COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Aggie Research Program

Summer 2022

Graph Mining and Cybersecurity Research

August 22, 2022

Anuj Ketkar	anujketkar@tamu.edu
Wyatt McGinnis	mw8088@tamu.edu
Shurui Xu	shuruixu@tamu.edu

https://github.com/tamu-edu-students/Graph-Mining-and-Cybersecurity

Contents

1. Background and Set Up

The beginning of the research project consisted of all the team members getting acquainted with the background of the topics we would be researching as well as the technologies we would be using to perform it.

1.1 Research Papers

Each team member first learned the fundamentals on how to properly read and understand a research paper, and then familiarized themselves with specific papers related to the topic of the research. This included the papers on **Peregrine** [1], **GraphPi** [2], and **Arabesque** [3].

1.2 Linux Server

Most of the work done in graph mining requires large amounts of concurrent processing that is not feasible to do on home computers. Thus we had to SSH onto a server to perform our tests so it was essential that each of us understood how to navigate through a Linux file system using a shell. During this setup period, we also loaded the Peregrine and GraphPi source code onto the system and became comfortable with executing the binaries on the server.

2. Graph Mining Testing Tools

A large portion of the time spent during the research period was spent writing scripts and programs to automate the tests for the area we were attempting to analyze. The final goal was to be able to run the Peregrine and GraphPi engines on all variants of a given pattern where the labels have been switched around. These new labeled graphs would retain the same vertices and edges making them *isomorphic*, which means there is a one-to-one mapping between all the variants and the original. We were interested in which labeling procedures lead to the best performance in the graph mining engines so we needed to collect data on the time and memory usage when running each of these variations.

All of these tools were upload and documented on the internal repository as well as in the appendix of this document for future researchers.

2.1 Memory and Time Scripts

To measure the timing and memory usage of the graph mining engine, Wyatt McGinnis created two bash scripts to gather the data and display it in an easy to read format. Originally, there were three separate scripts, memusage, timusage, and testsuite; however, timusage was deprecated as the engines themselves kept track of the programs run time. Memusage was used to measure the RSS and dirty memory used by the engine, while testsuite was able to take in a directory of different patterns and perform timing and memory tests on each one.

2.2 GraphPi Modifications

For ease of use, the GraphPi source files pattern.cpp and pattern.h were edited so that GraphPi can read in a peregrine-formatted pattern. Also, the file pattern_count_file.cpp was added to the tianhe folder and the CMakeLists.txt file was edited to build pattern_count_file and place it in bin, which uses the new edits to run the engine on a peregrine pattern.

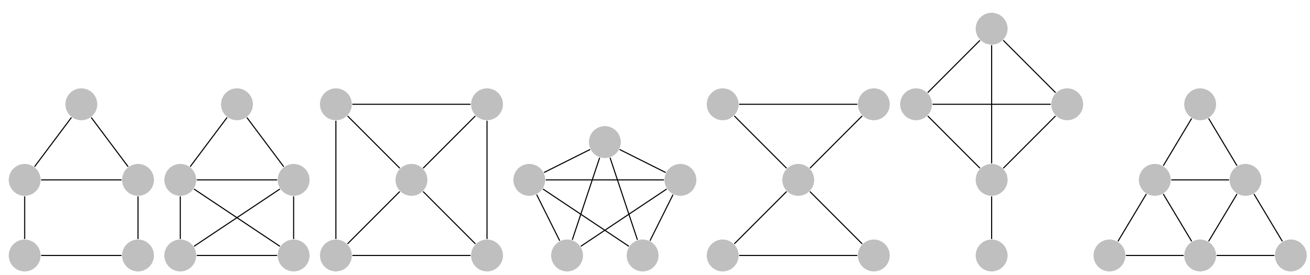
2.3 Pattern Generation Program

In order to generate all the variations for testing, Anuj Ketkar contributed isomorpher.cpp; a program that takes an input file in the Peregrine graph file format and exhaustively produces all n! isomorphic variants of the graph. Since it is necessary to visualize these patterns in order to make generalizations about them, it also accepts an optional second input of absolute coordinates for each vertex of the original input graph, which it will then cause it to produce a second output of LaTeX code which, when compiled, will display all the graphs on a PDF document using the tikz package.

3. Peregrine Data

3.1 Collection Methodology

A 5-node graphlet is a small graph consisting of 5 vertices and variable amount of edges that connect them. Several of these graphlets were written by hand on a text file and named based on resemblance to objects. A few of the patterns did not produce any outputs from Peregrine after hours of time, so they had to be removed from the test set. Below are the main patterns that were used for the Peregrine tests, which includes one 6-node graphlet as well:



From left to right: House, xHouse, xBox, Apple, Hourglass, Kite, Triforce

The patterns were each run through the exhaustive generator and all 120 variants of them were created in a directory (720 for the 6-node graphlet). We could then use the suite test script to automatically run the Peregrine count functionality on the MiCo graph three times for each pattern and average the memory and time usage for those runs. All tests were run on a remote server running Ubuntu 18.04.6 using a GNU Screen session to allow them to run in the background. We then pasted the results from the output TXT file into a CSV and used Excel formulas to sort the data and calculate relevant statistics on them.

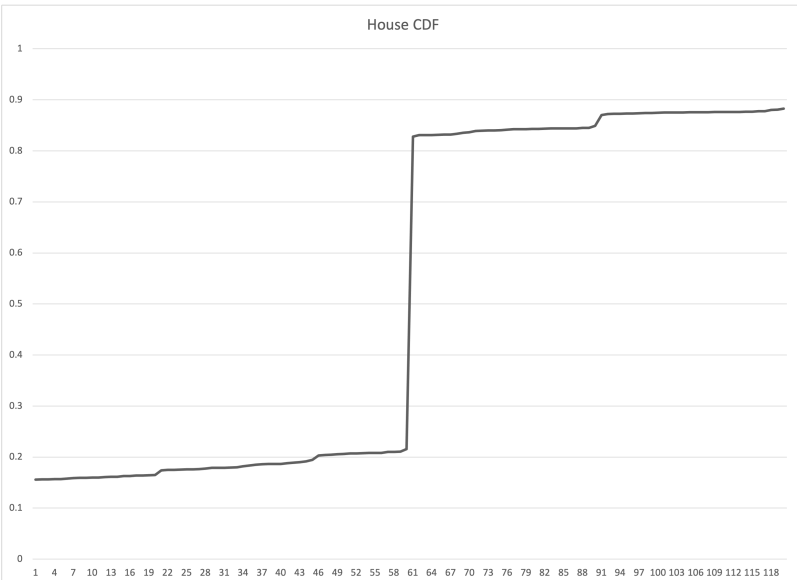
3.2 Results

For each of the seven patterns above, I have displayed the results in the form of

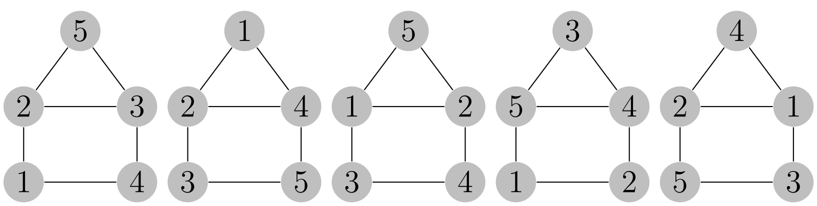
- The summary statistics of that pattern's tests
- The CDF Graph for all the variants' runtimes
- The five fastest and slowest labeling procedures

