2009

Quantifying Network Reliability Through Finding An Upper Bound For Graph Integrity Using Graph Coloring

Ian Burchett
Western Kentucky University

QUANTIFYING NETWORK RELIABILITY THROUGH FINDING AN UPPER BOUND FOR GRAPH INTEGRITY USING GRAPH COLORING

by

IAN BURCHETT

2009

A Capstone Experience/Thesis

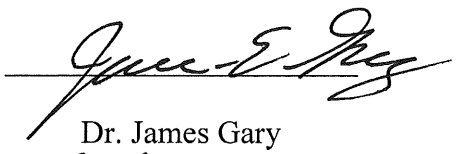
submitted in partial fulfillment of the requirements of

University Honors College at

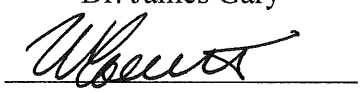
Western Kentucky University

Approved by:

Dr. Mustafa Atici



Dr. James Gary



Dr. Walter Collett

ABSTRACT

Integrity of a graph is defined as $(G) = min_{S \subseteq V(G)}\{|S| + m(G - S)\}$, where G is a graph with vertex set V and m(G-S) denotes the order of the largest component of G - S. This provides an upper estimate of the integrity of the given graph. Using graph coloring, the color sequence of the graph can be generated, with the leading term being the largest component of the graph, the maximal independent set. The determination of the set is too time intensive to be feasible for moderate to large graphs, since there is no polynomial time algorithm to do so. My algorithm completes this task with reasonable accuracy within $O(N^2)$ time. This allows for generation of an upper bound of integrity, and an estimation of real integrity, for even extremely large graphs. With integrity known or estimated, network reliability can be estimated based on their topography. Through comparison of different potential network architectures, network engineers may construct stronger networks based on which network has a higher integrity.

INDEX WORDS: Computer Science, Mathematics, Computer Networks, Graph Integrity, Graphs, Graph Coloring.

QUANTIFYING NETWORK RELIABILITY THROUGH FINDING AN UPPER BOUND FOR GRAPH INTEGRITY USING GRAPH COLORING

by

IAN BURCHETT

Committee Chair: Dr. Mustafa Atici

Committee: Dr. James Gary
Dr. Walter Collett

Electronic Version Approved:

Honors College
Western Kentucky University
May 2010

ACKNOLEDGEMENTS

This project would not be possible without the support, knowledge, and understanding of my thesis advisor Dr. Mustafa Atici. His disciplinary knowledge of mathematics and graph theory was critical in supporting my pursuit of this thesis. Dr. Atici also was pivotal in my selection of this topic for my thesis, and my interest in the topic.

Additionally, all the work that was done by the Honors department staff and the Computer Science department staff on accommodating my thesis hours into my degree plan. I sincerely thank all those involved in my thesis process.

TABLE OF CONTENTS

LIST OF FIGURES

1. Definition of terms

An **edge** is a connection between two vertices. The edge reaches between the two points, allowing a connection between them. There exist directed and undirected edges.

A **vertex is** a distinct point, or node. A vertex can be connected to other vertices by means of an edge. Vertices can represent any distinct body, such as a person, building, intersection, computer, etc.

A **graph** is a set of vertices and edges. A graph is commonly represented as a collection of dots and lines, where the dots are vertices and the lines are the edges connecting them. A graph can be conceptualized as a collection of computers connected by Internet connections. So long as the computer is plugged in to its jack, you are connected to the other machines. A representation of a graph is as below:

Figure 1.1: An example of a graph

A **connected graph** is one such that all vertices are reachable by all others in the graph by some path through the graph. This means that the entire graph is one cohesive body, and has no isolated portions. A connected graph is representative of a computer network

where everyone has a connection with the internet, or the Interstate Highway System, where there does not exist a highway that is by itself, unconnected from the rest of the highways.

A **complete graph,** also known as a full graph or an all-to-all graph, is a type of graph has all its vertices connected with all other vertices. This means that from any one vertex, you can reach any and all other vertices with only one hop, without interacting with any other third vertex.

An **open walk** is a series of edges and vertices that starts at one vertex, and ends at a different vertex. This can be conceptualized as a path you would walk along if you were transversing the graph, from node to node. A walk allows repetition of vertices and edges.

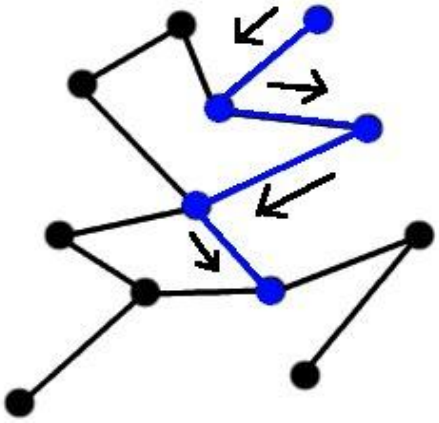


Figure 1.2: An open walk

A **closed walk** is a transversal of the graph as in an open walk, but the start and end vertex is the same. This means you transverse the graph, and return to the vertex that you started from.

A **path** is a transversal of a graph, like an open walk, between two vertices, without repeating any edges or vertices, and not ending on the same vertex you started on.

The **distance** between two vertices is the shortest path that can be travelled from one vertex to reach another.

A **cycle** is a path, but where you start and end on the same vertex.

Graph coloring is a subclass of the field of graph labeling, where the vertices of a graph are assigned “colors” such that no two vertices connected by a common edge share the same “color.” This type of graph labeling represents special cases in nature, problems in the sciences, and has far reaching mathematical and scientific applications, being one of the primary problems in graph theory.

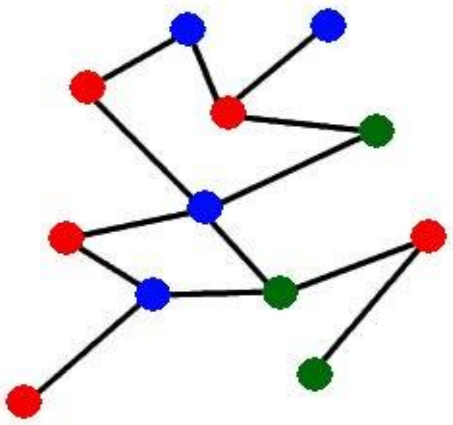


Figure 1.3: A colored graph

A graph's **chromatic number** is the smallest number of different colors that a graph may be colored with, provided that no two colors are bordering each other.

A **bipartite graph** is a graph where all its vertices can be divided into two groups, where each vertex in any given group may connect with any of the vertices in the other group, but not to any within its own group. Below is a bipartite graph, with the two groups highlighted by colors.

