Ain't Nothin But a G-Trie

A Thesis
Presented to
The Division of Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Arthur James Lawson III

May 2022

Approved for the Division
(Computer Science)

James D. Fix

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class.

List of Abbreviations

You can always change the way your abbreviations are formatted. Play around with it yourself, use tables, or come to CUS if you'd like to change the way it looks. You can also completely remove this chapter if you have no need for a list of abbreviations. Here is an example of what this could look like:

AI	Artificial Intelligencet
CPU	Central Processing Unit
GFD	Graphlet Frequency Distribution

Table of Contents

List of Tables

List of Figures

Abstract

The preface pretty much says it all.

Dedication

You can have a dedication here if you wish.

Chapter 1

Introduction

Millions of people use social media every day. The amount of data available from this casual use is mind-numbingly large and can be used to serve a variety of important purposes. This includes training artificial intelligence (AI), making more personalized advertisements, and various forms of analysis of human social behavior. As social media becomes more prevalent, concerns around privacy are growing. In an ideal world, we can use this data in a way that can help us learn, grow, and improve. Before we do that, we take a look at how graphlet census helps identify the structure of a network.

Have you ever wondered how anonymous a network truly is? Many social network studies claim their data is safe and ethical because it has been anonymized, but what does anonymous mean in this context and how do we measure it? Is data safe because it replaces names with serial numbers? Is it possible to work backwards from an "anonymous" network and figure out what the original data set was? These are the questions that have guided my research over the past (–n–) months.

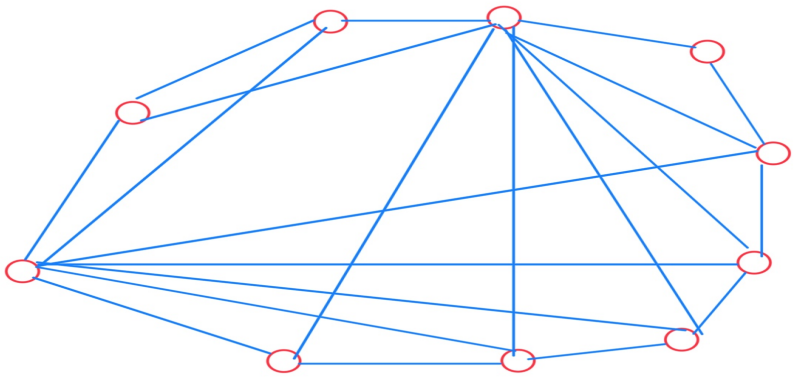
1.1 Social Networks

Let's take a look at Facebook. The core purpose is to help people connect with other people. The concept works because a user knows that when they create a profile, they can easily find and connect with their friends, family, and colleagues. A lot of information can be taken from even small friend networks.

For example, a friend network of 3 people could tell you a lot. In the case where there are 0 connections, it is pretty safe to assume that this social network won't be very successful. Each person that posts can only see their own posts and this site for sharing with others quickly becomes a diary that requires internet connection. If 2/3 of these people are friends, both of those two now have much more reason to use this application. Instead of talking to themselves, they are interacting with another person (being social!). Now suppose that the three people form a triangle. This means that they are all friends and become a lot more likely to post and interact because they know that they have two different people that could interact with their posts. They no longer log on to see only what person A had for breakfast, but they

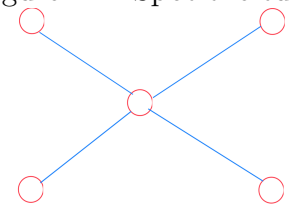
walk into a virtual world where person B has commented an unbelievable anecdote about eating that exact same cereal across from The Weeknd.

Figure 1.1: Can you pick the professor?



As networks increase in size, the possible shapes, and their implications, grow in complexity and potential value. For example, see Figure 1.1. In a social network of 10 Reed CS community members, there are a lot of questions to be asked. If I told you the student to professor ratio was 8:2, could you pick out who the two professors are without peeking at their long list of degrees? How about you look at each node's degree (–can turn this into a much more clever joke later –). After a look at the network and one would notice that two of the nodes have a lot more connections than the rest. It's a safe assumption that the two professors are the people with the high number of connections because students are very interested in their wisdom, humor, and pictures of Eitan flying his plane!

Figure 1.2: Spot the tutor?

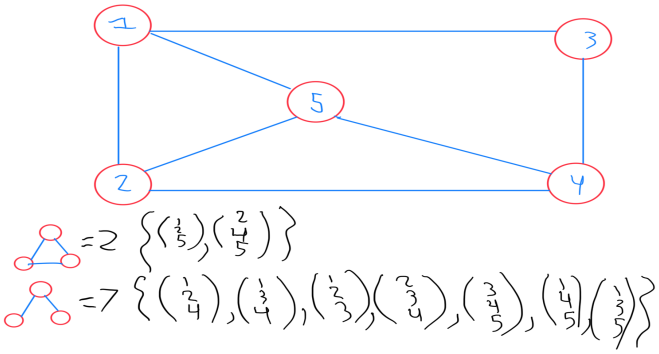


Similarly, what if I asked you to identify the 121 tutor in a group of 5 Reed students (Figure 1.2). Well, if you notice that there is only one person connected to every other, it would be a safe bet to say the person in the middle is the tutor. (With image, include definition of a star. OR can do so in later reference when we are defining shapes)

1.2 Subgraph Counting

Analysis of connections within a large network can be simplified with a bottom-up approach. Tens of millions of people in the US alone use Facebook on a daily basis. How can you analyze millions of people, and the billions of connections between them? You break the larger problem into a smaller problem –insert- parallel joke– . In a network of 5 people, you can focus on a smaller network of 3 people. The network of 3 people is a *subgraph* of the original network. We will provide a more formal definition for subgraph in the next chapter. The focus of this thesis is tackling the subgraph counting problem which can be defined as: given a specific collection of subgraphs and a network, how many times does each subgraph occur in the larger network?

Figure 1.3: Subgraph Counting

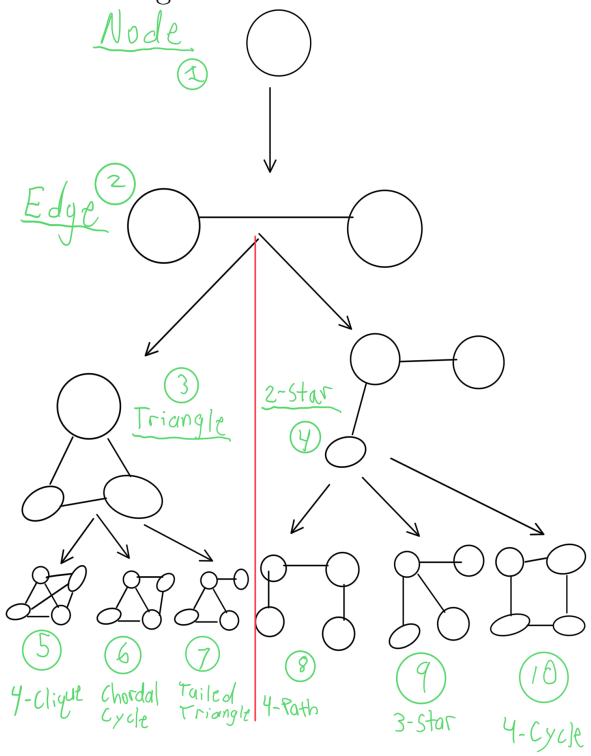


In Figure 1.3 we see an example of subgraph counting within a larger network. A single vertex can participate in different subgraphs as long as at least one of the others is different (such as node 1 participating in multiple subgraphs).

1.3 Motifs

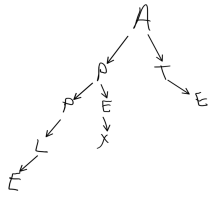
In any network, there are certain patterns that are bound to become recurring characters. *Network motifs* are reoccurring subgraphs with statistical significance. So far we've discussed triangles and stars (a graph term to represent shapes such as that of Figure 1.2 where one person is connected to everyone else). Let's take a look at some more specific shapes that can occur in a network.

