus1 + numnminus2, "graphs"
    elif n == 5:
        print "Building on", numnminus1, "graphs"
    else:
        print "Not building on any graphs"

    if numnminus1 != 0:
        for i in range(numnminus1):
            if i%10 == 0:
                print i
            g = Graph.Read_Pajek("./" + str(n-1) + "-vertices/
               graph" + str(i) + ".net")

            #Operation 1 - Find a vertex, and a non-adjacent
               edge
            for x in range(g.vcount()):
                for e in g.get_edgelist():
                    if e[0] != x and e[1] != x:
                        # Perform Operation 1
                        saveGraph(operation1(g.copy(),x,e[0],e[1
                           ]))

            #Operation 3 - Find three distinct vertices
            for x in range(g.vcount()):
                for y in range(x,g.vcount()):
                    for z in range(y,g.vcount()):
                        if x != y and y != z and x != z:
```

```
                                        if x not in g.neighbors(y) and x not
                                            in g.neighbors(z) and y not in g
                                           .neighbors(z):
                                            # Perform Operation 3
                                            saveGraph(operation3(g.copy(),x,
                                               y,z))
    else:
        print "Need Graphs of size",(n-1),"first"

    # For every graph on n-2 vertices (Operation 2)
    if n > 5:
        if numnminus2 != 0:
            for i in range(numnminus2):
                if i%10 == 0:
                    print i
                g = Graph.Read_Pajek("./" + str(n-2) + "-
                    vertices/graph" + str(i) + ".net")

                for ab in g.get_edgelist():
                    for cd in g.get_edgelist():
                        if ab != cd:
                            # Perform Operation 2
                            saveGraph(operation2(g.copy(),ab,cd)
                               )
    else:
        print "Need Graphs of size",(n-2),"first"

    #print "There is ", len(graphs_lib[n]), " graphs on ", n, "
       vertices"

def saveGraphs(n):
    global treemap
    print "Saving", n
    i = 0
    if n>4:
        print treemap.values()
        for g in treemap.values():#graphs_lib[n]:
            #plot(g, **visual_style)

            g[0].write_pajek("./" + str(n) + "-vertices/graph" +
                str(i) + ".net")
            i = i + 1

def isMinimal(G):
    for e in G.get_edgelist():
        temp_edge = e

        if G.degree(e[0]) > 3 and G.degree(e[1]) > 3:
            G.delete_edges(G.get_eid(e[0],e[1]))
            if G.vertex_disjoint_paths(e[0],e[1]) == 3:
                return False
            G.add_edges(temp_edge)

    return True


def findMinimumGraph(n):
    global minGraphs

    for i in range(len(os.listdir("./" + str(n) + "-vertices")))
        :
        g = Graph.Read_Pajek("./" + str(n) + "-vertices/graph" +
             str(i) + ".net")

        numspan = numberOfSpanningTrees(g)

        if int(numspan) < int(minNumberOfSpanningTrees[g.vcount(
            )]) or int(minNumberOfSpanningTrees[g.vcount()]) == 0
            :
            print "Found graph on ", g.vcount(), " vertices with
                 ", numspan, " spanning trees"
            minGraphs[g.vcount()] = [g]
            minNumberOfSpanningTrees[g.vcount()] = numspan
        elif int(numspan) == int(minNumberOfSpanningTrees[g.
            vcount()]):
            print "Found another graph on ", g.vcount(), "
                vertices with ", numspan, " spanning trees"
            minGraphs[g.vcount()].append(g)

def myPlotter(G):
    G.layout_kamada_kawai()
    G.layout_fruchterman_reingold()
    plot(G,**visual_style)

def plotGraphList(Graphs):
    width = 200
    cols = 3

    num = 0
    for G in Graphs:
        if G != 0:
            if type(G[0]).__name__ == "Graph":
                num = num+1

    p = Plot("test.png",bbox=(0,0,cols*width, (1+int((num-1)/
        cols))*width))

    i = 0

    for G in Graphs:
        if G != 0:
            if type(G[0]).__name__ == "Graph":
                hpos = i%cols
                vpos = int(i/cols)
                p.add(G[0],(hpos*width, vpos*width, hpos*width+
                    width, vpos*width+width),**visual_style)
                i = i + 1

print (0,0,cols*width, (1+int((i-1)/cols))*width)
p.show()
p.save()
```