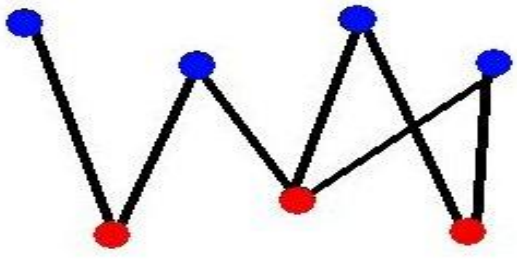


Figure 1.4: Bipartite Graph

A **cubic graph** is a graph where all vertices have exactly three edges connected

The **degree** of a vertex the number of edges connected to a vertex.

A **tree** is a connected graph where no cycle is possible. These stretch out from a common root vertex. Each vertex may only have one parent, but any number of children. The parent of a vertex is a vertex that is closer to the root (top of the graph). The ultimate parent of all vertices is the root node. The vertices that are the furthest away from the root have no children, and are called leaf nodes. These leaves are at the bottom of the graph generally, and are the last nodes in any parent-child series originating from the root. Below is an example of a tree:

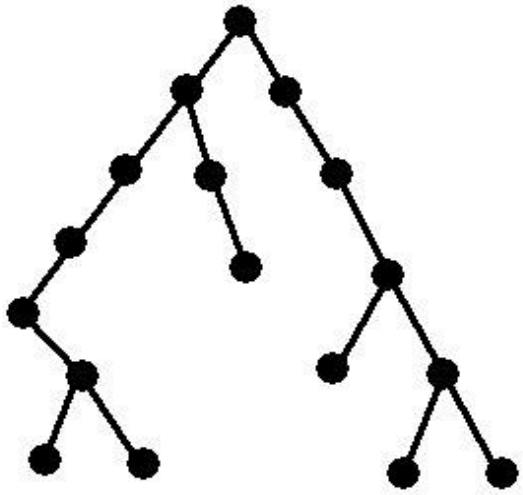


Figure 1.5: A tree graph

2. Network Integrity

Communication is about exchanging ideas between individuals. Getting this information from one to another, particularly when transmitting through technological means, can be problematic at times. These new communication methods rely upon interconnected smaller systems, which mean a message usually must cross between many devices before reaching its recipient. These communication networks are generally effective, but lost calls and uplinks are testament to the reality failures in these systems. Technology allows us to communicate in new ways, across greater distances, and more effectively, but only if the technology is effective enough to enhance communication, rather than inhibit it. One of the primary criteria of an effective communication technology is that this method of

communication must be highly reliable, so users can grow comfortable with the medium and integrate it into their habits.

Reliability within communication networks is a measure of just how robust the network is. A network is a "web" of devices, computers for example, which all are connected by cables and intermediaries, to other computers across a network. This network transmits data from one computer to one or many of the others, but only so long as there is a way to route the data to the other machine. Likewise, a machine can only receive and send messages so long as it is connected to the network. This means that the reliability of a network is a function of just how interconnected this network is, assuming all the links are equally resilient.

Interconnections in the network represent how a computer can be connected to many other computers. Devices are generally connected to a network device such as a Switch, which interconnects many devices, and connects to the larger network outside the room or building. A simple explanation of these interconnections in a standard setup would be that the computer in a room is connected to an Ethernet jack on a wall. The jack on the wall is connected to a cable a couple of hundred feet long, which connects to a network switch, which handles 48 connections in most cases. This switch interconnects all 48 connected to it directly, and also connects to all the other switches in the room via another Switch, which only Switches connect to. This "Distribution Switch" allows all the switches, and all the computers connected to it to communicate with one-another. This Switch, which ensures connectivity of a building or a large section of it at least, connects to a central router. This router handles many connections coming in from many Distribution Switches, which it

handles a bit more complexly than just a switch, and determines where the traffic coming in from the Switches goes. The Router also allows all the computers on all the switches, all the way down the hierarchy to communicate with one-another. The Router can be connected to other routers, etc, across an extensive network, but ultimately these are often connected to the Internet through an ISP Uplink. On larger networks, this means the router connects through a fiber-optic modem to a larger modem elsewhere, which connects to the large continental fiber network, administered by the government and large phone companies. This fiber network, and all the networks like above connect together to make what we know today as the Internet; simply a large interconnection of computers, routers, and switches, with wires, antennas, dishes, and fiber-optics linking them all together.

Using this relationship, we can see how networks and graphs inter-relate. The computers, routers, switches, etc are the vertices within the graph, and the fiber-optics, Ethernet cables, dishes, etc are the links between these vertices. The diagram below demonstrates such a hierarchy, which we see can easily be converted into a graph representation.