Figure 1.4: Network Motifs



In Figure 1.4, there are 10 shapes that represent the possible combinations of 1, 2, 3, and 4 objects within a network. The primary work of graph tries (G-trie) is to store these shapes in a tree-like structure.

Figure 1.5: Trie Example



See Fig 1.5 for an example of a *prefix trie*. At each level, there is another letter added. If you go from the top letter and follow a path to any other letter, each step you take adds a letter to a potential word. Each letter would have a boolean value, *isWord* that tells you if the collection of letters is a valid word. Graph tries (G-tries) are very similar. Each step you take adds a vertex to a graph, and has an attribute *isGraph* that tells you whether or not it is a valid graph. For the purposes of my research, the motifs of size 3 and size 4 will have this value set to true. Large scale subgraph counting becomes much slower when computing subgraphs of size 4 or greater. Because of this, focusing on "easy" subgraphs of size 3 and harder graphs of size 4 has shown to be a balanced approach that is scalable to size n.

-Talk about value of shapes (and applications within real world networks)

1.4 Applications

random network comparison/testing?

comparing random models to real networks?

biology structures

In the field of biology, protein-protein interaction (PPI) networks are often represented as a graph. According to (Hayes 13)

(Cellular) "These networks are commonly modeled by graphs (also called networks) with nodes representing biomolecules such as genes, proteins, metabolites, etc., and edges representing physical, chemical, or functional interactions between the biomolecules. "

"A PPI network models the physical bindings between types of proteins in a cell, where a node represents a type of protein and an undirected edge exists between two nodes if the corresponding two proteins can physically bind to each other. The network of all such interactions in a species has been dubbed the *interactome*, and understanding its structure is an integral step towards understanding cellular systems."

From Li 2021

A popular source of analysis in biological networks is comparing diseased (or cancerous) cell networks with those of healthy cells. Finding the difference between these networks can be helpful in drug development. As researchers discover the irregular patterns, they can be targeted for treatment and to prevent further spread.

Another use is to compare networks across species which would allow for insights into their evolutionary progress.

comparing a diseased cellular network to a healthy one may aid in finding a cure for the disease, and comparing cellular networks of different species could enable evolutionary insights.

anonymous networks

Discussion on status of isomorphism problem and graph comparison

comparing structures

1.5 Algorithms

Algorithm 1, for sequential algorithm

Algorithm 2, for parallel Algorithm 3, for symmetry breaking conditions

Chapter 2

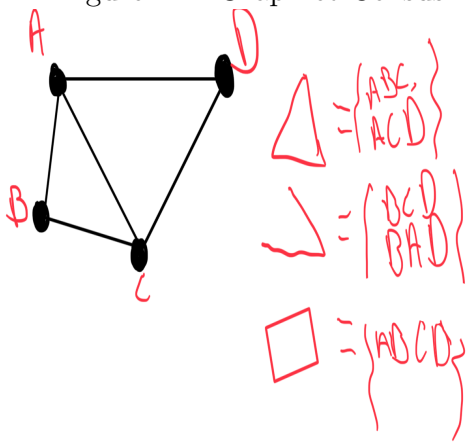
Preliminaries (II)

2.1 Graphlet Census

What is Graphlet Census? Graphlet Census can be defined as the collection of motifs within a graph that we value. In the literature, **Graphlet Degree Distribution** is the table used to keep track of these motifs. Each time a new version of motif m is found, GDD[m]is incremented by one. At the end of the program, the counts of all motifs serves as our fingerprint for the network.

Math definition:

Figure 2.1: Graphlet Census



Graphlet Degree Distribution		
Motif	Count	Occurrences
Triangle	2	{ABC}, {ACD}
Two Star	2	{BCD}, {BAD}
Square	1	{ABCD}

Table 2.1: Simple Census Table

2.1.1 Example

In Figure 2.1, we see an example of this. Say we are doing a census of triangles, two stars, and squares. Table 2.1 shows the GDD for fig. 2.1. It is important to note, that one motif may occur within another of the same degree. Every triangle includes a two star, but it isn't counted as a two star because that collection of vertices has the additional edge a two star is missing.

2.1.2 Naive Approach

A naive algorithm for graphlet census would do an individual search for each possible shape. Initial trials started with motifs of degree 3.

This approach has a number of issues. The primary one being that every motif is found 3 times.

Chapter 3

Graphlet Census With G-Tries (III)

G-tries are the magic tool we'll be using to advance our sequential hopes.

3.1 Definition

A *graphlet-trie (G-trie)* can be defined as a structure used to store graphlets in a tree-like structure. Recall figure 1.4 where we showed the structure of a prefix trie. This is the form used for a G-trie. At each level, *depth* increases by one, and another node is added. Each motif's new node has a different relationship with the previously established vertices.

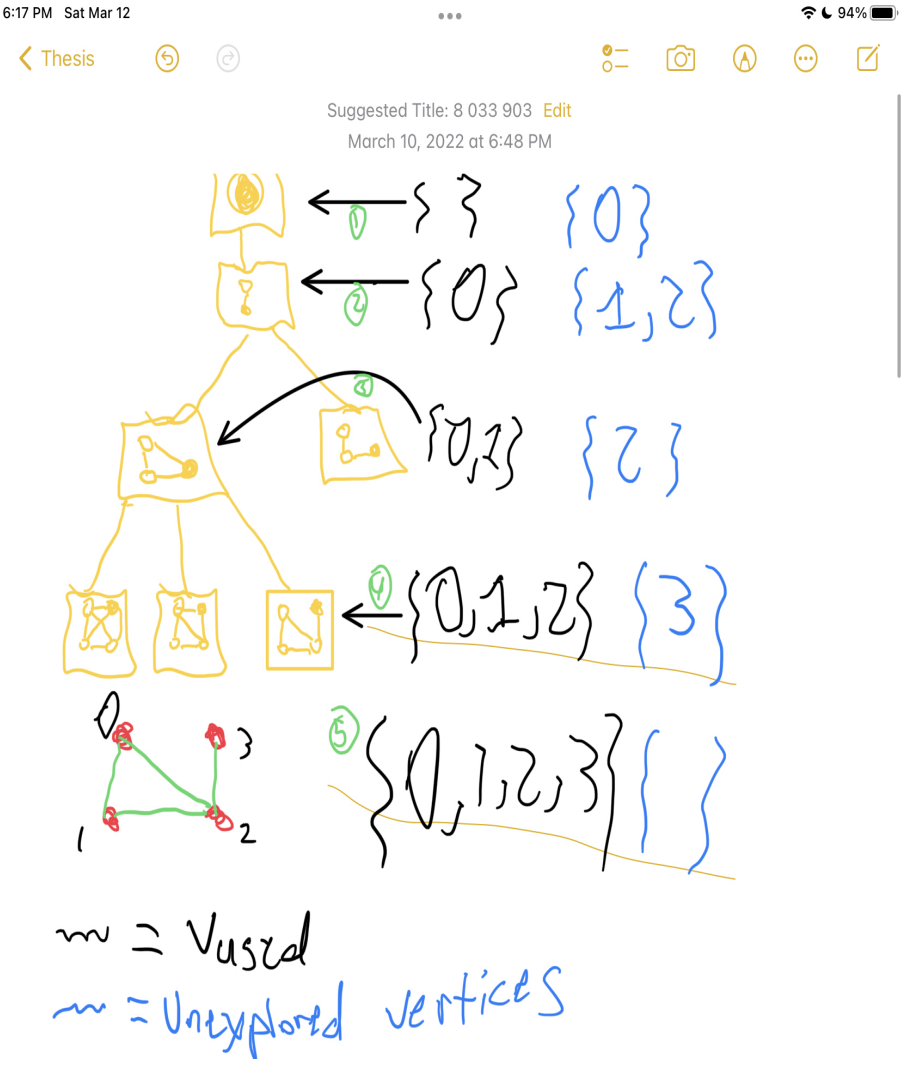
In our implementation, each graphlet holds a specific amount of information. The most important being $isGraph$ and adj. $IsGraph$ is a boolean used to determine if a graphlet is one we are interested in counting. When false, we know the current graphlet is only a middle man between us and higher depth graphlets of interest. In the prefix trie, "ap" is not considered a valid english word, but "app" is. Adj is an adjacency matrix that serves to check if the newest vertex has the necessary connections to the vertices already in V_{used}.