3.2.1 House

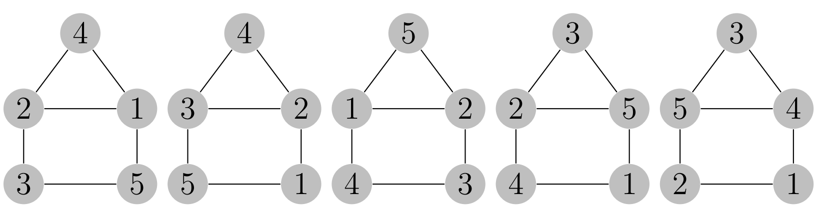
Time		**RSS Memory**		**Dirty Memory**	
Range	7.6243	Range	686	Range	488
Percent Difference	42.558%	Percent Difference	4.187%	Percent Difference	3.913%
Min	17.91503	Min	16384	Min	12472
Max	25.53933	Max	17070	Max	12960
Mean	21.69641075	Mean	16779.35	Mean	12768.2417
Standard Deviation	3.461391778	Standard Deviation	136.715963	Standard Deviation	108.144006



Fastest patterns:

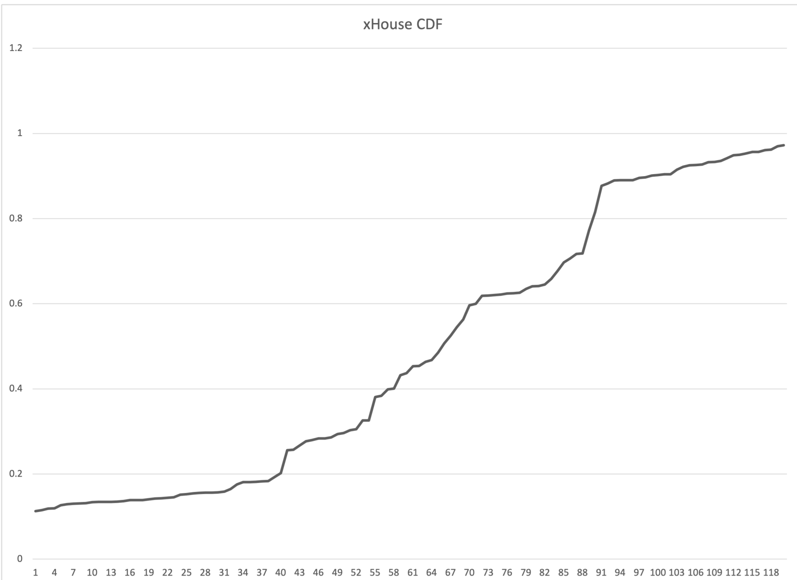


Slowest patterns:

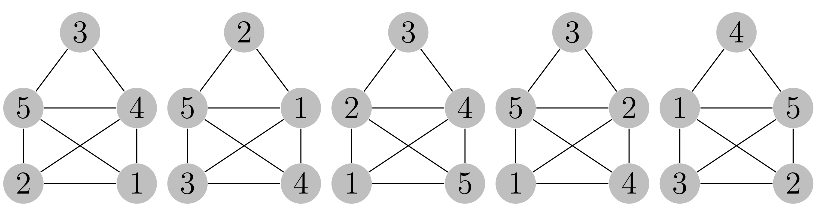


3.2.2 xHouse

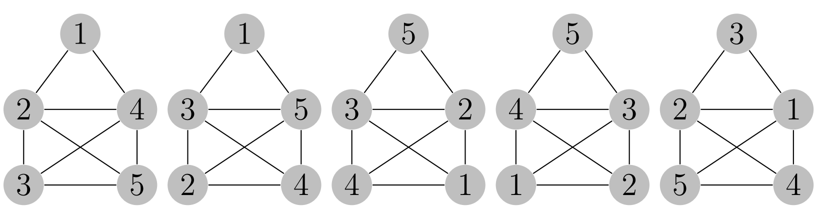
Time		**RSS Memory**		**Dirty Memory**	
Range	0.80793	Range	1644	Range	1407
Percent Difference	8.031%	Percent Difference	10.405%	Percent Difference	11.702%
Min	10.0602	Min	15800	Min	12024
Max	10.86813	Max	17444	Max	13431
Mean	10.37644667	Mean	17042.625	Mean	13044.4333
Standard Deviation	0.258518707	Standard Deviation	278.831537	Standard Deviation	256.745197



Fastest Patterns:

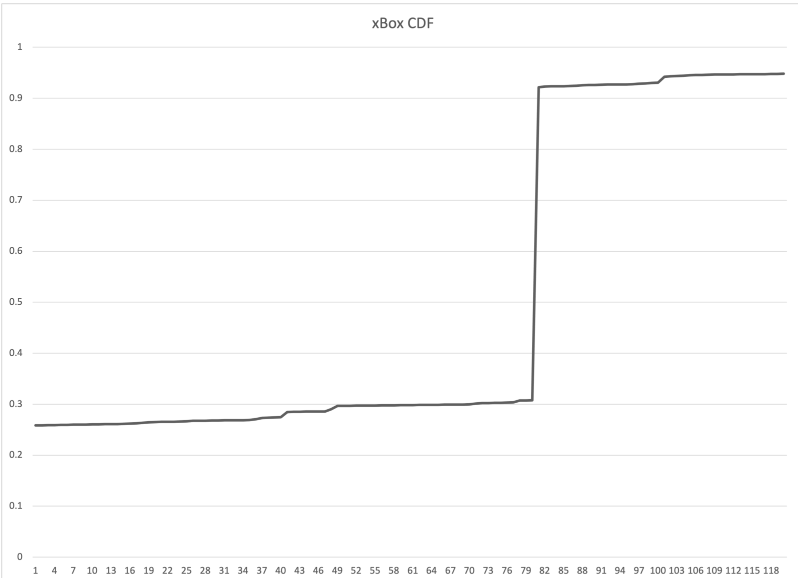


Slowest Patterns:

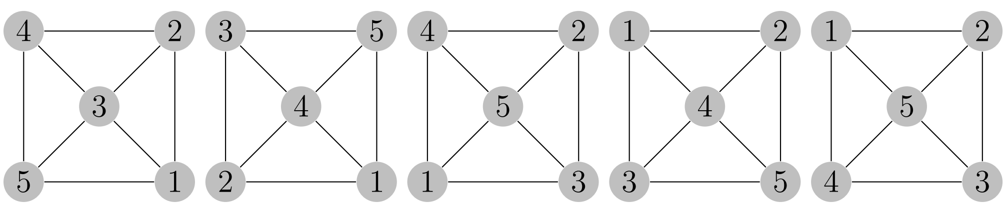


3.2.3 xBox

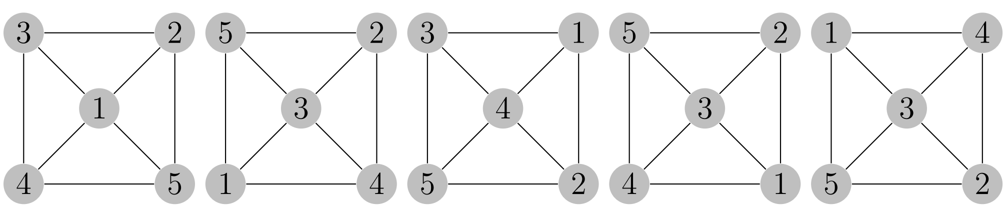
Time		**RSS Memory**		**Dirty Memory**	
Range	8.3324	Range	1027	Range	739
Percent Difference	78.557%	Percent Difference	6.517%	Percent Difference	6.185%
Min	10.60676	Min	15759	Min	11948
Max	18.93916	Max	16786	Max	12687
Mean	13.43027808	Mean	16487.9667	Mean	12484.925
Standard Deviation	3.663603462	Standard Deviation	214.670078	Standard Deviation	175.761688



Fastest Patterns:

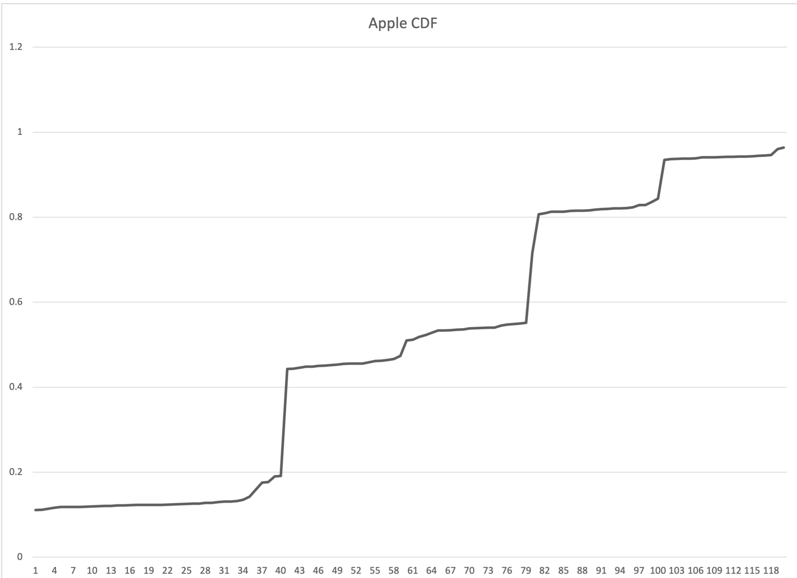


Slowest Patterns:

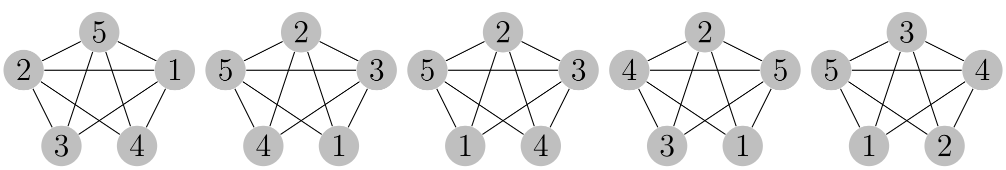


3.2.4 Apple

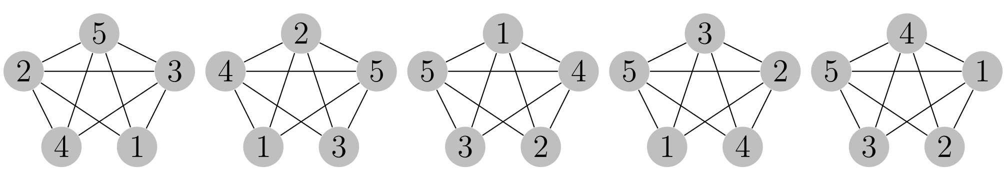
Time		**RSS Memory**		**Dirty Memory**	
Range	0.63405	Range	888	Range	892
Percent Difference	28.367%	Percent Difference	5.754%	Percent Difference	7.783%
Min	2.23518	Min	15432	Min	11461
Max	2.86923	Max	16320	Max	12353
Mean	2.500170917	Mean	15768.025	Mean	11795.7
Standard Deviation	0.209946285	Standard Deviation	179.651505	Standard Deviation	177.164719



Fastest Patterns:

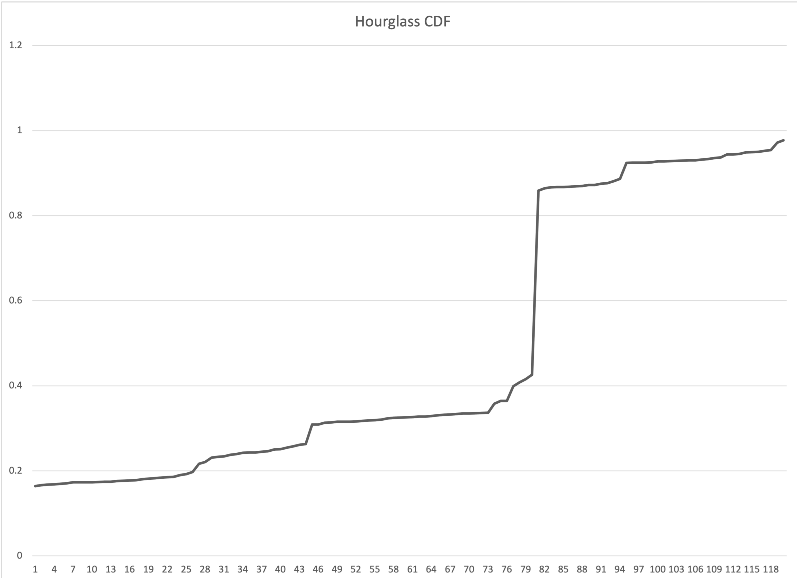


Slowest Patterns:

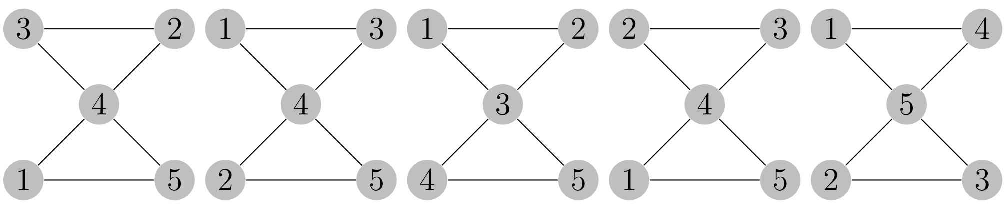


3.2.5 Hourglass

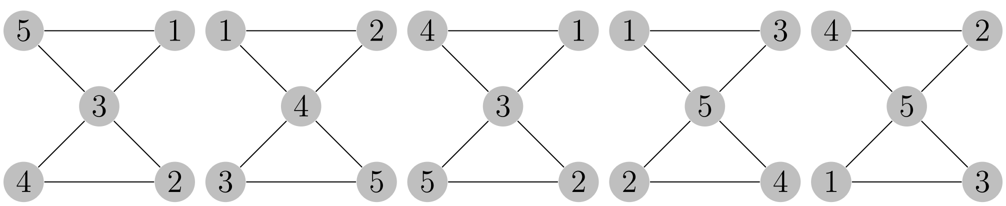
Time		**RSS Memory**		**Dirty Memory**	
Range	3.7002	Range	598	Range	439
Percent Difference	25.058%	Percent Difference	3.658%	Percent Difference	3.529%
Min	14.76633	Min	16346	Min	12439
Max	18.46653	Max	16944	Max	12878
Mean	16.02841167	Mean	16789.95	Mean	12778.2083
Standard Deviation	1.242803891	Standard Deviation	102.811556	Standard Deviation	75.6034431



Fastest Patterns:

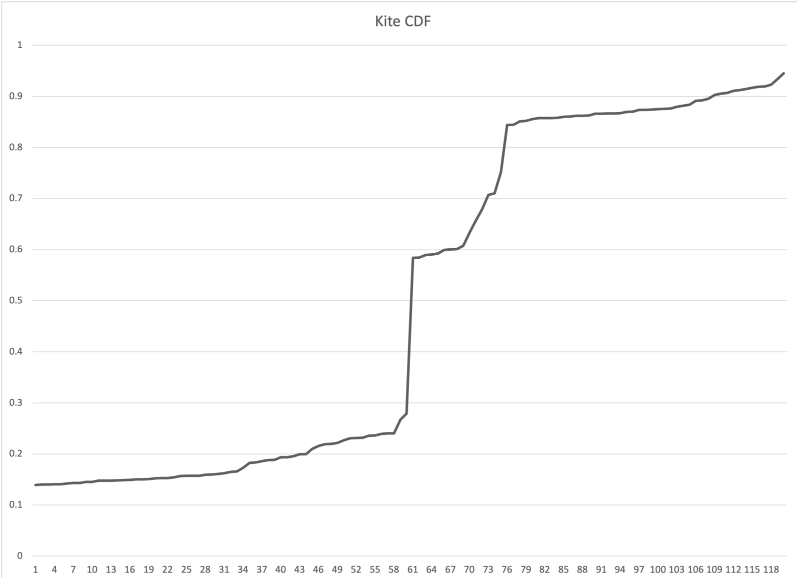


Slowest Patterns:

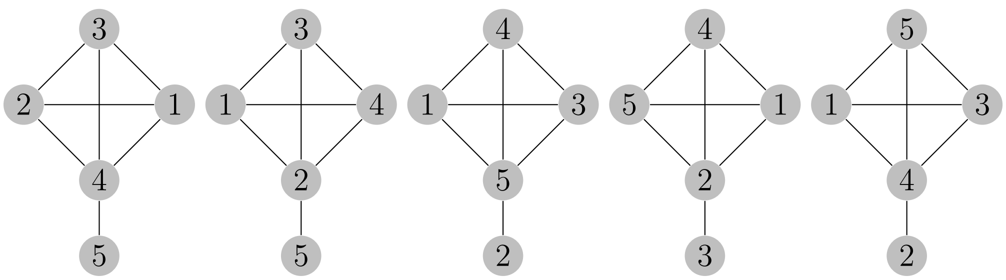


3.2.6 Kite

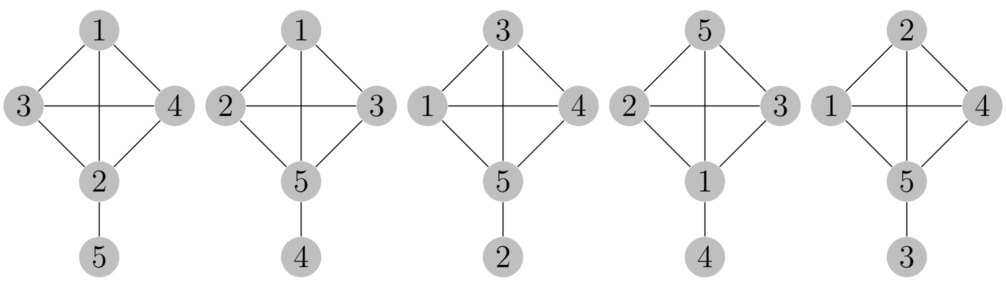
Time		**RSS Memory**		**Dirty Memory**	
Range	0.96401	Range	1349	Range	1165
Percent Difference	15.917%	Percent Difference	8.526%	Percent Difference	9.718%
Min	6.0563	Min	15823	Min	11988
Max	7.02031	Max	17172	Max	13153
Mean	6.455420833	Mean	16787.6833	Mean	12795.3917
Standard Deviation	0.359184095	Standard Deviation	256.093896	Standard Deviation	226.637878



Fastest Patterns:

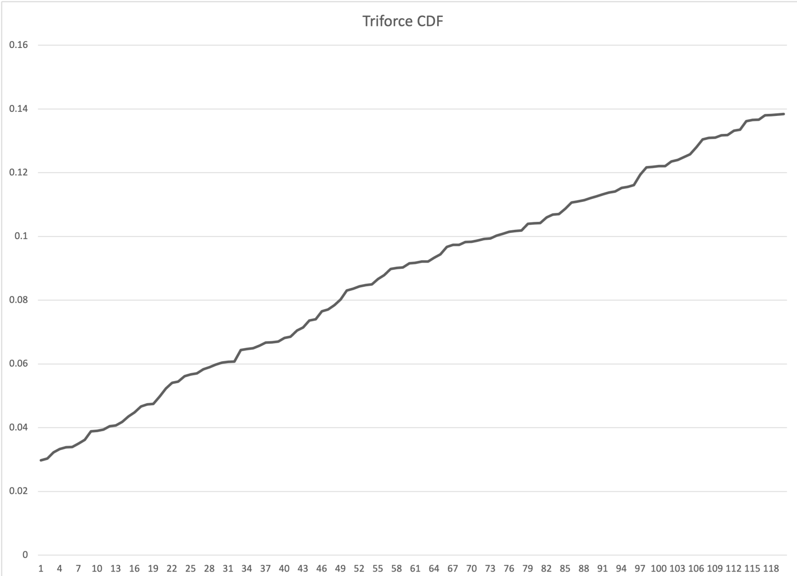


Slowest Patterns:

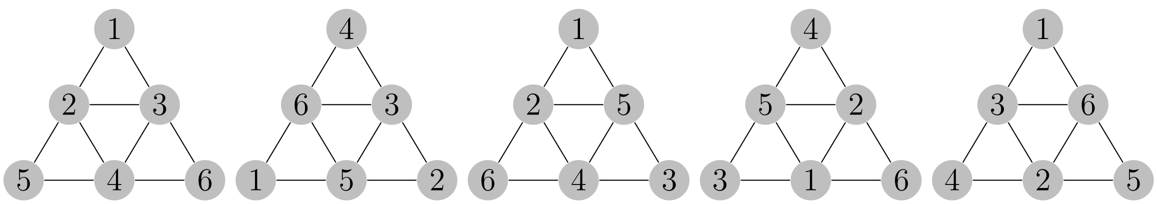


3.2.7 Triforce

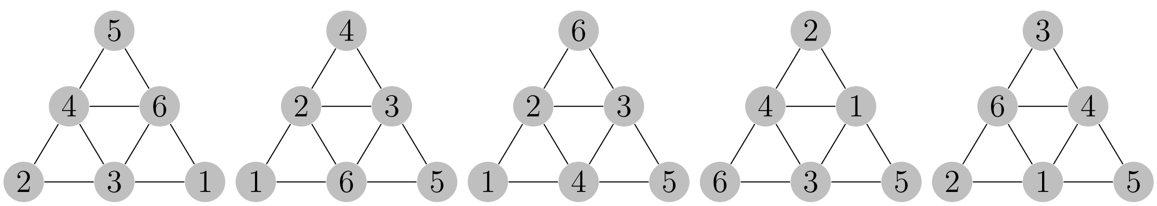
Time		**RSS Memory**		**Dirty Memory**	
Range	0.80614	Range	1289	Range	1097
Percent Difference	9.232%	Percent Difference	7.915%	Percent Difference	8.772%
Min	8.73159	Min	16285	Min	12505
Max	9.53773	Max	17574	Max	13602
Mean	9.105632306	Mean	17166.4097	Mean	13169.7319
Standard Deviation	0.197391869	Standard Deviation	229.749789	Standard Deviation	200.758199



Fastest Patterns:



Slowest Patterns:

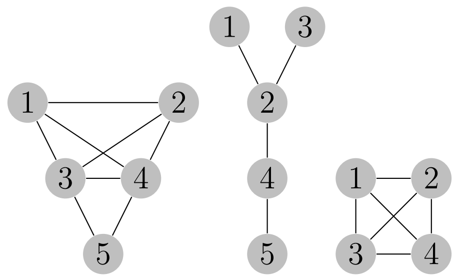


Looking at the CDF graphs for House and xBox, there appears to be single alteration that costs a significant difference in time as evident by a large spike that separates the test cases into two groups. The Apple pattern may be an even more useful indicator since there are multiple spikes throughout the distribution. The future plans of this research would involve analysis of the source code of Peregrine while running patterns from different ends of these separations and find what causes the differences.