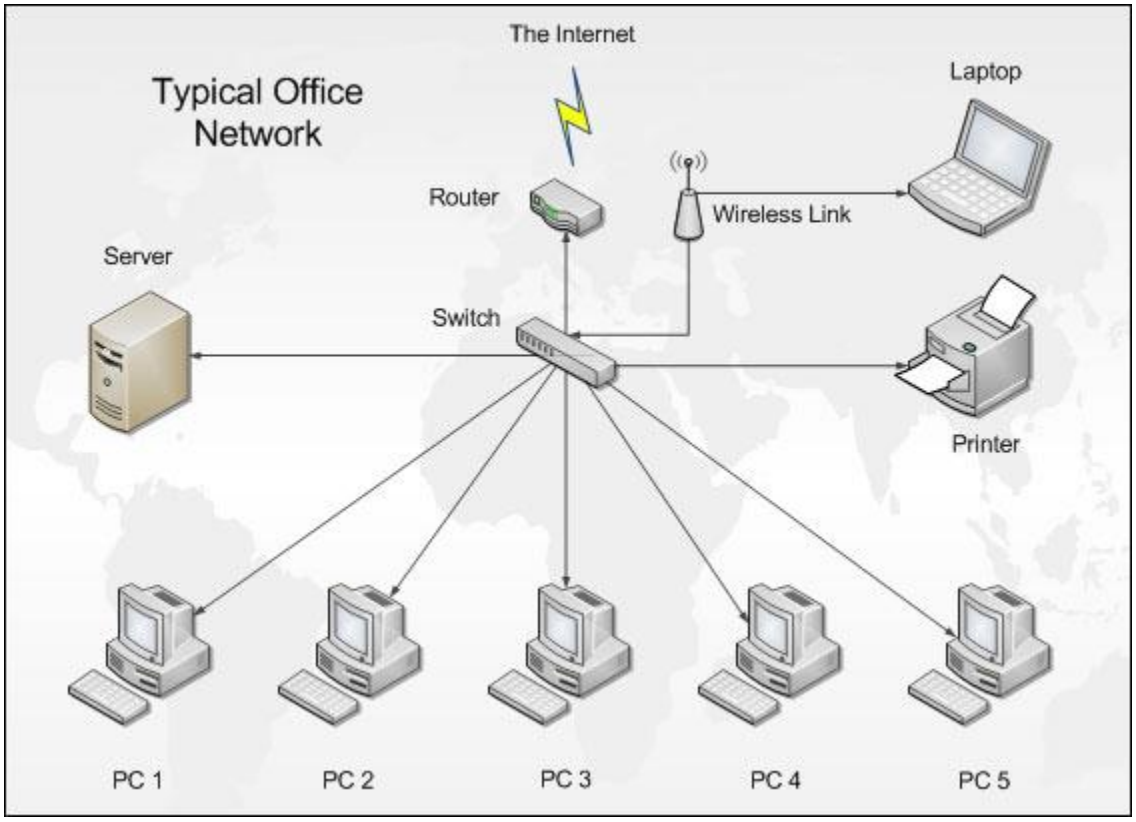


Figure 2.1: An example network topology

These various interconnections are a focus of much interest. These links are the single most expensive part of the network, with this being their primary limitation. In an imaginary world, the perfect network would connect each computer directly with every other computer in the world. This system is not realistic, because it would not only be impossible to secure and administer, but also the sheer amount of wires, dishes, or however the connections would be generated, is not possible. With any network, a network of this sort where every node is connected to every other, if given N computers for example, there would be $\frac{N(N-1)}{2}$ connections between them. For example, if there were 100 computers in an office, and they were all connected, there would be 4950 wires connecting them all together. This is

unreasonable, even on networks the size of a few offices. A university's network, which might have around 50,000 connections, would require 1,249,975,000 wires to connect them, which is very much unreasonable.

To combat the high cost of connections and still achieve connectivity between machines, the obvious solution is to use as few connections as possible. Trying to achieve the absolute minimum amount of connections would be the lowest cost, since all users would be connected, and the least wire would have to be run, for example. A graph such as this would be a tree graph, where even one broken connection would sever the rest of the network, leading out from the root node (the internet uplink). The idea that the fewest connections is best gets many in trouble though, in that keeping only the costs of building the network in mind will lead to a very "lean" network with few connections that only works in optimal conditions. With no redundant connections, and only one connection going to each machine or network node, the resistance to damage, and reliability of the network would be minimal. If only one connection were broken in the right spot, the network would be split in two, with only half of the remaining users connected. Likewise, if connection were cut at the user side, only one break would be necessary to sever a machine from the network.

A fine balance must be struck then between the cost of connections, and the reliability of the network. No one wants to pay for the network where everyone is interconnected, but no one wants to rely on a network where one broken wire results in half of the network going down. Finding this balance is of high interest to those within the networking field, and several different strategies have been put forth, but usually due to costs, the fewer connections, weaker networks are far more common, since they work in theory, at least until

the first problems. The perception that network problems are common, and that fixing problems when they occur is the correct way to approach this problem, is a result of the community's familiarity with this setup. A better approach is to increase a network's reliability in general through analysis of the network, and working towards shoring up weaknesses, or re-doing the network topology entirely. This is what my project hopes to achieve: analyzation of a network's reliability, enabling quantification of a network's strength. Through creation and analyzation of possible network topologies, businesses and network engineers will be able to create networks with fewer connections required to achieve the level of integrity that they are comfortable with implementing and maintaining.

3. Graph Coloring

Before we discuss how to estimate the integrity of a given graph, we must first introduce the foundation on which my work is built upon. The work I have done is based on Graph Theory, as I have introduced before, but more specifically, my technique utilizes concepts from the field of Graph Coloring. Graph Coloring is focused on assigning unique signifier for each vertex, to classify vertices based on whether they are touching or not, by assigning them “colors.”

Under graph coloring, each vertex which borders other vertices through edges cannot possess the same color as any other vertex it borders. This means that the color of a selected vertex (blue for example), cannot be the color of any of its neighbors. This does not restrict

the color of any vertex that is at least one vertex away from the blue vertex, other than the color restrictions that the vertices around it place upon its color.

So why color graphs? Graph coloring has applications such as representing mathematical relationships, ensuring that certain things represented by the vertices are not in contact with one-another and otherwise providing an enforced system of representation for any system that can be applied to its framework, for future research applications. In our case we will use graph coloring to estimate graph integrity through a relationship between graph coloring and integrity of graphs, which I will demonstrate. My work specifically related to graph coloring and these colorings corresponding color series.

The **Chromatic number** of a graph G is the minimum number of colors needed to color the graph G and usually shown with $\chi(G)$.

A **color sequence** of k-color (n_1, n_2,…,n_k) is a listing of the quantity of nodes of each color, in descending order, that are present on a legally colored graph. In the following examples, we see color sequence (1,1,1,1,1,1,1,1) of 8 colors, (3,2,2,1) of 4 colors and (5,2,1) of 3 colors. Depending on the number of colors we will have different color sequence for same graph. The colors are not especially important unless importance is placed on them based on the application, but in general they just delineate the vertices into categories that are isolated from one another. A **maximal coloring sequence** of k-color is a sequence of length k that is the result of seeking to have the most vertices of a certain color present, and would look something like (5,2,1) for a 8-vertex graph with 3-colors in the following example. That is, there is no other color sequence of length 3 lexicographically bigger than (5,2,1). This

would mean that there are five vertices with the same color, and that these five are isolated from one another.

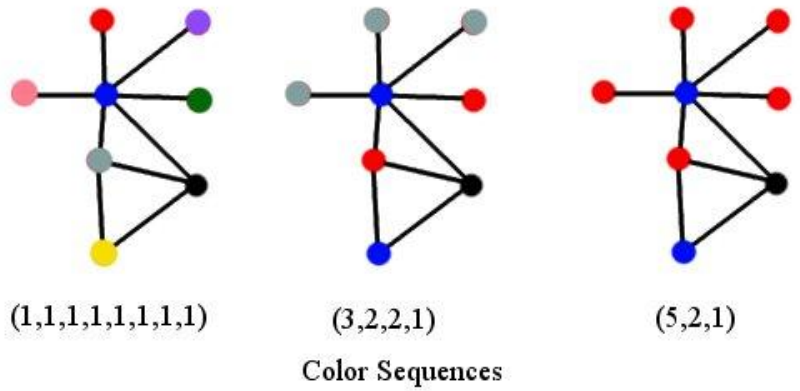


Figure 3.1: Graph Colorings

Below we show all the possible 3-colorings of given graph G. The colorings are progressively more maximal as we progress from the left to right, with the rightmost coloring being the maximal coloring for this graph.