3.2 Examples

3.2.1 An example walk through

In figure 3.1, We start with a walk through example, followed by a larger one in figures 3.2, 3.3 and 3.4.

Figure 3.1: Sequential Walk



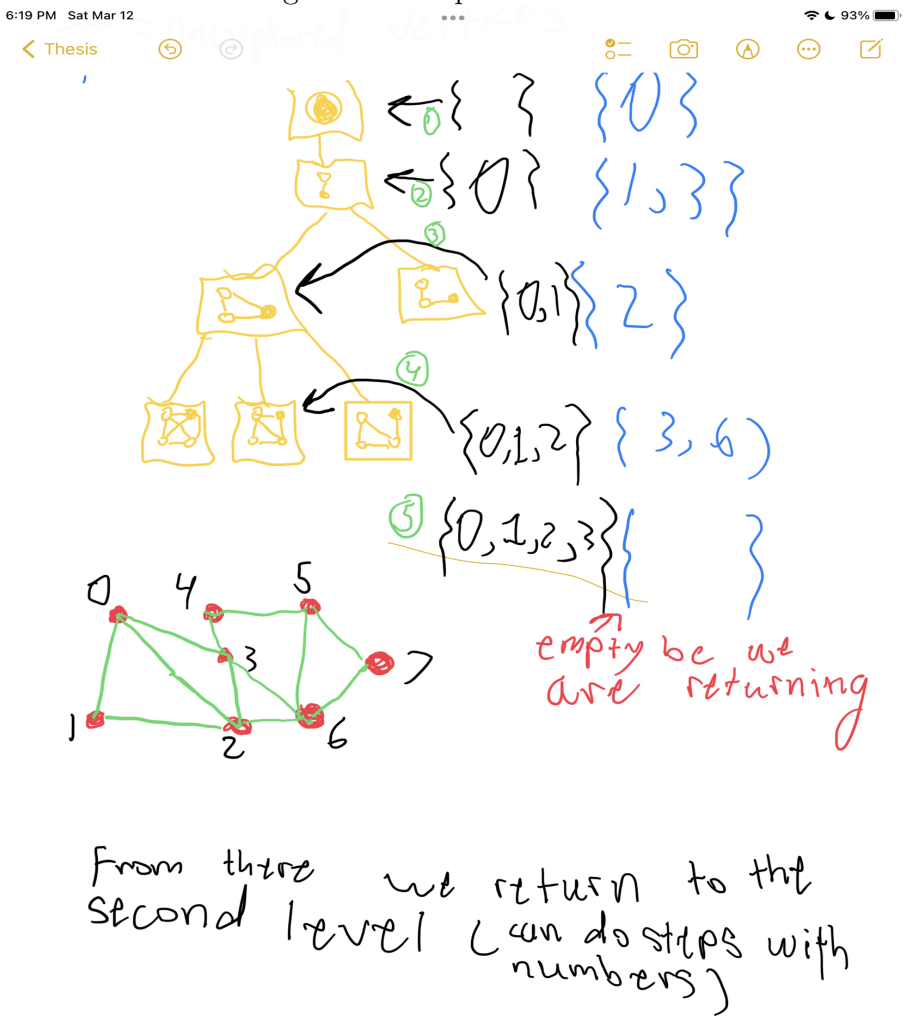
At step {1} we have nothing in V_{used} and 0 as our unexplored vertex. Note, in this chapter we will only keep track of the matching vertices at the current depth. At step two, we see that {0} has two neighbors, $\{1, 2\}$ and 1 is chosen. At the end of steps {3} and {4}, we have two nodes that are identified as a graphlets of interest. Because of this, steps {4} and {5} output them (as line x in Algorithm x).

3.2.2 A longer walk through

Figures 3.2, 3.3 and 3.4 will show us a more complex example.

Steps {1} through {5} of figure 3.2 follow the same as figure 3.1. At the conclusion of step {5}, we have reached the maximum depth for graphlets of interest. Now we stepped back up one level, saw there was an unexplored path, so we removed the one we explored already and pursue the new one. Note that figure 3.3 shows the current status of us returning to level {4} and moving past the previously explored path at the current depth.

Figure 3.2: Sequential Walk 2



In figure 3.4, we continue through the rest of its neighbors that weren't selected. This gives us a new output of $\{0, 1, 2, 6\}$. From there we go even further back up the stack to process through potential paths in the same fashion as a classic Depth First Search (DFS) (include info/reference at the end for DFS and have a footnote pointing to it).

It also warrants mentioning that symmetry breaking conditions are important here. There are different ways to come across the same graphlet. When we return up the stack, there will be different routes to the same path (such as the triangle $\{0, 1, 2\}$ also being found as $\{0, 2, 1\}, \{2, 0, 1\}$, and so on). Our algorithm handles this by using C++ *set* objects to maintain no duplicates being stored in the GDD counter. We will talk in Chapter about what an optimized setup would look like with symmetry breaking conditions.