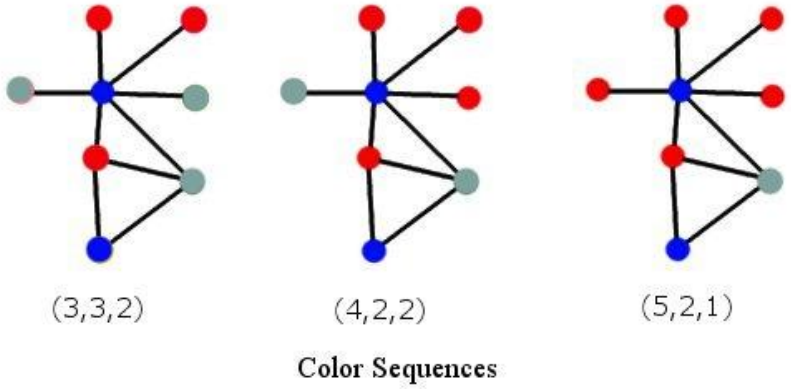


Figure 3.2: All possible 3-colorings of graph G

The following theorem gives the relationship between chromatic number and the graph's color sequence:

Theorem 1: *Every graph G with chromatic number 2 has a unique color sequence.*

Proof: Let G be a given graph with $\chi(G) = 2$. Hence G is bi-partite [1].
One can easily observe the following about the distance between two distinct vertices of colored graph G.

$$d(u, v) = \begin{cases} even & \text{if u and v have same color} \\ odd & \text{if u and v have different color} \end{cases}$$

Suppose there are two different color sequence of the G, say $cs_1 = (n_1, n_2)$ and $cs_2 = (m_1, m_2)$. Without loss of generality we can assume that $n_1 > m_1$. Since the graph G is a bi-partite, $V(G) = V_1 \cup V_2 = U_1 \cup U_2$ where $|V_i| = n_i$ for $i = 1, 2$ and $|U_i| = m_i$ for $i = 1, 2$. Since $n_1 > m_1$ there must be at least one vertex v, in V_1 which is not in U_1. So v has different colors in cs_1 and cs_2. Pick a vertex which has same color in both sequences, say u. Then from v to u there are two different paths, one of them has length odd and the other has length even. Hence there is cycle of length odd from vertex v to v. This contradicts with a fact that G is bi-partite.

4. Finding an Upper Bound for Graph Integrity

Our problem is one of efficiency. We must find the graph with the highest resilience against removal of nodes, while still using the fewest connections possible. Our approach to

measuring this resilience, also known as integrity, is to utilize a unique approach of achieving the highest color series possible on a given graph.

Definition of integrity of a graph G is given as follows:

$$I(G) = min_{S \subseteq V(G)}\{|S| + m(G - S)\}$$

Where |S| indicates number of vertices in set S and m(G-S) is the number of vertices of largest component in G-S [2].

Example:

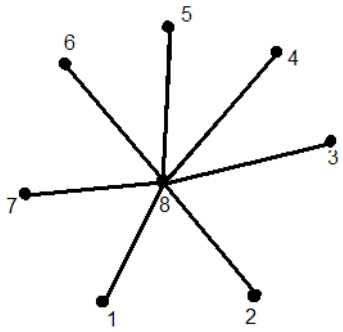


Figure 4.1: An example star graph

Set of vertices of the above graph is V(G)={1,2,3,4,5,6,7,8}. Among all possible subsets of V(G), if we take S={8}, then m(G-S)=1. Hence I(G)=1+1=2.

For any given graph G, to compute the integrity of G, we need to know which subset S will give minimum summation |S|+m(G-S). So if graph G has n vertices, then there are 2^n possible subsets of V(G). For example for a network with 1000 computers, we need to

compute all the 2^{1000} subsets to determine its exact integrity. Such number of computations is going to take quite some time.

Since computing exact integrity for a graph with large number of vertices is not feasible within reasonable time, we must try to find a bound for the integrity of the graph. For such a bound, we will use the maximal color sequence of the graph to determine an upper bound for the integrity. This bound will also serve as an estimate of the integrity of the graph. Let G be a graph with maximal k-maximum color sequence (n_1,n_2,…,n_k). Hence $V(G) = V_1 \cup V_2 \cup ... \cup V_k$, where V_i contains $n_i$vertices which have same color. Then we can give the following upper bound for the integrity of G: Define set $S = V_2 \cup ... \cup V_k$, then m(G-S)=1 since $V_1$is an independent set. Therefore we have $I(G) \leq |S| + 1$. Hence we can give the following theorem:

Theorem 2: *Let G be a graph with maximum color sequence ($(n_1,n_2,\ldots,n_k)$ with k-color. Then $I(G) \leq n_1+1$.*

An *independent set* is a group of vertices that are not connected directly within the graph. This equates directly to the most common coloring within a graph colored with the fewest colors possible. This maximal coloring strategy is what we will utilize to seek the largest independent set possible.

Example: Here are two graphs to demonstrate finding of an upper bound for the integrity of graphs:

Figure 4.2: Sample graph G

Graph G consists of eight vertices and nine edges, and as such can be represented as G(8,9). This graph is neither complete, nor a minimally connected graph, but something in-between, a fairly average graph that might be encountered in a networking situation. G has been colored maximally (with 3 colors) on the right, with a coloring sequence of (4, 3, 1). Take set S to be vertices with color orange and brown. Hence |S|=4 and m(G-S)=1. Therefore $I(G) \leq |S| + 1$ = 4+1 = 5. On the other, for all possible 2^8 subsets of V(G), the minimum summation |S|+m(G-S) = 5. So our upper bound indeed is equal to the exact integrity of G.

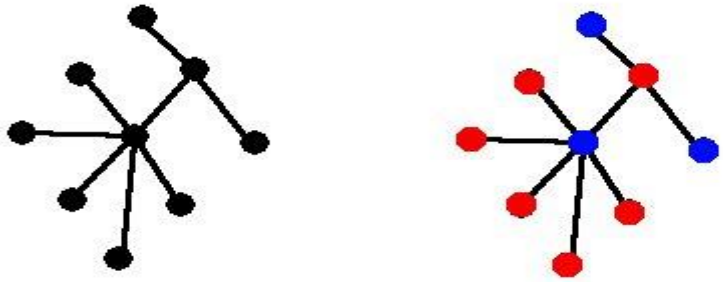


Figure 4.3: Sample graph H

Graph H has nine vertices, and only eight edges, making the graph less connected. The general rule is that the lower the number of edges to vertices, the less connected the graph. So, H is a (9,8) graph, and is a bipartite, so it is colorable in two colors. The maximal coloring sequence of H is (5,3). With an independent set of five, an upper bound for the integrity of the graph is 3+1=4. On the other hand if we take S to be set of vertices with degree 6 and 3, then I(H)=2+1=3. So our bound is very close to the exact integrity of the H.

Though these graphs are small and relatively trivial, on larger graphs the maximal coloring in order to discover the true integrity of the graph is not so easy to figure out. Even more importantly, there is no known way to confirm that a given graph coloring is the maximal coloring sequence without comparing it to each and every other possible coloring. Even for a small graph the number of possibilities is quite high. As the number of vertices and edges goes up, the number of possibilities increases extremely rapidly. As such, a program to compute all possibilities and try them is not feasible on all but the smallest graphs. In fact, as graphs approach only several dozen, the computing time increases to years, even on state of the art computers, making such calculations unreasonable to compute.

Polynomial Time: an algorithm is computable within polynomial time if the algorithm can be computed within a number of steps that is a function of the number of elements being calculated on. For example, an algorithm that takes $4N^2 + 2N + 1$ steps to complete would be a polynomial time algorithm, because the number of steps is based on N. These problems are typically solvable with modern computers, for all but the most extreme cases.

Exponential Time: an algorithm is computable within exponential time if the number of steps required to run the algorithm has the number of elements concerned within the algorithm as an exponent. For example: an algorithm that takes $10^N + 4N$ steps to complete is an exponential time algorithm. These algorithms, in that their complexity increases exponentially as the number of elements increases, very quickly become incalculable on any computer.

Such problems without a reasonable way to solve them are generally considered to not be solvable within polynomial time. My research on determining graph integrity very quickly became determining whether there is a reasonable polynomial algorithm to determine k-color coloring sequence and k-color maximum color sequence.

Though much research was done, and many things were tried, my conclusion is that the problem of finding a graph's color sequence in fact has no polynomial algorithm, and the problem (k-color maximum sequence) falls within the subset of problems where there random solutions can be generated, but they cannot be verified within polynomial time. The maximum color sequence of a graph cannot be extrapolated through any relationship of the connections of the vertices, the number of edges, vertices, the ratio of the two, or any number of other things that a graph's representation demonstrates. Likely this is due to graphs generally being non-uniform, and very complexly connected. I considered whether in special circumstances I would be able to determine the k-color maximum coloring sequence on specific graphs, or types of graphs, and in fact have discovered a way to color bipartite graphs, which I will discuss later.

Decision problem is a problem that has a "Yes" or "No" answer. Here are some examples for decision problems:

Example:

Prime: For given positive integer n, is n prime number?

k-Color Maximum Color Sequence: Given graph G with n vertices and integers $n_1, n_2, \dots, n_k$, is $(n_1, n_2, \dots, n_k)$ the maximum color sequence of G?

NP classification of decision problems includes those problems which can meet the following requirements: Any given randomized (nondeterministic) solution can be verified in polynomial time (quickly). These problems are generally of the type that do not have a polynomial time deterministic solution, and include many of the biggest and most important unsolved mathematics and computer science problems. Since these problems cannot be deterministically solved in polynomial time, this type of solution cannot be applied to the science, nor written into any programs that will complete within our lifetimes, even for trivial cases. An example of this is that an exhaustive search for the maximum coloring sequence on a graph, which will find the maximum coloring sequence once it completes about 2^N steps, where N=the number of vertices within the graph. This means that for a graph, or network, about the size of Western Kentucky University's internal network, with over a thousand active connections, would take about 2^{1000} steps to complete. With a computer that has a modern processor running at 3GHz, processing optimally $3*10^9$ operations per second, and with nothing else running on the processor, and each step in the algorithm taking only one

cycle, this type of calculation would take about $2^{1000}/(3*10^9)$ seconds, or $2^{1000-12}/3$, or approximately 2^{988} seconds, or about $8.2897422*10^{289}$ years in the absolute best case. For scale, a billion years is of course only 10^9 years. This is what it means for the solution to not be computable within reasonable time; the calculation would be completed after hundreds of orders of magnitude longer amounts of time than have passed in the universe based on current estimates. Since the problem is within this class, it becomes obvious that the best we can do with current computational methods is estimate, or try to get "the best we can" with algorithms that do not calculate every possibility.

As for the problem being NP-hard, the problem falls within this category, which can be simply explained as "solving this problem is equivalent to solving every problem within NP-class." NP-hard is special in that the solutions to the problems are not verifiable with a randomized polynomial algorithm either, unlike NP problems which must have such a solution. This means that when you have a candidate for an answer to the question of whether this coloring sequence is the maximal one, you cannot confirm whether this is true or not without seeing all the possibilities. There is no way to look at a graph and determine the coloring is maximal unless all possibilities are considered. This inability to verify solutions means that there can be no non-deterministic solution generation method for this problem. This means that we cannot, for example, generate a random coloring of the graph and then easily look and see if this is the maximum coloring sequence or not. This means that again, the only way to see a solution is to compare all possible colorings, which is not reasonable within polynomial time.

The Maximum Independent Set (MIS) problem is equivalent to our problem in many ways. Firstly, the goal of maximal coloring and the MIS problem are the same, find the largest group of elements within the set (graph in our case), that are independent of one-another, but within the set (graph). This set is exactly the first element of the maximal color sequence. Since for our application of the coloring sequence we are not concerned with the other colors in the sequence, only the number of elements within all the other subsets (colors), in the graph. To put it plainly, if we know the maximum independent set (most common color when the graph is colored maximally), we know that the number of the rest of the nodes (the ones removed to find the set), is equal to the number of nodes within the graph, minus the number of nodes within the maximum independent set. We can then gather an upper bound for the integrity of the graph from this quantity. So finding maximum color sequence is at least a NP-hard problem.

5. My Algorithm

Due to the fact that any type of exhaustive search for the maximal coloring sequence is not feasible, we require an estimation of the maximum color sequence of a given graph, so we must use an estimation algorithm to generate an estimate. My algorithm achieves this through achieving approximately $O(N^2)$ speed when searching from every starting point, which enables even large graphs to be processed very quickly. This means that the number of steps required for the estimation is only the number of nodes squared (N*N because I search N nodes starting from each node once, where N is the number of nodes within the graph). For the aforementioned WKU network, my algorithm would take just 10000^2 steps, and

would be calculable on a 3GHz processor in less than a second; using the same time estimates as before, in the complete search case. This is of critical importance, in that this would enable different network designs to be constructed and tested in nearly real-time, enabling even automated programs to perhaps generate the best graph given certain parameters, utilizing this algorithm.

The algorithm works by coloring the graph utilizing a Breadth-First-Search (BFS) advancing strategy, which proceeds through the graph, coloring as it goes deeper. BFS is used normally to find a certain vertex within a graph, but by coloring the entire graph (as if the search is exhaustive, and the vertex is not found), the entire graph can be covered, and colored in our case. Coloring using BFS means that the graph is colored in a certain fashion and order, as I will explain. Firstly, you begin at a certain node, which is the start node. When you first begin coloring, each node is considered to have a state of *uncolored.* Each time you search a node, that node goes to a *coloring* state, and when the node has been completely colored, it is put into a *colored* state. These states are important, in that the BFS algorithm uses them in order to know how to proceed.

Beginning with the start node, we start coloring. The start node becomes *coloring.* All its neighbors (vertices that are connected to it directly by an edge), also become *coloring,* and are added to a list of nodes which the algorithm keeps track of, that are currently *coloring.* For the sake of visualization, *colored* are shown in white, *coloring* are grey, and *uncolored* are black.