Figure 3.3: Sequential Walk 3

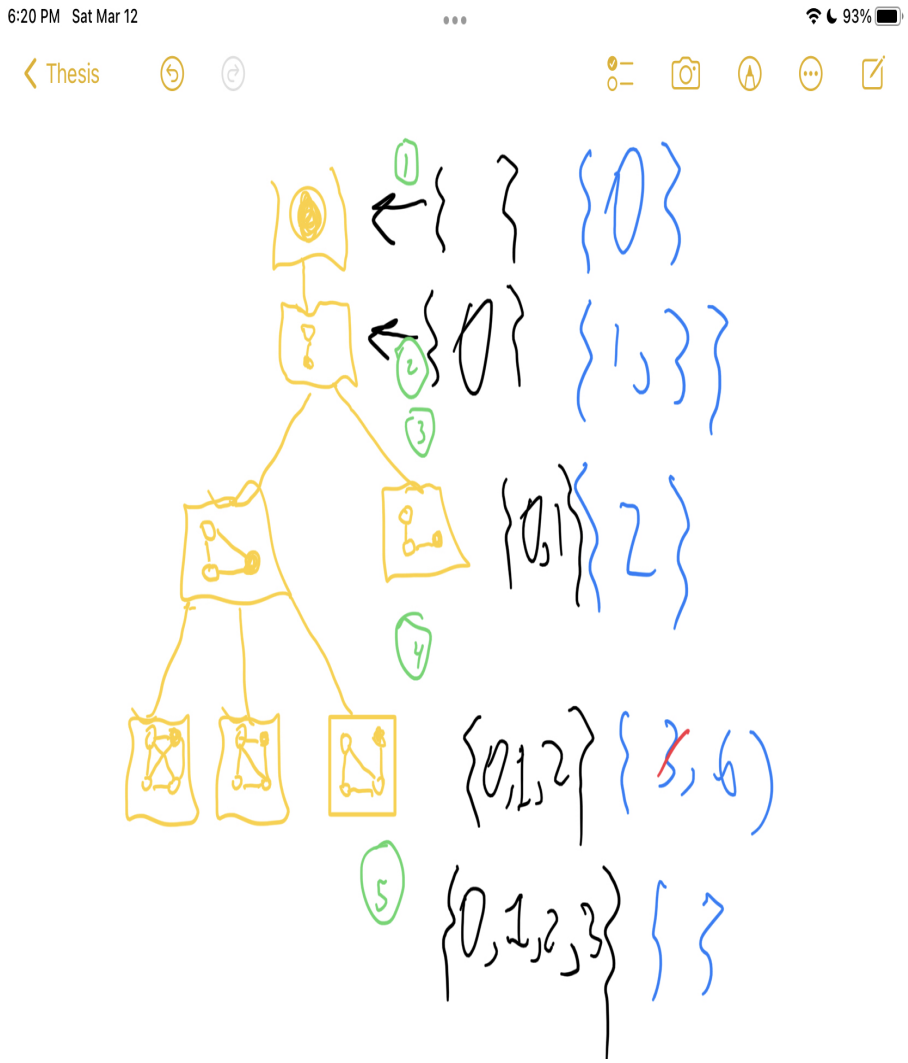
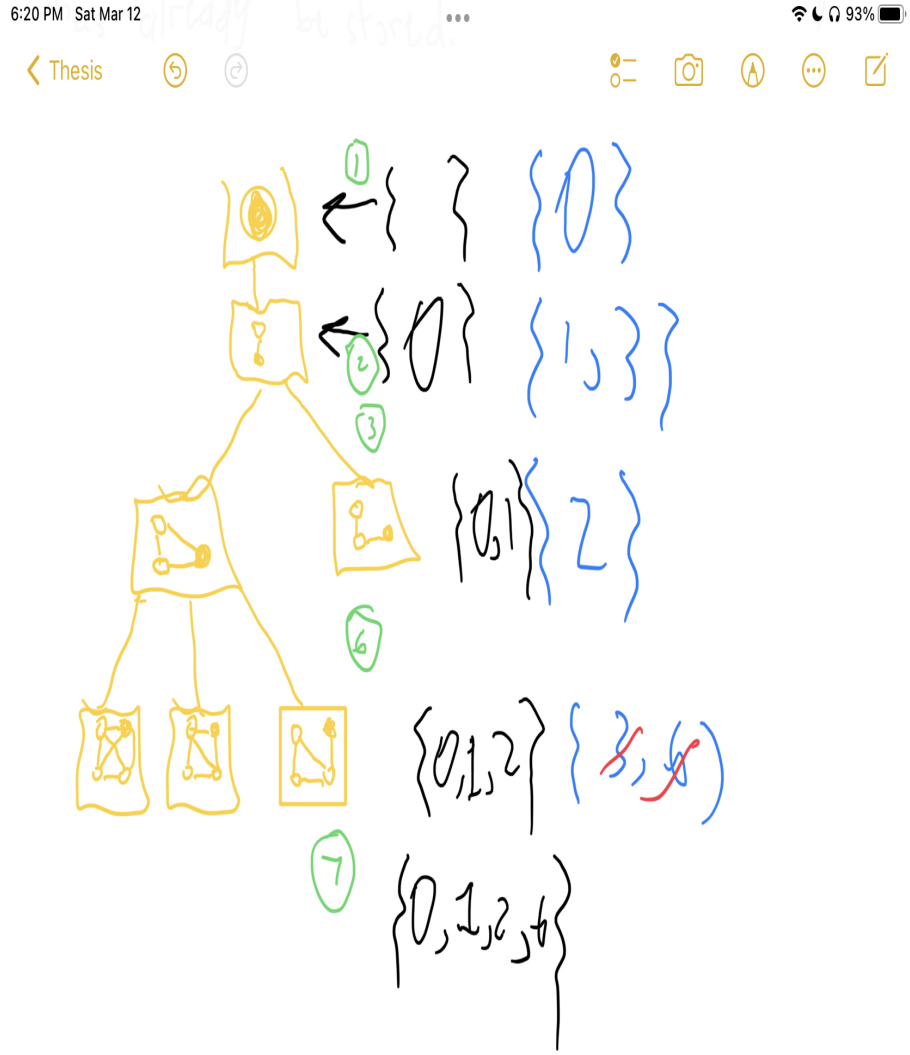


Figure 3.4: Sequential Walk 4



Chapter 4

Parallel Census G-Tries

Following the completion of the sequential algorithm, we set our sights on a parallel solution. We worked on two models, one with work sharing and one without. *Work sharing* is defined as simply as it is named: when one processor is done with it's work, it asks for more from one of the processors still grinding away. Say you and a friend are working on 100 Christmas cards split 50-50. When you are done, your friend has 10 cards remaining so you split the last 10 5-5. That way, you are not idling and the overall task gets done faster. This creates a need for efficient communication and sharing.

4.1 Without Work Sharing

To create a parallel algorithm from the sequential one, we start by looking at the functionality within $MatchingVertices$, more specifically, within the candidate creation. On line xxx, we see that when V_{used} is empty, all vertices within the graph are listed as candidates. For our parallel version, this initial allocation is divided up by the number of processor **round-robin style**, similar to what was done in **par˙t**. So if there are 100 vertices and 10 processors, processor 1 will get vertices 1,11,21... processor 2 will get 2,12,22... and so on. This follows the the distribution choice made in **par˙t**. This division means that each processor will have its own set of starting vertices as it looks for the different motifs of interest. More on this in the next section.

While the parallel approach offers good speed-ups over the sequential algorithm, there are certainly areas to improve on. For example, each processor is assigned an initial pool of start vertices. Every vertex in a graph can have between 0 and —V-1— neighbors. This quickly leads to some processors having much more work than others. If processor 2 has 5x as much work as processor 3, that means processor 3 will be idling at the finish line while processor 2 continues to chug away. Because of this, it makes sense to work towards a solution where a processor can ask for more work once it includes its initial allocation.

4.2 Work Sharing

The next step in working towards an efficient parallel algorithm is to implement *work sharing*. For this to function properly, we needed to implement a way for processors to communicate their needs, share work, and resume work after this sharing takes place.

4.2.1 Functions and Data Structures

An important preliminary to understand is the concept of a *WorkUnit*. In our implementation, the work unit keeps track of V_{used}, UnexploredVertices, and ExploredNodes. V_{used} is the same as the V_{used} from the sequential algorithm, but now there is one per processor. UnexploredVertices is the storage of the vertices from line xxx of the sequential algorithm (V = MatchingVertices). This is used to store the paths for continued exploration once work is resumed. ExploredNodes does something similar, but for the remaining Nodes to be explored. UnexploredVertices is stored as a map, mapping the depth to the vertices being explored at that depth. ExploredNodes is stored as a list because it does not need to keep track of depth. When needed, it stores only the nodes at the current depth.

Each processor is assigned a work unit that keeps track of its current workload, the motif matches it has found thus far, and most importantly, communicate when it needs more work. We implement the following functions: $AskForWork$, $DivideWork$, $ResumeWork$, and $CheckMessages$. We also used a $GlobalWorkQueue$ to manage the communication of need and sharing of work.

Once a processor has completed its initial workload, it calls $AskForWork$ to signal that it needs more work. In our implementation, this sends a pointer to its workload to the global queue. The other processors are periodically checking if the queue is empty. If there is something in the queue, then a processor pulls in that workUnit, splits it's current workload using DivideWork, and signals to the unemployed processor that it can continue working.

$DivideWork$ is perhaps the most important function to our work sharing model. When a processor asks for work ($P_{receiver}$), it has exhausted its search pool, meaning there is no remaining V_{used}, currentNode, or unexploredVertices. This means that the processor sharing its work (P_{sender}), needs to fulfill all of these variables in addition to giving it exploredNodes so it knows where to resume its search. UnexploredVertices is what is defined as the actual work to be done. **par˙t** made the decision to split matching vertices even-odd at each depth. P_{sender} would keep the odd numbered vertices and $P_{receiver}$ would take the even. There was no clear performance gain from this decision. They mentioned the benefit of having an odd-even split (which follows intuitively, you would want each processor to have approximately the same amount of work). For simplicity, I decided to divide the vertices at each depth by splitting each list in half and shipping the 2nd half to P_{sender}. This maintains the same amount of vertices being shared. The rest, V_{used}, exploredNodes, and currentNode are copied directly to the requesting processor so it has a foundation to build upon. ExploredNodes is used to store the remaining children of T when a work request

causes a stop mid-search. This is useful in saving state from both parallel match and resumeWork.