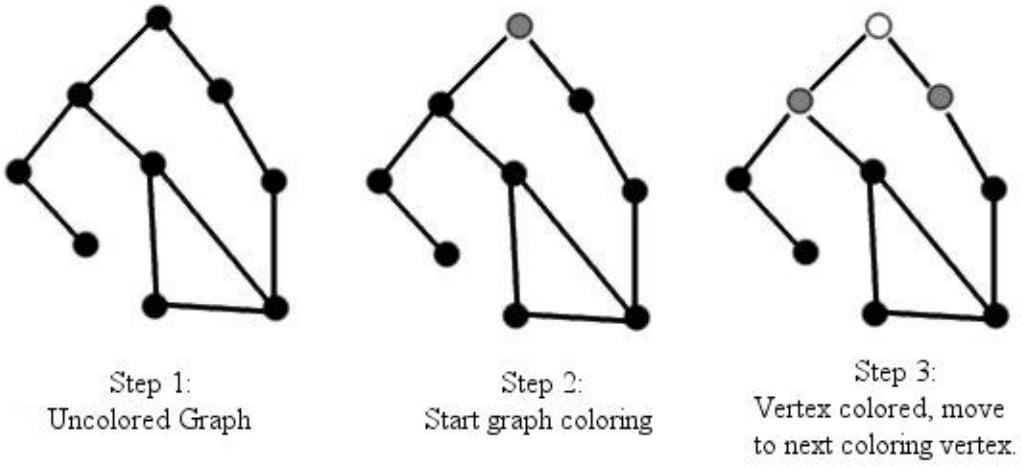


Figure 5.1: Transitional States in BFS Progression

Once a node has been set to *colored* state, the algorithm moves to the next node that is *coloring,* as in the order that we turned them into *coloring* (first in, first out, like a queue). The graph will eventually be completely *colored* as the method proceeds, in the order as depicted below, with the small arrow pointing at the currently active node of the search:

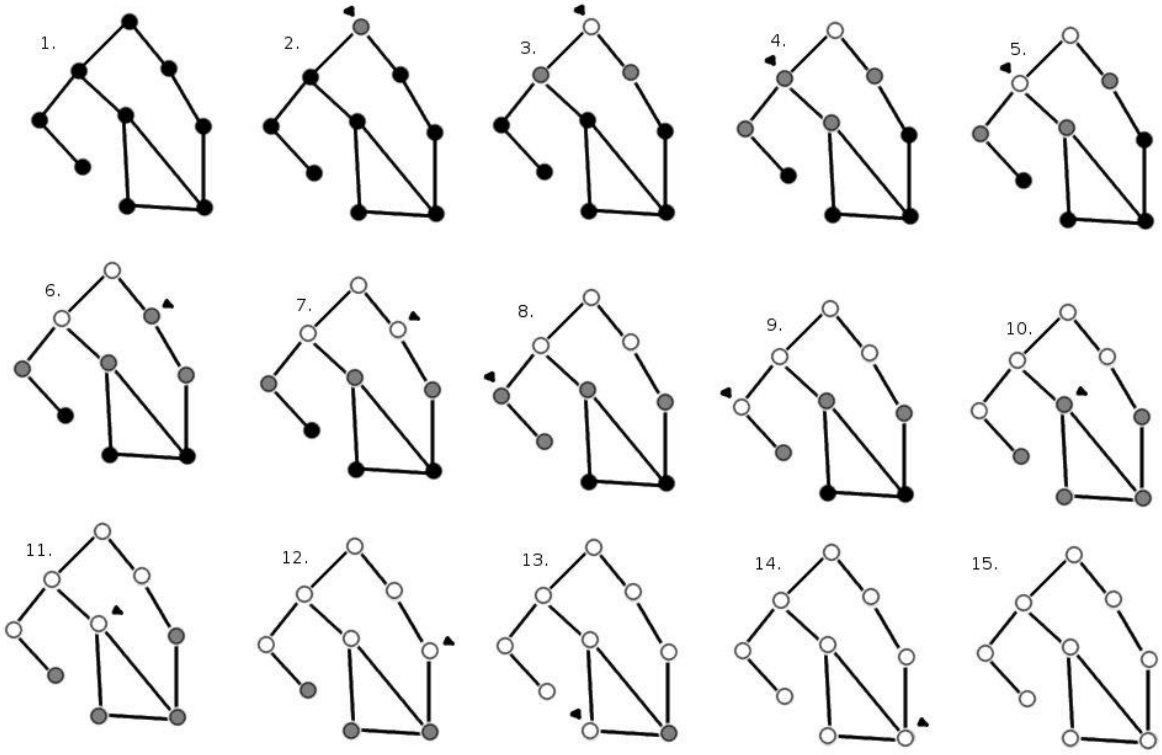


Figure 5.2: BFS Traversal of Graph

Utilizing the BFS sequence of node coloring, the algorithm will work its way through the graph. The node is colored after all its neighbors are set to *coloring*, but before the current node is set to *colored.* The algorithm will attempt to color utilizing a Greedy Algorithm, as it proceeds through the graph, trying to keep the number of colors down to a minimum. If at all possible, the first, most common color is used, proceeding down the sequence of most common colors, until if necessary it will use a new color on the graph. Whenever this coloring is selected, the algorithm checks the other vertices adjacent to the current vertex to determine which colors are off limits for it to use (since no vertex may have the same color as its neighbors), and uses this information in its attempt to determine which

color the vertex should be. Once the vertex has its color, just as in BFS, the next *coloring* node is visited, and the coloring is set on this node, etc, until all the nodes are colored.

The Greedy Algorithm that this coloring algorithm utilizes means that whenever a color is being selected, it will try always to achieve the *best* outcome based on the current situation whenever the node is colored. This means that the algorithm does not base its decision upon past events or predict future ones by extrapolating possibilities, but instead makes the best decision in the short term. In the case of coloring the graph this is acceptable, because upon coloring each node, we check to make sure each coloring is legal, making it fine for us to not plan ahead in terms of making sure a coloring is going to work a certain way from the start. This lack of planning ahead and optimizing the coloring completely is why this algorithm is viable, and completes so quickly.

The downside of this Greedy Algorithm is that constant decision making on the local scale does not always generate the optimal solution for the whole graph. The best choices are made in each decision, but the lack of planning and an overall strategy can get the algorithm in trouble, in terms of maintaining accuracy. Fortunately, the algorithm works well with BFS transversal of a graph, and with the inherent rule that any neighboring vertex cannot share a color with its neighbors. This inherently means that we must know the color of our neighbors before we know the restrictions on which colors the current vertex can be colored, so any intuitive algorithm would have to base its decisions in this way, which the greedy algorithm does as a rule.