Once the work has been split, it is important that there is a clear mechanism for both processors to resume their respective searches. This is where explored nodes comes in handy. When P_{sender}, receives the call for a workRequest, it must stop its current search and store its current state. When resumeWork is called, P_{sender} needs to pick back up from where it left off. Similar to recursive calls of ParallelMatch returning to the top of the stack, resumeWork needs to clear that level of the stack by making calls to ParallelMatch can talk with Jim about a prettier way to describe the return to the top of the stack. It does this by cycling/working through the exploredNodes and calling ParallelMatch there on its current version of V_{used} (finishing the loop that was interrupted when P_{sender} stopped to send work), before continuing through its load of unexploredVertices. Each of these loops has a check for new work requests. When one is received, the current state is stored before retreating back to the larger loop to split and send work. Each level up stores the state data at it's respective depth. (**par˙t**) was ambiguous about what it meant by saving state in these instances. After thinking it through, it makes sense that the first loop in resumeWork would store the up-to-date explored nodes information (removing those that had already been checked) and maintain the current state of unexploredVertices. The secondary loop would clean up the unexplored vertices we have at each depth, while adding the remaining children of T to explored nodes. The difference between receiving a work request during resumeWork and parallelMatch is that resumeWork modifies the state to remove what it has already processed. ParallelMatch must populate the workUnit object itself.

4.3 Thread-Safety

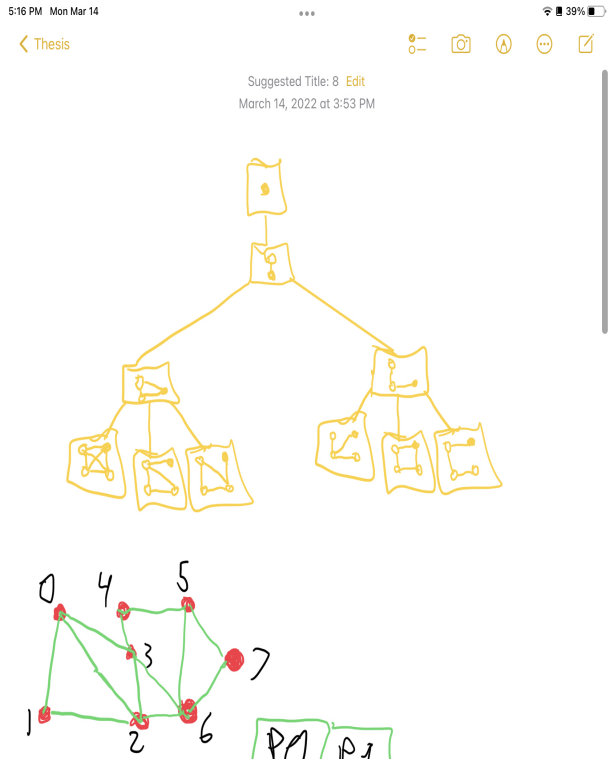
Another important aspect of the work-sharing model is proper thread safety. The original parallel algorithm had to safeguard against multiple writes to a global GDD. This means that two processors attempted to put new groups of V_{used} into the GDD at the same time. Competing writes leads to undefined (and very problematic) behavior without proper protection. Initially, we fixed this by creating a thread-safe GDD map, locking access whenever one processor was actively writing to it. With frequent writes from all processors, this locking, unlocking, and associated overhead, caused large slow downs (can talk with Jim about whether it would be useful to include a table or numbers to show this). We solved this problem by diverging from the sequential code and giving each processor it's own GDD to write to. Once all calculations have been performed, these are all aggregated into one global GDD. This offered great speedups that put us a lot more in-line with the hope of linear speedups as P increases.

Once we got to the work-sharing model, thread safety became even more important. In addition to safe GDD, we need to protect the global queue. Similar to our first approach for GDD, creating a custom class with a mutex to protect calls to *enqueue and dequeue* otherwise known as *push and pop* This prevent multiple processors from trying to fill the same request and helps avoid the situation of a processor

stopping to split work only to error because the queue became empty as it prepared to share.

4.4 A More Complex Example

Figure 4.1: Parallel Walk Graph



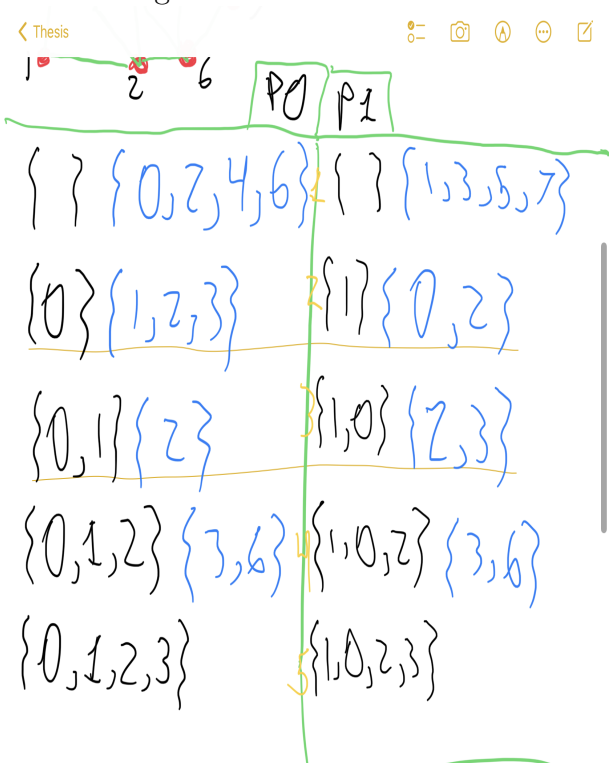
Here we'll show a step-by-step example of how work-sharing works. We will reuse the same graph from figure 3.2, shown in figure 4.1.

Steps 1-5 in figure 4.2 maintain what we showed previously (reference sequential OR parallel code, depends on if we showed a parallel w/o ws example). At step {6} we have made it through the first run with $\{0, 1, 2, 3\}$ and $\{1, 0, 2, 3\}$ being outputted (hello symmetry breaking conditions!) From here, both processors go back one depth, to include 6 instead of 3.

Step {7} in figure 4.4 is where things start to get interesting. From here, we have fast-forwarded to when P_0 has made it through all of it's work (V_{used} and unexploredVertices are both empty) so it asks for more[1]. Off screen, P_1 or P_{sender}, stops its search, stores its current state, and shares with $P_{receiver}$. Figure 4.5 has step {8} showing the aftermath of this transaction. V_{used} has been copied, and unexploredVertices has been split between the two processors. It may be trivial, but necessary

1. It is important to note, that there are other reasons why one processor would finish faster than another even with similar workloads. One being that each core in a CPU is usually made for performance (more powerful) or efficiency (preserve battery life), especially on consumer machines. We will learn more about this in the results chapter.

Figure 4.2: Parallel Walk 2



to say for correctness, that the split accommodates when UnexploredVertices is of an odd length. Step {9} shows both processors resuming their workload from this point.