For any case that the algorithm finds itself within, it will choose the most common color for the current node if it is legal, proceeding down the list of color commonality until it

must choose a new color. The preference towards the first coloring being colored if at all possible is because the real importance of the coloring is to find the Maximal Color Sequence (the leading element being the Maximum Independent Set), so that we may estimate the maximum color sequence as accurately as possible. By coloring the node the leading color whenever possible, this estimate proves accurate.

Once the graph is completely colored, the coloring is stored, and the algorithm runs, starting the coloring process one time, starting from each node within the graph in turn. Each time the data is stored, and once all the colorings have been computed, the color sequence with the largest leading element is selected. This coloring has what the algorithm has calculated is the largest independent set, and based on this; an upper bound for integrity set is computed using the formula for integrity explained earlier. This is the algorithm's best estimation for the integrity of the graph, and is an absolute upper bound, with the integrity being equal to or less than this estimate in all cases.

6. Statistical Analysis

Through the generation of random graphs, the estimation of their integrity with my algorithm, and calculating their actual integrity through an exhaustive calculation program, I have analyzed the accuracy of the estimation of the upper bound of the integrity of my algorithm. This estimation is not complete though, in that it is only possible to compute the absolute integrity on rather small graphs. Were more computing resources available I would be able to calculate estimates for graphs going up perhaps a hundred more nodes, but the increase in required computational power and time are so rapid as you increase the number of

nodes, calculation of accuracy in general for larger case graphs is not possible with current computing technology, using current absolute integrity calculation software.

For this analysis, 250 graphs were generated randomly by a computer program. For each graph generated, the graph was fed into the program to my algorithm which provides an estimate for integrity and an absolute upper bound. Once this finished, the file was reformatted, and run through the program which calculates the exact integrity of the graph. These results were tabulated, and statistically analyzed after this work was completed.

First, let’s consider the relationship between the estimates of integrity and these graphs’ actual integrity. I charted the relationship as below, with the integrity on the y-axis, and the number of nodes in the graph on the x-axis. The red data are the algorithm's average estimates for each node count. The blue data are the actual integrity of the cases. As you can see; the integrity trends higher as the number of nodes increases in this case, for my test data. At the same time, the estimate follows the increase upwards, deviating slightly higher as the node count rises.

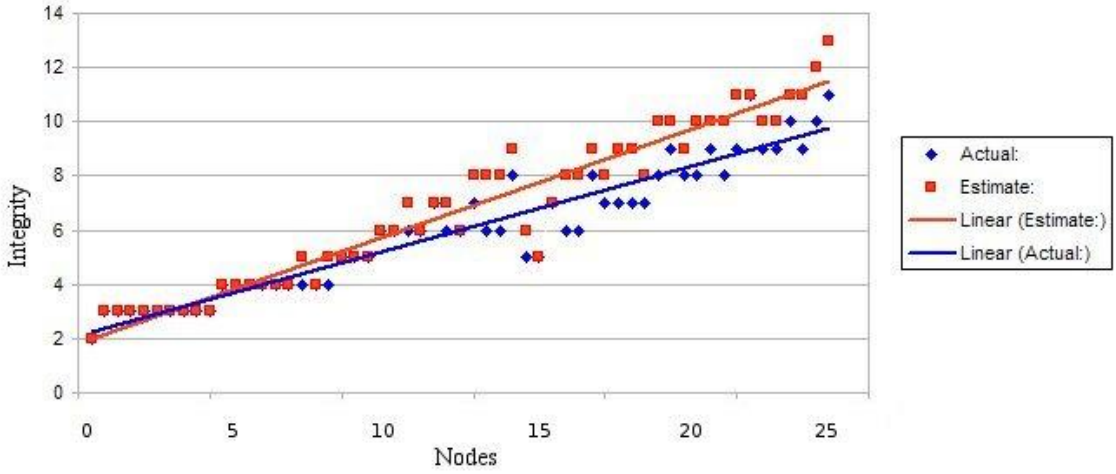


Figure 6.1: Estimate vs. Actual Integrity

While the size of the deviation increases as the number of nodes increases, the percentage of deviation remained constant. Certain structures within the graph, such as triangles, tetrahedrons, square, etc, and the increasing complexity of the graphs may explain why some graphs deviate further from the estimate than others. Even though this is the variable that effects the deviation the most, there seemed to be a fairly steady increase in inaccuracy as the number of nodes increased. Though the number increased, this maintained a proportionate relationship with the number of nodes.

The following graph shows the relationship between the number of nodes, and the deviation in integrity from the actual that the estimate was. In all cases the estimate was higher than the actual integrity, so these deviations reflect how much higher the estimate was than the actual figures. The fact that all cases have a higher integrity estimate than the actual integrity is expected, and follows from the fact that this algorithm finds an estimate, but the estimate is also the upper bound of possible integrity.

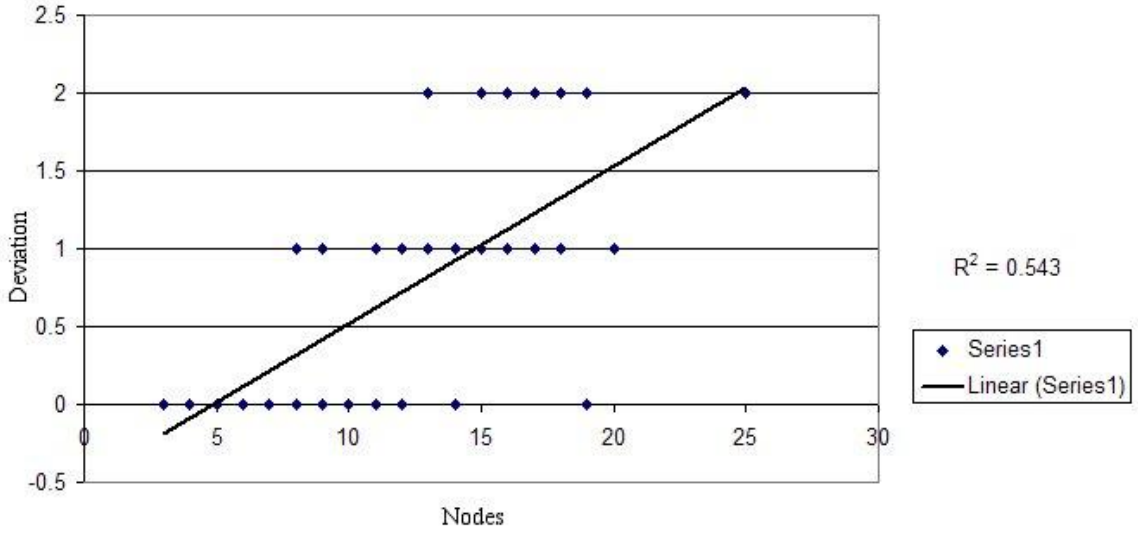


Figure 6.2: Deviation from Actual Integrity

The deviation rises as the number of nodes increase, but the proportionate deviation does not rise above two for the tested graphs. For most cases on graphs less than ten nodes, the algorithm was almost always perfectly accurate in fact. Following the trend, the perfect accuracy dropped away as the number of nodes increased. The increase in deviation follows the trend line shown, in general, with the R-squared value of 0.543. Based on this value, we can see that the relationship is fairly linear as the number of nodes and integrity of the graph increase.