Figure 4.3: Parallel Walk Current Status

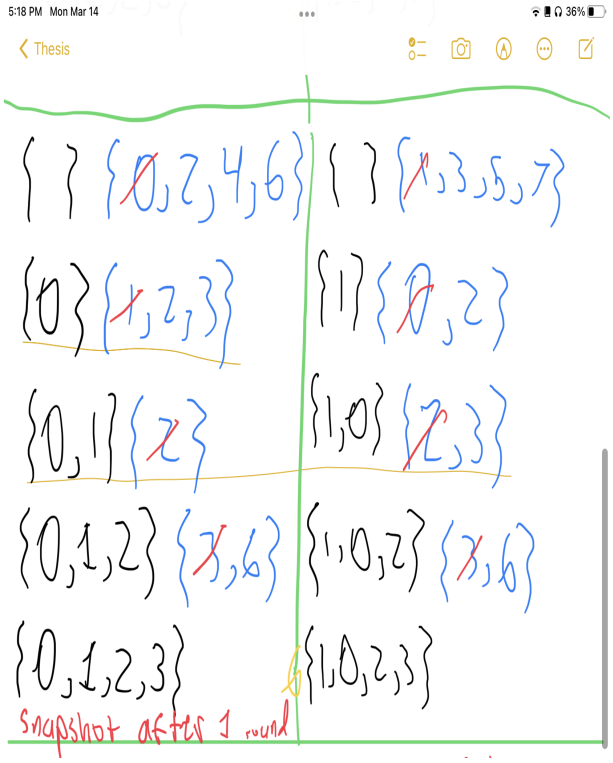


Figure 4.4: Parallel Walk 4

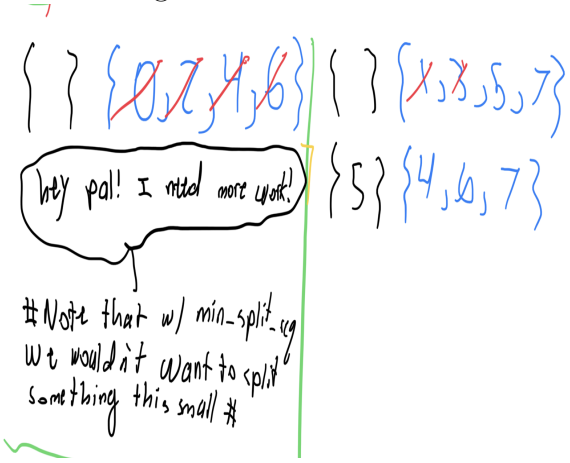
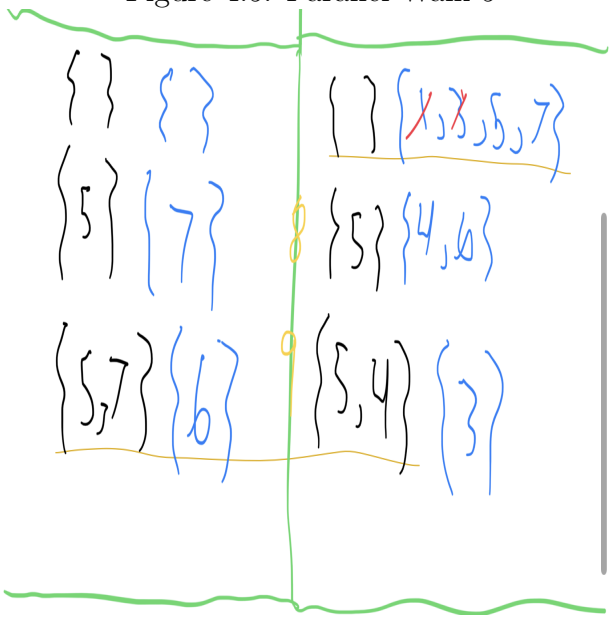


Figure 4.5: Parallel Walk 5



Chapter 5

Results and Discussion

5.1 Optimizations

5.2 Furthering this work

Chapter 6

Preliminaries old

6.1 Useful Definitions

Graph We can define a graph as an abstract data type representing the connections between objects. This is often represented as a collection of edges and vertices. All graphs are either *undirected* or *directed*. Twitter followers are a directed graph. Just because I follow LeBron James, doesn't mean he follows me back. Undirected graphs are like Facebook friends, I can only be friends with Sarah if and only if Sarah is friends with me. We will be working primarily with undirected graphs. See Figure

Node clarify g-trie node vs graph node

Edge An edge represents a connection between two nodes.

canonical form:: "the simplest representation of an object and which allows it to be identified in a unique way." every isomorphic graph will have the same canonical form

Graphlets:: Small induced non-isomorphic subgraphs

Parallel Read/Write::

Degree:: The total number of edges connected to a single node. A node with n edges connected to it has a degree of n

CPU:: Every computer, phone, smartwatch, etc. works because it follows a set of instructions. More specifically, all of these devices have a Central Processing Unit (CPU) that retrieves and executes instructions. For decades, CPU's would get substantially better year over year to the point where the same code would run significantly faster just from running it on the latest machine. In the late 90's / early 2000's, people began to see the diminishing returns of this. –Free lunch is over–. As a result, more research went into building parallel processing and sufficient architecture to run it on. CPU's typically have 2-8 cores. For simple tasks, such as web browsing, word documents, and streaming videos, the CPU primarily uses 1 core. With more intensive tasks, the rest of the cores are called in.

Core:: The hardware component on a chip that handles instructions

canonical form:: graph canonization is the problem finding a canonical form of a given graph G. A canonical form is a labeled graph Canon(G) that is isomorphic to G, such that every graph that is isomorphic to G has the same canonical form as G.

Thus, from a solution to the graph canonization problem, one could also solve the problem of graph isomorphism: to test whether two graphs G and H are isomorphic, compute their canonical forms Canon(G) and Canon(H), and test whether these two canonical forms are identical.

Gtrie:: (will want to talk about struggles with larger scale, and transitioning to molding graph then using g-trie instead (a little more overhead for BIG improvements

Thread:: The software component that manages the tasks/instructions For example, lets say you have 50 addition problems to complete. You're smart, so it takes you at most, a minute to complete these problems. Imagine you have a friend that could also complete these problems in at most a minute. If you split the problems 25-25, it's reasonable to assume that each of you could complete your 25 problems in 30 seconds. The same logic applies to sequential vs parallel programing. Each thread –define thread vs core– can do a task in n time. Most parallel programming aims to get linear speedups, meaning, 1 proccessor takes n time, 2 processors takes n/2 time, 4 processors takes n/4 time and so on. The initial goal for this assignment was to get an algorithm to do a similar 'divide and conquor

(section on graph representation) Adjacency matrix:: (Graph representation [show with example of drawn graph]) Another representation of a graph. A —V— x —V— matrix that shows all possible connections in a graph.

math assignment 1 if (u,v) in edges, 0 otherwise from Jiarong's thesis

Chapter 7

The Methods

My ambition was to look at previous work with graphlet census and further it by parallelizing the process.

7.1 First Try

Initially, I studied sequential algorithms for graphlet census. Primarily I used what I learned from –inseert source– to inform a sequential algorithm for counting the number of each desired shape in a census. As expected, this worked well for 3-node census. maybe include pseudocode? Once we got the 4-node census things were much slower detail time for triangles vs 4-clique.

From there, I transitioned to a parallel model that was centered around dividing the graph into n/p sections that each perform the sequential algorithm on a n/p size chunk of the graph. This performed very well for 3-node census, but was buggy during 4-node census. I also realized that, even fully implemented, it wouldn't be good enough.

7.2 G-Trie

Further research pushed me towards investigating the use of g-tries. This approach has some overhead, but was shown to perform much faster than state of the art sequential search.

7.2.1 Building G-Trie

As previously mentioned, a g-trie is a structure. This means it must be built for it to have any use. –Algorithm – shows the building of a g-trie. Talk further about what a g-trie is This algorithm allows us to build and place the shapes within the structure that we need. Each level to the trie also has attributes to describe its place in the overall structure–?? such as –isGraph –

7.2.2 Canonical Form

Before we work with a g-trie, the initial graph is put into canonical form. This form maintains the structure of the network while making it easier to search through (less repeats and ..). The canonized version of the graph is isomorphic to the original graph G. This means that any other isomorphic graph to G will have the same canonized form as G. From here, the canonized version

Chapter 8

More need to know

Talk about graphs and and get into what the foundational problem is.

8.1 What is a Graph?

8.2 Parallel Programming

Suppose you have 50 addition problems to complete. You're smart, so it takes you at most, a minute to complete these problems. Imagine you have a friend that could also complete these problems in at most a minute. If you split the problems 25-25, it's reasonable to assume that each of you could complete your 25 problems in 30 seconds. This thinking is the core concept of parallel programming. A lot of problems that rely on going through a long list of calculations can be divided up into different sections and ideally this would scale in a linear fashion be much faster.

Chapter 9

The First

This is the first page of the first chapter. You may delete the contents of this chapter so you can add your own text; it's just here to show you some examples.

9.1 References, Labels, Custom Commands and Footnotes

It is easy to refer to anything within your document using the `label` and `ref` tags. Labels must be unique and shouldn't use any odd characters; generally sticking to letters and numbers (no spaces) should be fine. Put the label on whatever you want to refer to, and put the reference where you want the reference. LaTeX will keep track of the chapter, section, and figure or table numbers for you.

9.1.1 References and Labels

Sometimes you'd like to refer to a table or figure, e.g. you can see in Figure 11.2 that you can rotate figures . Start by labeling your figure or table with the label command (`\label{labelvariable}`) below the caption (see the chapter on graphics and tables for examples). Then when you would like to refer to the table or figure, use the ref command (`\ref{labelvariable}`). Make sure your label variables are unique; you can't have two elements named "default." Also, since the reference command only puts the figure or table number, you will have to put "Table" or "Figure" as appropriate, as seen in the following examples:

As I showed in Table 11.1 many factors can be assumed to follow from inheritance. Also see the Figure 11.1 for an illustration.

9.1.2 Custom Commands

Are you sick of writing the same complex equation or phrase over and over?

The custom commands should be placed in the preamble, or at least prior to the first usage of the command. The structure of the `\newcommand` consists of the name of the new command in curly braces, the number of arguments to be made in square

brackets and then, inside a new set of curly braces, the command(s) that make up the new command. The whole thing is sandwiched inside a larger set of curly braces.

In other words, if you want to make a shorthand for H_2SO_4, which doesn't include an argument, you would write: `\newcommand{\hydro}{H$_2$SO$_4$}` and then when you needed to use the command you would type `\hydro`. (sans verb and the equals sign brackets, if you're looking at the .tex version). For example: H_2SO_4

9.1.3 Footnotes and Endnotes

You might want to footnote something.[1] Be sure to leave no spaces between the word immediately preceding the footnote command and the command itself. The footnote will be in a smaller font and placed appropriately. Endnotes work in much the same way. More information can be found about both on the CUS site.

9.2 Bibliographies

Of course you will need to cite things, and you will probably accumulate an armful of sources. This is why BibTeX was created. For more information about BibTeX and bibliographies, see our CUS site (web.reed.edu/cis/help/latex/index.html)[2]. There are three pages on this topic: *bibtex* (which talks about using BibTeX, at /latex/bibtex.html), *bibtexstyles* (about how to find and use the bibliography style that best suits your needs, at /latex/bibtexstyles.html) and *bibman* (which covers how to make and maintain a bibliography by hand, without BibTeX, at at /latex/bibman.html). The last page will not be useful unless you have only a few sources. There used to be APA stuff here, but we don't need it since I've fixed this with my apa-good natbib style file.

9.2.1 Tips for Bibliographies

1. Like with thesis formatting, the sooner you start compiling your bibliography for something as large as thesis, the better. Typing in source after source is mind-numbing enough; do you really want to do it for hours on end in late April? Think of it as procrastination.

2. The cite key (a citation's label) needs to be unique from the other entries.

3. When you have more than one author or editor, you need to separate each author's name by the word "and" e.g.
   `Author = {Noble, Sam and Youngberg, Jessica},`.

4. Bibliographies made using BibTeX (whether manually or using a manager) accept LaTeX markup, so you can italicize and add symbols as necessary.

1. footnote text
2. **reedweb:2007**

5. To force capitalization in an article title or where all lowercase is generally used, bracket the capital letter in curly braces.

6. You can add a Reed Thesis citation[3] option. The best way to do this is to use the phdthesis type of citation, and use the optional "type" field to enter "Reed thesis" or "Undergraduate thesis". Here's a test of Chicago, showing the second cite in a row[4] being different. Also the second time not in a row[5] should be different. Of course in other styles they'll all look the same.

9.3 Anything else?

If you'd like to see examples of other things in this template, please contact CUS (email cus@reed.edu) with your suggestions. We love to see people using LaTeX for their theses, and are happy to help.

3. **noble:2002**
4. **noble:2002**
5. **reedweb:2007**

Chapter 10

Mathematics and Science

10.1 Math

TEX is the best way to typeset mathematics. Donald Knuth designed TEX when he got frustrated at how long it was taking the typesetters to finish his book, which contained a lot of mathematics.

If you are doing a thesis that will involve lots of math, you will want to read the following section which has been commented out. If you're not going to use math, skip over this next big red section. (It's red in the .tex file but does not show up in the .pdf.)

MATH and PHYSICS majors: Uncomment the following section

$$\sum_{j=1}^{n}(\delta\theta_j)^2 \leq \frac{\beta_i^2}{\delta_i^2+\rho_i^2}\left[2\rho_i^2+\frac{\delta_i^2\beta_i^2}{\delta_i^2+\rho_i^2}\right] \equiv \omega_i^2$$

From Informational Dynamics, we have the following (Dave Braden):

After n such encounters the posterior density for θ is

$$\pi(\theta|X_1 < y_1, \ldots, X_n < y_n) \propto \pi(\theta)\prod_{i=1}^{n}\int_{-\infty}^{y_i}\exp\left(-\frac{(x-\theta)^2}{2\sigma^2}\right)\,dx$$

$$\det\begin{vmatrix} c_0 & c_1 & c_2 & \ldots & c_n \\ c_1 & c_2 & c_3 & \ldots & c_{n+1} \\ c_2 & c_3 & c_4 & \ldots & c_{n+2} \\ \vdots & \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & c_{n+2} & \ldots & c_{2n} \end{vmatrix} > 0$$

Lapidus and Pindar, Numerical Solution of Partial Differential Equations in Science and Engineering. Page 54

$$\int_t\left\{\sum_{j=1}^{3}T_j\left(\frac{d\phi_j}{dt}+k\phi_j\right)-kT_e\right\}w_i(t)\,dt=0, \qquad i=1,2,3.$$

L&P Galerkin method weighting functions. Page 55

$$\sum_{j=1}^{3} T_j \int_0^1 \left\{ \frac{d\phi_j}{dt} + k\phi_j \right\} \phi_i \; dt = \int_0^1 k\, T_e \phi_i dt, \qquad i = 1, 2, 3$$

Another L&P (p145)

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f\big(\xi,\eta,\zeta\big) = \sum_{k=1}^{n}\sum_{j=1}^{n}\sum_{i=1}^{n} w_i w_j w_k f\big(\xi,\eta,\zeta\big).$$

Another L&P (p126)

$$\int_{A_e} (\,\cdot\,) dx dy = \int_{-1}^{1}\int_{-1}^{1} (\,\cdot\,) \det[J] d\xi d\eta.$$

10.2 Chemistry 101: Symbols

Chemical formulas will look best if they are not italicized. Get around math mode's automatic italicizing by using the argument `$\mathrm{formula here}$`, with your formula inside the curly brackets.

So, $Fe_2^{2+}Cr_2O_4$ is written `$\mathrm{Fe_2^{2+}Cr_2O_4}$`
Exponent or Superscript: O^-
Subscript: CH_4

To stack numbers or letters as in Fe_2^{2+}, the subscript is defined first, and then the superscript is defined.
Angstrom: Å
Bullet: $CuCl \bullet 7H_2O$
Double Dagger: ‡
Delta: Δ
Reaction Arrows: $\longrightarrow$ or $\xrightarrow{solution}$
Resonance Arrows: $\leftrightarrow$
Reversible Reaction Arrows: $\rightleftharpoons$ or $\xrightleftharpoons{solution}$ (the latter requires the chemarr package)

10.2.1 Typesetting reactions

You may wish to put your reaction in a figure environment, which means that LaTeX will place the reaction where it fits and you can have a figure legend if desired:

$$C_6H_{12}O_6 + 6O_2 \longrightarrow 6CO_2 + 6H_2O$$

Figure 10.1: Combustion of glucose

10.2.2 Other examples of reactions

$NH_4Cl_{(s)} \rightleftharpoons NH_{3(g)} + HCl_{(g)}$

$MeCH_2Br + Mg \xrightarrow[below]{above} MeCH_2 \bullet Mg \bullet Br$

10.3 Physics

Many of the symbols you will need can be found on the math page (http://web.reed.edu/cis/help/latex/math.html) and the Comprehensive LaTeX Symbol Guide (enclosed in this template download). You may wish to create custom commands for commonly used symbols, phrases or equations, as described in Chapter 9.1.2.