The real correlation present is that of the correlation between the estimated integrity and actual integrity. The correlation was very strong, and the relationship very linear:

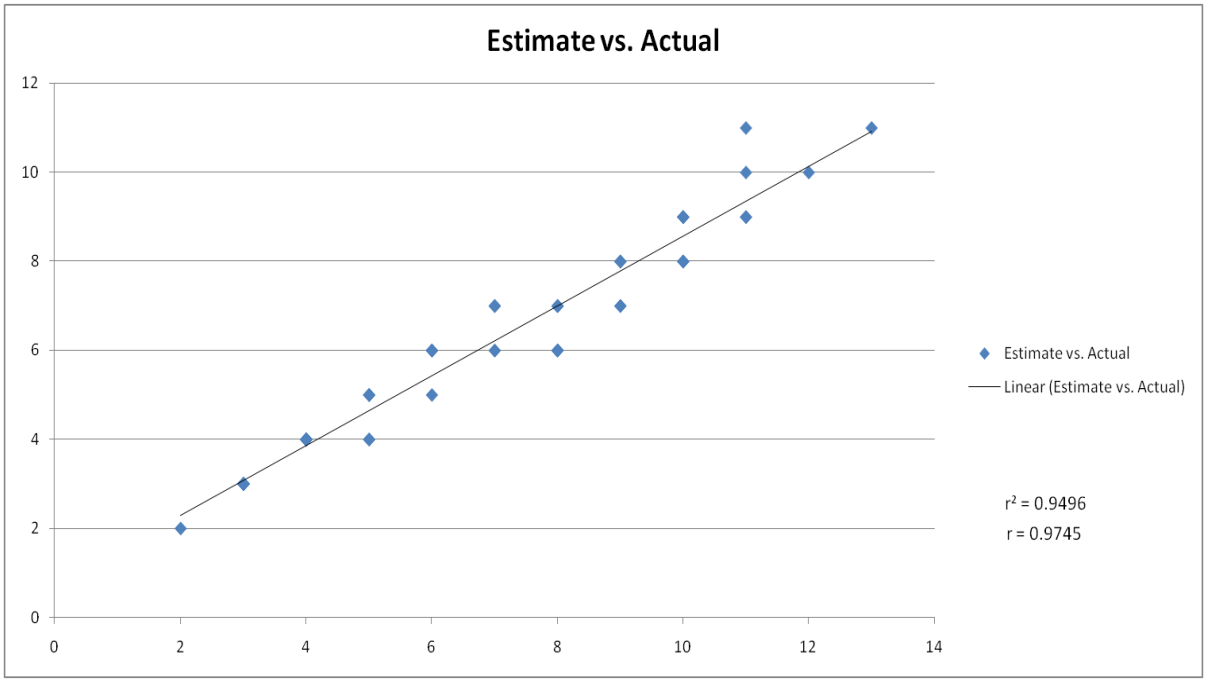


Figure 6.3: Estimated Integrity vs. Actual Integrity

Based on these analyses, we see that the estimation of the graph integrity is fairly accurate, with some deviation, and the correlation between the estimate and the actual integrity is very strong, and very linear. Based on these results we also have empirical support for the fact that this algorithm will always generate an upper bound, making this useful for estimating integrity, in that there is a confirmed bound on the integrity, rather than a +/- deviation, which would have been much more troublesome.

7. Conclusions

Calculating the integrity of a network or a graph cannot be done within polynomial time to perfect accuracy. Very minor graphs can be calculated this way, but for computer networks in the field, the sizes of these graphs are too large to be calculable. My algorithm provides an estimate of the integrity of the graph; with the estimate being an absolute upper

bound for the integrity (integrity must be equal or lower). Using this algorithm, calculation of the integrity of a graph can be done in seconds, rather than over impossibly long periods of time.

This algorithm enables network engineers and researchers to analyze networks and graphs to determine their relative integrity to other networks, or even to test different configurations to see which configuration has higher integrity. Using these estimations, stronger networks can be built, increasing reliability, security, and usefulness of the networks. Additionally, this estimation may be useful in some graph theory applications for comparing graphs' integrity in some advanced research as the field progresses.

Through this research we have also established that determining the Maximal Coloring Sequence is comparable to determining the maximum independent set problem, firmly planting the problem within the scope of NP-Hard. This confirms there is currently no way to calculate the problem fully within reasonable time. Thus, estimation algorithms are the only option, so long as NP problems are not P.

8. Future Work and Conjecture

Though my algorithm, and my analysis of the graph integrity problem goes into uncharted territory, and provides a reasonable answer to the question of just how to determine integrity for any graph, there is more than can be done.

During my work, I determined that my algorithm always find the correct integrity for any bipartite graph (a ring, tree, or row). In this case there is only one way that my graph can possibly be colored maximally, no matter where it starts, but it does present an interesting question: is it possible to calculate the integrity of certain special classes of graphs faster or

more accurately through other algorithms or heuristics? I believe that by focusing on certain groups of graphs, such as degree-regular (all nodes have the same number of edges connected to them) graphs, special solutions or short-cuts that could streamline the calculation process can be devised.

Additionally, my algorithm seems to have problems with certain structures such as triangles, other completely connected sub-graphs, and web-graphs. The completely connected sub-graphs present a problem in that they will require N colors for the N nodes within them, and depending on whether I start coloring within the sub-graph or outside it, I will get different results for integrity. This leads me to believe that through some special consideration of these sub-graphs within the algorithm, a more accurate estimate might be possible. The web-graphs are the same problem, in that they are one node with lots connected to this one node, and not interconnected themselves. This is similar to a high-degree node within a lower-order graph. The coloring of the center node in this case is critical in determining the color distribution within the graph, which means that identification of these sub-graphs, and the coloring of them can significantly change the color sequence. Special focus on these might yield more accuracy, at the cost of speed.

Lastly, more statistical analysis of my algorithm could help confirm whether it maintains its accuracy as the complexity and size of the graph increases. This is future work by definition, in that a couple of orders higher of computing power will be necessary to achieve these studies. Through this analysis, a greater understanding of the inaccuracy of the algorithm and perhaps better applications of it will result.

References

[1] G. Chartrand and P. Zhang, *Introduction to Graph Theory* McGraw- Hill, New York, 2005

[2] C. A. Barefoot, R. Entringer, and H. C. Swart: Vulnerability in graphs- A comparative survey. J. Combin. Math. Combin. Comput. 1(1987) 12-22.