10.4 Biology

You will probably find the resources at http://www.lecb.ncifcrf.gov/~toms/latex.html helpful, particularly the links to bsts for various journals. You may also be interested in TeXShade for nucleotide typesetting (http://homepages.uni-tuebingen.de/beitz/txe.html). Be sure to read the proceeding chapter on graphics and tables, and remember that the thesis template has versions of Ecology and Science bsts which support webpage citation formats.

Chapter 11

Tables and Graphics

11.1 Tables

The following section contains examples of tables, most of which have been commented out for brevity. (They will show up in the .tex document in red, but not at all in the .pdf). For more help in constructing a table (or anything else in this document), please see the LaTeX pages on the CUS site.

Table 11.1: Correlation of Inheritance Factors between Parents and Child

Factors	Correlation between Parents & Child	Inherited
Education	-0.49	Yes
Socio-Economic Status	0.28	Slight
Income	0.08	No
Family Size	0.19	Slight
Occupational Prestige	0.21	Slight

If you want to make a table that is longer than a page, you will want to use the longtable environment. Uncomment the table below to see an example, or see our online documentation.

Table 11.2: Chromium Hexacarbonyl Data Collected in 1998–1999

Chromium Hexacarbonyl			
State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	266 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	0.57
$y^7P_3^\circ$	266 nm	Argon	1
$y^7P_3^\circ$	355 nm	Argon	0.14
$y^7P_2^\circ$	355 nm	Argon	0.14
$z^5P_3^\circ$	266 nm	Argon	1.2
$z^5P_3^\circ$	355 nm	Argon	0.04
$z^5P_3^\circ$	355 nm	Helium	0.02
$z^5P_2^\circ$	355 nm	Argon	0.07
$z^5P_1^\circ$	355 nm	Argon	0.05
$y^5P_3^\circ$	355 nm	Argon	0.05, 0.4
$y^5P_3^\circ$	355 nm	Helium	0.25
$z^5F_4^\circ$	266 nm	Argon	1.4
$z^5F_4^\circ$	355 nm	Argon	0.29
$z^5F_4^\circ$	355 nm	Helium	1.02
$z^5D_4^\circ$	355 nm	Argon	0.3
$z^5D_4^\circ$	355 nm	Helium	0.65
$y^5H_7^\circ$	266 nm	Argon	0.17
$y^5H_7^\circ$	355 nm	Argon	0.13
$y^5H_7^\circ$	355 nm	Helium	0.11
a^5D_3	266 nm	Argon	0.71
a^5D_2	266 nm	Argon	0.77
a^5D_2	355 nm	Argon	0.63
a^3D_3	355 nm	Argon	0.05
a^5S_2	266 nm	Argon	2
a^5S_2	355 nm	Argon	1.5
a^5G_6	355 nm	Argon	0.91
a^3G_4	355 nm	Argon	0.08
e^7D_5	355 nm	Helium	3.5
e^7D_3	355 nm	Helium	3
f^7D_5	355 nm	Helium	0.25
f^7D_5	355 nm	Argon	0.25
f^7D_4	355 nm	Argon	0.2
f^7D_4	355 nm	Helium	0.3
Propyl-ACT			

State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	355 nm	Argon	1.5
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25
$z^7F_5^\circ$	355 nm	Argon	2.85
$y^7P_4^\circ$	355 nm	Argon	0.07
$y^7P_3^\circ$	355 nm	Argon	0.06
$z^5P_3^\circ$	355 nm	Argon	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.14
Methyl-ACT			
$z^7P_4^\circ$	355 nm	Argon	1.6, 2.5
$z^7P_4^\circ$	355 nm	Helium	3
$z^7P_4^\circ$	266 nm	Argon	1.33
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25, 1.3
$z^7F_5^\circ$	355 nm	Argon	3
$y^7P_4^\circ$	355 nm	Argon	0.07, 0.08
$y^7P_4^\circ$	355 nm	Helium	0.2
$y^7P_3^\circ$	266 nm	Argon	1.22
$y^7P_3^\circ$	355 nm	Argon	0.08
$y^7P_2^\circ$	355 nm	Argon	0.1
$z^5P_3^\circ$	266 nm	Argon	0.67
$z^5P_3^\circ$	355 nm	Argon	0.08, 0.17
$z^5P_3^\circ$	355 nm	Helium	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.09
$y^5H_7^\circ$	355 nm	Argon	0.06, 0.05
a^5D_3	266 nm	Argon	2.5
a^5D_2	266 nm	Argon	1.9
a^5D_2	355 nm	Argon	1.17
a^5S_2	266 nm	Argon	2.3
a^5S_2	355 nm	Argon	1.11
a^5G_6	355 nm	Argon	1.6
e^7D_5	355 nm	Argon	1

11.2 Figures

If your thesis has a lot of figures, LaTeX might behave better for you than that other word processor. One thing that may be annoying is the way it handles "floats" like tables and figures. LaTeX will try to find the best place to put your object based on the text around it and until you're really, truly done writing you should just leave it where it lies. There are some optional arguments to the figure and table environments

to specify where you want it to appear; see the comments in the first figure.

If you need a graphic or tabular material to be part of the text, you can just put it inline. If you need it to appear in the list of figures or tables, it should be placed in the floating environment.

To get a figure from StatView, JMP, SPSS or other statistics program into a figure, you can print to pdf or save the image as a jpg or png. Precisely how you will do this depends on the program: you may need to copy-paste figures into Photoshop or other graphic program, then save in the appropriate format.

Below we have put a few examples of figures. For more help using graphics and the float environment, see our online documentation.

And this is how you add a figure with a graphic:

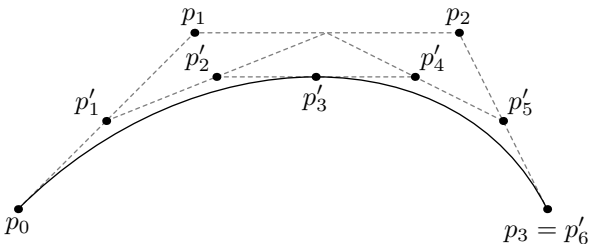


Figure 11.1: A Figure

11.3 More Figure Stuff

You can also scale and rotate figures.

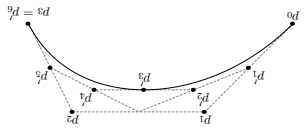


Figure 11.2: A Smaller Figure, Flipped Upside Down

11.4 Even More Figure Stuff

With some clever work you can crop a figure, which is handy if (for instance) your EPS or PDF is a little graphic on a whole sheet of paper. The viewport arguments are the lower-left and upper-right coordinates for the area you want to crop.

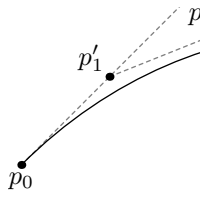


Figure 11.3: A Cropped Figure

11.4.1 Common Modifications

The following figure features the more popular changes thesis students want to their figures. This information is also on the web at web.reed.edu/cis/help/latex/graphics.html.

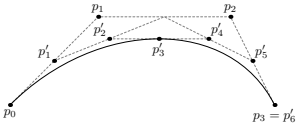


Figure 11.4: Subdivision of arc segments. You can see that $p_3 = p'_6$.

Conclusion

Here's a conclusion, demonstrating the use of all that manual incrementing and table of contents adding that has to happen if you use the starred form of the chapter command. The deal is, the chapter command in LaTeX does a lot of things: it increments the chapter counter, it resets the section counter to zero, it puts the name of the chapter into the table of contents and the running headers, and probably some other stuff.

So, if you remove all that stuff because you don't like it to say "Chapter 4: Conclusion", then you have to manually add all the things LaTeX would normally do for you. Maybe someday we'll write a new chapter macro that doesn't add "Chapter X" to the beginning of every chapter title.

4.1 More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

The First Appendix

Appendix B

The Second Appendix, for Fun