4. GraphPi Data

4.1 Collection Methodology

The collection methodology for GraphPi was done as closely to Peregrine's as possible, to limit the effect of extraneous changes in the test results. However, three new patterns were tested on GraphPi shown below:



From left to right: Cone, Eiffel, xSquare

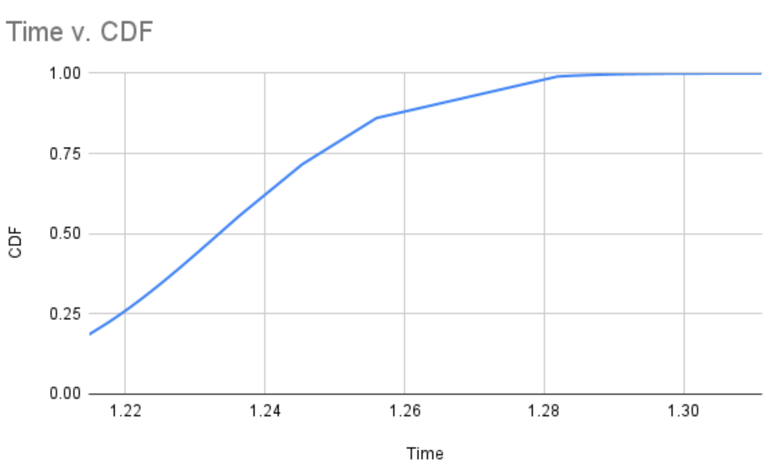
4.2 Results

For each of the seven patterns shown in the Peregrine section plus the three shown above, I have displayed the results in the form of

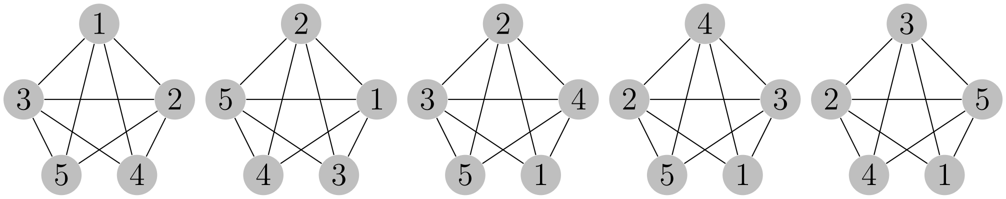
- The summary statistics of that pattern's tests
- The CDF Graph for all the variants' runtimes
- The five fastest and slowest labeling procedures

4.2.1 Apple

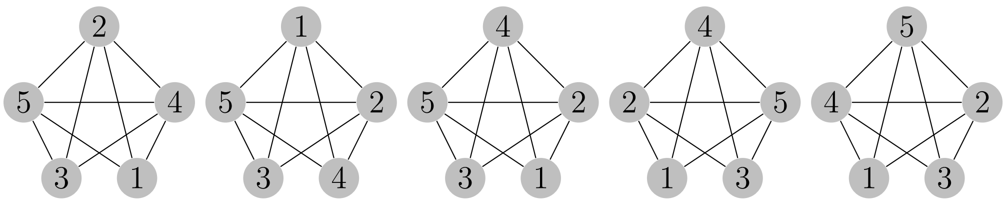
Time		**RSS Memory**		**Dirty Memory**	
Range	0.09647	Range	6414	Range	5667
Percent Difference	7.94155176%	Percent Difference	46.73564558%	Percent Difference	55.051486
Min	1.21475	Min	13724	Min	10294
Max	1.31122	Max	20138	Max	15961
Mean	1.233589669	Mean	18019.4876	Mean	13917.033
Standard Deviation	0.02084348007	Standard Deviation	928.3474935	Standard Deviation	879.69416



Fastest Patterns:

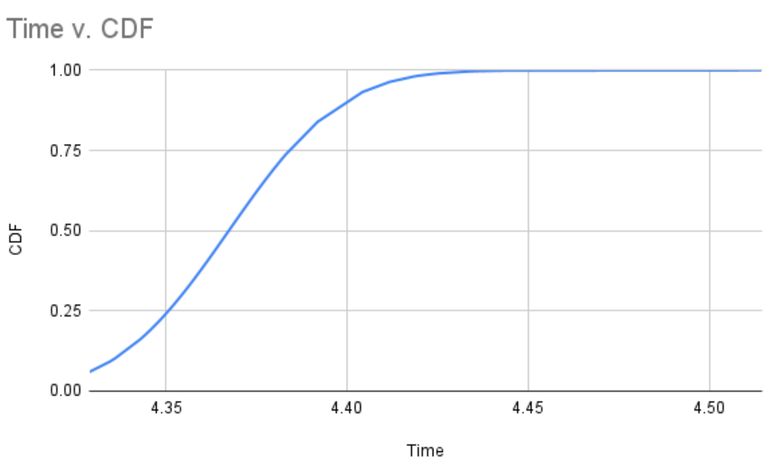


Slowest Patterns:

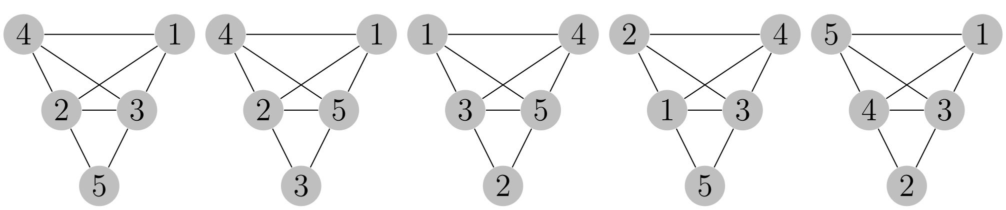


4.2.2 Cone

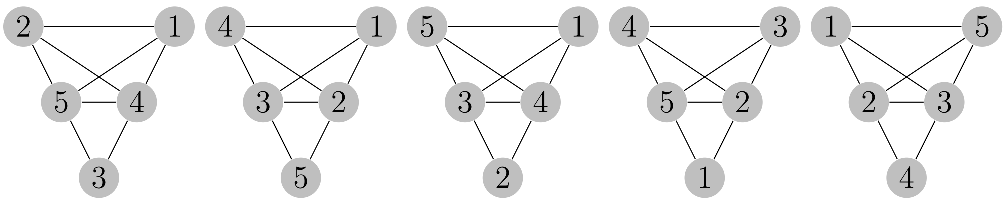
Time		**RSS Memory**		**Dirty Memory**	
Range	0.18566	Range	2080	Range	1949
Percent Difference	4.288888992%	Percent Difference	12.01756413%	Percent Difference	14.731670
Min	4.32886	Min	17308	Min	13230
Max	4.51452	Max	19388	Max	15179
Mean	4.367454793	Mean	18023.92562	Mean	13895.694
Standard Deviation	0.02470596872	Standard Deviation	448.389826	Standard Deviation	438.44341



Fastest Patterns:

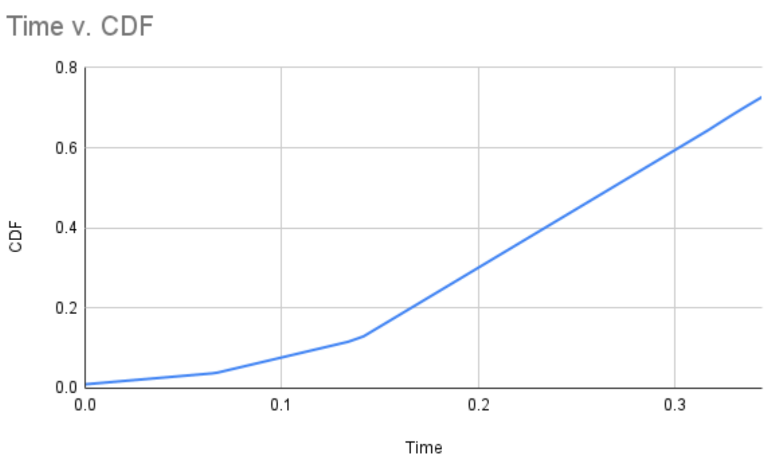


Slowest Patterns:

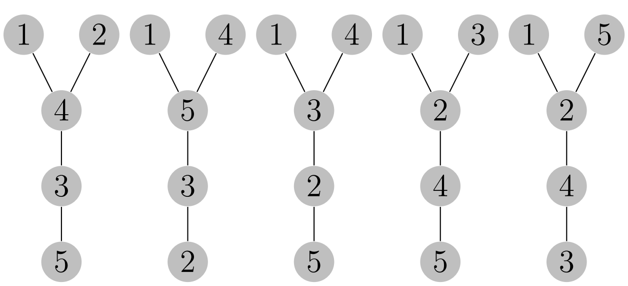


4.2.3 Eiffel

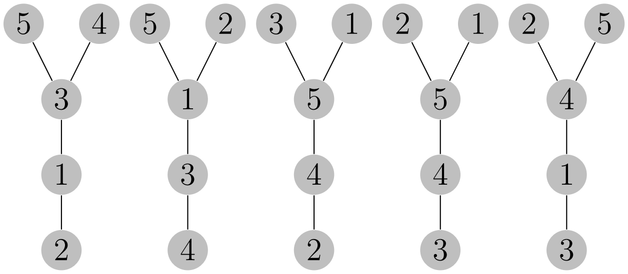
Time		**RSS Memory**		**Dirty Memory**	
Range	0.34408	Range	7946	Range	7922
Percent Difference	100.00%	Percent Difference	66.23874625%	Percent Difference	100.06315
Min	0	Min	11996	Min	7917
Max	0.34408	Max	19942	Max	15839
Mean	0.2701636364	Mean	16886.93388	Mean	12799.785
Standard Deviation	0.116925831	Standard Deviation	1469.485872	Standard Deviation	1465.1786



Fastest Patterns:

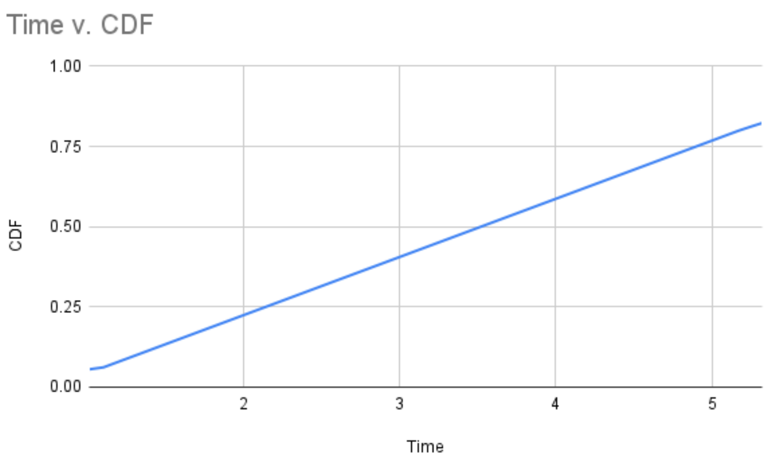


Slowest Patterns:

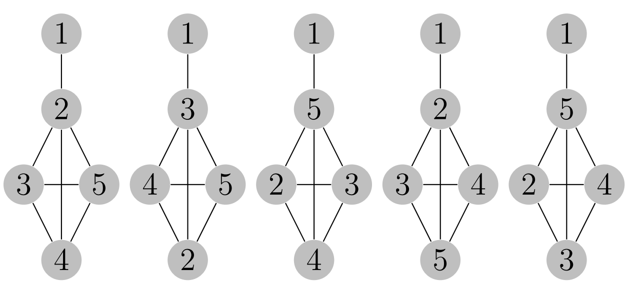


4.2.4 Kite

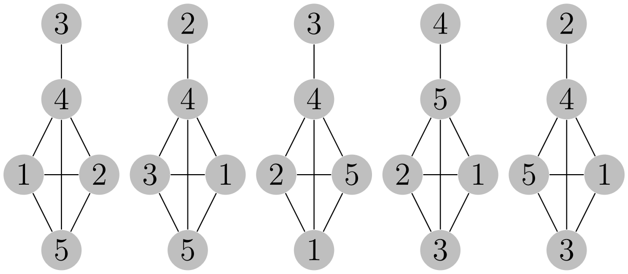
Time		RSS Memory		Dirty Memory	
Range	4.30822	Range	7939	Range	7987
Percent Difference	424.9324364%	Percent Difference	71.41957539%	Percent Difference	114.36139
Min	1.01386	Min	11116	Min	6984
Max	5.32208	Max	19055	Max	14971
Mean	4.363522397	Mean	17475.84298	Mean	13350.975
Standard Deviation	1.708609991	Standard Deviation	1650.53975	Standard Deviation	1608.2724



Fastest Patterns:

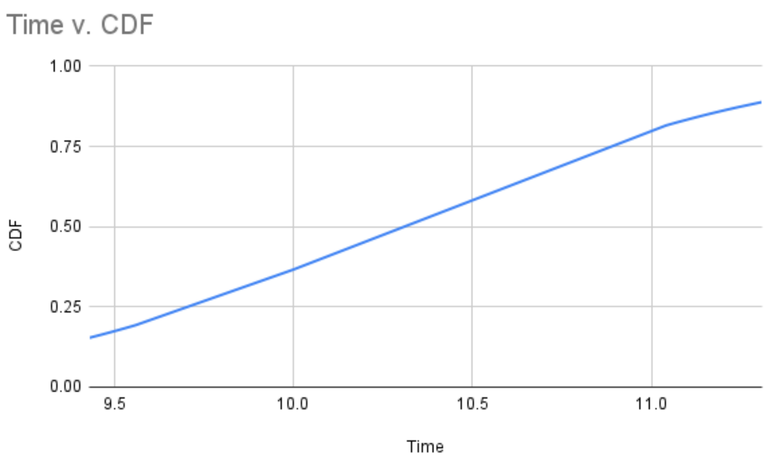


Slowest Patterns:

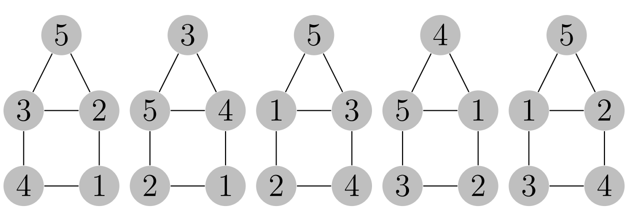


4.2.5 House

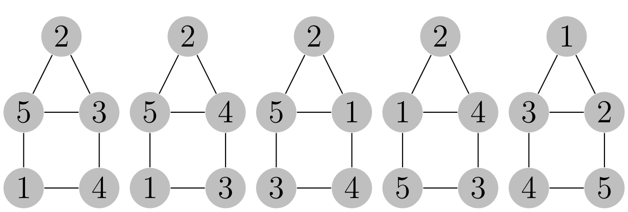
Time		**RSS Memory**		**Dirty Memory**	
Range	1.87834	Range	1656	Range	1462
Percent Difference	19.91712244%	Percent Difference	9.578344612%	Percent Difference	10.986698
Min	9.43078	Min	17289	Min	13307
Max	11.30912	Max	18945	Max	14769
Mean	10.32121446	Mean	18309.8595	Mean	14179.520
Standard Deviation	0.8388583459	Standard Deviation	262.7996482	Standard Deviation	260.12760



Fastest Patterns:

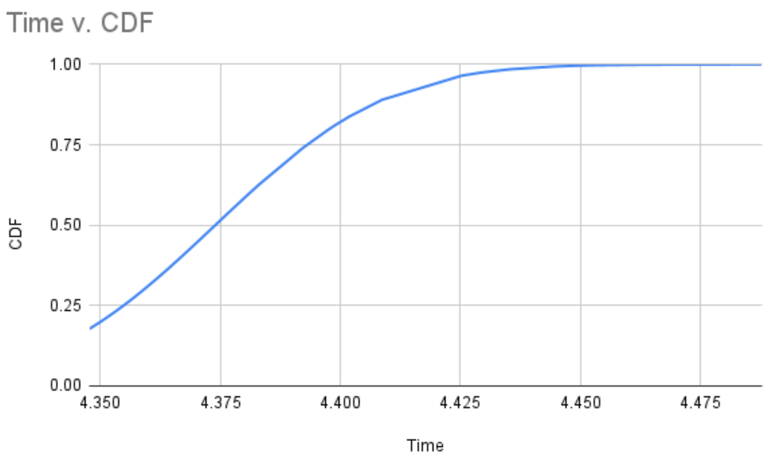


Slowest Patterns:

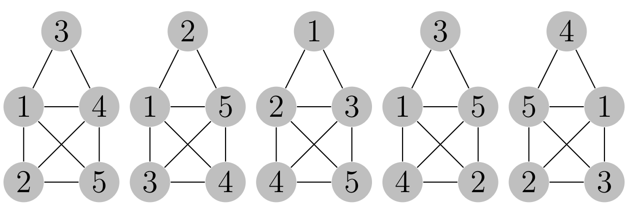


4.2.6 xHouse

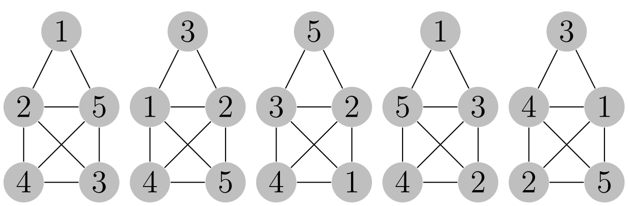
Time		**RSS Memory**		**Dirty Memory**	
Range	0.14018	Range	1749	Range	1726
Percent Difference	3.224255751%	Percent Difference	10.08185382%	Percent Difference	12.97256
Min	4.34767	Min	17348	Min	13305
Max	4.48785	Max	19097	Max	15031
Mean	4.374022893	Mean	17977.28099	Mean	13839.10
Standard Deviation	0.02833383697	Standard Deviation	446.3755561	Standard Deviation	443.8182



Fastest Patterns:

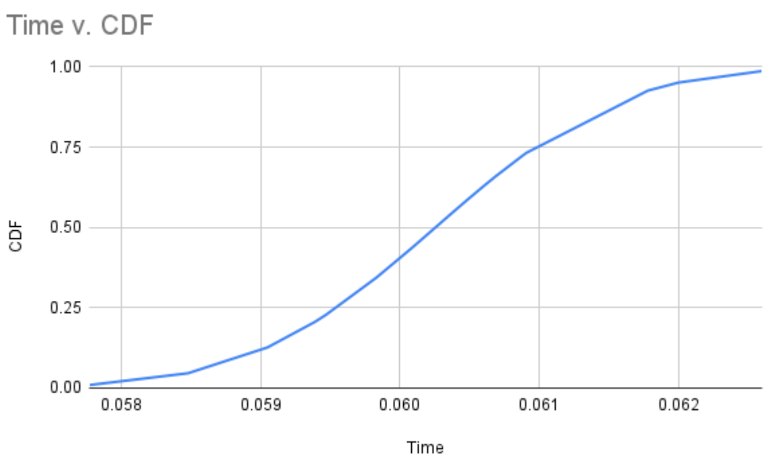


Slowest Patterns:

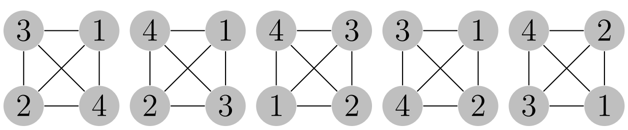


4.2.7 xSquare

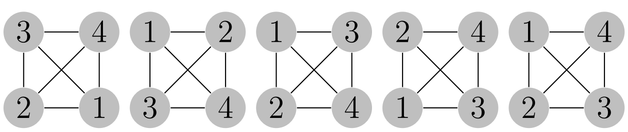
Time		**RSS Memory**		**Dirty Memory**	
Range	0.00483	Range	7192	Range	6151
Percent Difference	8.360740869%	Percent Difference	54.98050608%	Percent Difference	61.24054
Min	0.05777	Min	13081	Min	10044
Max	0.0626	Max	20273	Max	16195
Mean	0.0602644	Mean	14768.08	Mean	11438.12
Standard Deviation	0.001054079693	Standard Deviation	2213.238199	Standard Deviation	1806.823



Fastest Patterns:

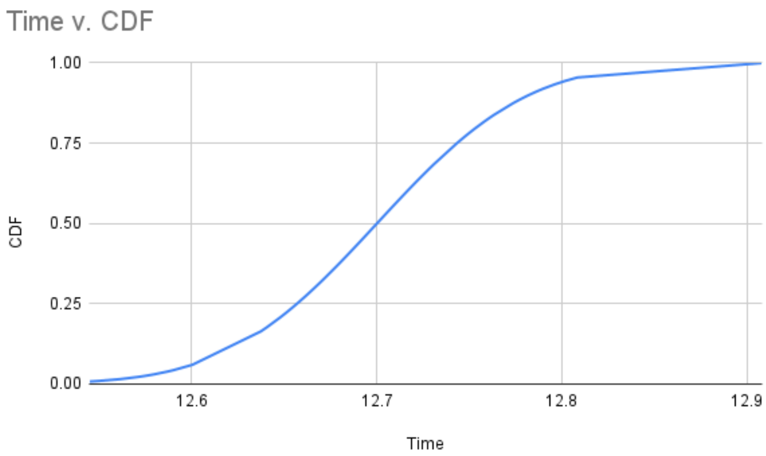


Slowest Patterns:

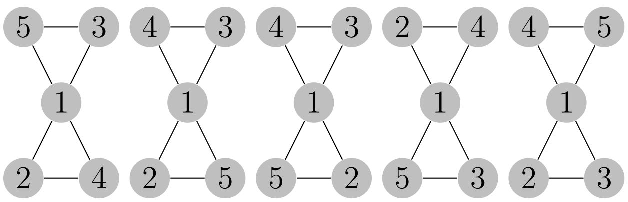


4.2.8 Hourglass

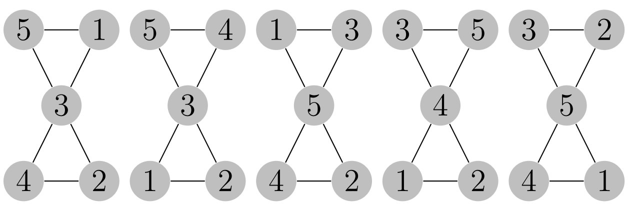
Time		**RSS Memory**		**Dirty Memory**	
Range	0.3634	Range	1574	Range	1335
Percent Difference	9.083564174%	Percent Difference	9.964173757%	Percent Difference	2.896829
Min	12.54475	Min	17328	Min	13398
Max	12.90815	Max	18902	Max	14733
Mean	12.70033669	Mean	18335.09917	Mean	14186.06
Standard Deviation	0.06407778739	Standard Deviation	249.1605709	Standard Deviation	237.4201



Fastest Patterns:

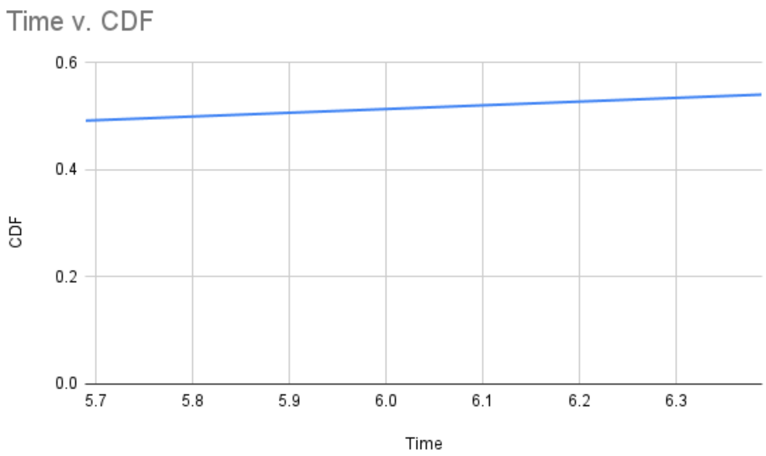


Slowest Patterns:

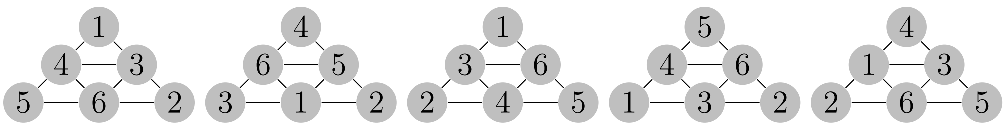


4.2.9 Triforce

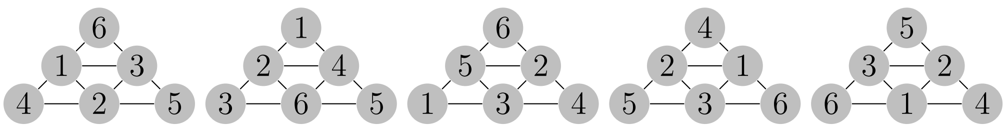
Time		**RSS Memory**		**Dirty Memory**	
Range	0.70027	Range	2881	Range	2400
Percent Difference	12.30943119%	Percent Difference	17.53926702%	Percent Difference	18.891687
Min	5.68889	Min	16426	Min	12704
Max	6.38916	Max	19307	Max	15104
Mean	5.741784189	Mean	18305.26075	Mean	14166.782
Standard Deviation	0.06195568222	Standard Deviation	423.3537839	Standard Deviation	407.93912



Fastest Patterns:

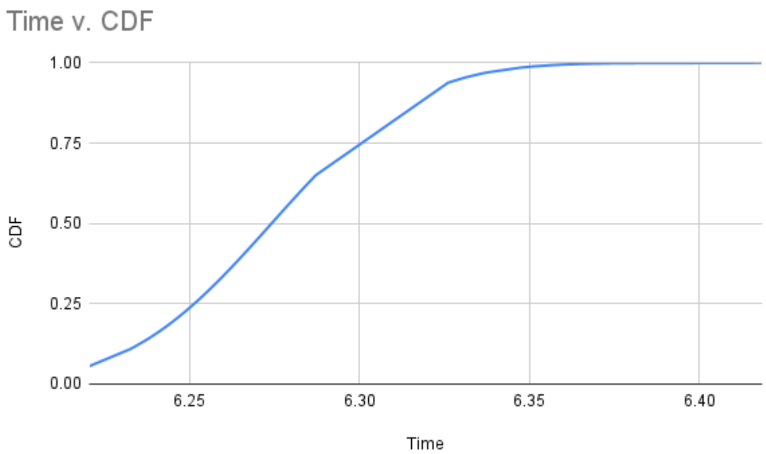


Slowest Patterns:

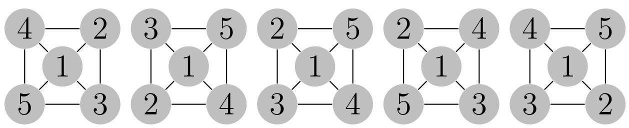


4.2.10 xBox

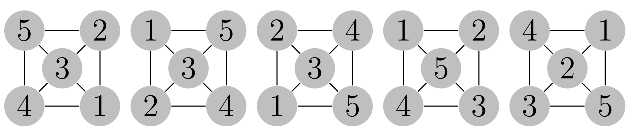
Time		**RSS Memory**		**Dirty Memory**	
Range	0.19796	Range	2687	Range	2397
Percent Difference	3.182426882%	Percent Difference	16.34726532%	Percent Difference	19.091995
Min	6.22041	Min	16437	Min	12555
Max	6.41837	Max	19124	Max	14952
Mean	6.274158182	Mean	18005.38017	Mean	13895.033
Standard Deviation	0.03372494645	Standard Deviation	609.3543339	Standard Deviation	537.82634



Fastest Patterns:



Slowest Patterns:



5. Cybersecurity

5.1 Introduction

The Cyberseucrity json files contain a list of information, including packet dates, times, frames, source MAC addresses, and destination MAC addresses. There are two main json files in the coe-fs server provided by Dr. Khanh Nguyen from TAMU, one named output_benign.json and the other output_icmp_3_attack.json. The benign file contains 193GB of data and the attack file contains 111GB of data. Although they are in different formats, both data sets are represented as large arrays that can be parsed using some particular technique to reduce data extraction time. The goal is to find a way that GraphPi can read in the Cybersecurity files so that a graph can be built. Further tests, such as representing the resulted graph visually, finding the patterns of certain clusters, and finding the attacker's habits are encouraged in a future study.

5.2 JSON Parsing Methods

In order to turn a list containing useful and not-so-useful information into a graph, the group has decided to use the source and destination MAC addressed are vertices connected to each other. Since GraphPi does not accept inputs of multiple edges from the same vertices, it will automatically assume the over lapsing data packets as the same edge. Peregrine does not support such behavior at all. The difficult part comes when parsing a super-large json file that cannot be reformatted easily. There are several possible solutions:

1. Using Python, extract the two MAC addresses using the built in package json (import json).
2. Using Python or C++, extract the MAC addresses as vectors (shown below) in a new C++ file. This method reduces future access time while increases initial compilation running time.

```
const std::vector<GpsPt> route_1 =
    {
      { 35.6983464357, -80.4201474895},
      { 35.6983464403, -80.4201474842},
      // several hundred more lines like this
    };
const std::vector<GpsPt> route_2 =
    {
      { 35.8693464357, -80.1420474895},
      { 35.8693464392, -80.1420474821},
       // another thousand lines
    };
// more routes like this
```

3. Using an already built custom library that specializes in parsing large json files.

5.3 JSON Parsing Tests

In order to test the practicality and speeds of each option, the experiments started with python scripting. The following script extracts each MAC addresses in a pair and places them into a .txt file.

```
#author: Shurui Xu
#Texas A&M Graph-Mining-and-Cybersecurity
import json

f = open('benign.json')
data = json.load(f)

w = open("mac_addresses.txt", "w")

for i in data:
    w.write(i["_source"]["layers"]["arp"]["arp.src.hw_mac"] + "\n")
    w.write(i["_source"]["layers"]["arp"]["arp.dst.hw_mac"] + "\n")
f.close()
w.close()
```

The result of running the first four data packets is shown below.

```
12:22:0a:01:01:64
00:00:00:00:00:00
12:22:0a:01:01:64
00:00:00:00:00:00
3c:a8:2a:fe:04:80
12:22:0a:01:01:64
12:22:0a:01:01:65
00:00:00:00:00:00
```

While these are the mac addresses represented in octet, GraphPi requires an integer representation of these digits. So, another python script is used to convert the octets into integers.

```
#author: Shurui Xu
#Texas A&M Graph-Mining-and-Cybersecurity
import re
f = open('mac_addresses.txt')
w = open("mac_address_int.txt", "w")

lines = f.readlines()
for line in lines:
    mac_int = int(line.translate(str.maketrans('','',":")), 16)
    w.write(str(mac_int) + '\n')
f.close()
```

After the conversion, the result of the integers are saved in another .txt file.

```
19937406026084
0
19937406026084
0
66692973462656
19937406026084
19937406026085
0
```

Using Python, we can easily modify the parameters of the input and output for future pattern findings if there is a need to use other data than MAC addresses.

The second approach of generating C++ data sets was also experimented on. The following Python script generates pairs in a vector format.

```
#author: Shurui Xu
#Texas A&M Graph-Mining-and-Cybersecurity

import json
import re

f = open('benign.json')
data = json.load(f)

w = open("mac_addresses_in_int.cpp", "w")

for i in data:
    w.write("{")
    w.write(str(int(i["_source"]["layers"]["arp"]["arp.src.hw_mac"].translate(str
    w.write(str(int(i["_source"]["layers"]["arp"]["arp.dst.hw_mac"].translate(str

f.close()
w.close()
```

The result is stored in a C++ file.

```
{19937406026084, 0},
{19937406026084, 0},
{66692973462656, 19937406026084},
{19937406026085, 0},
```

While these methods are easy to manipulate, the efficiency is low and the program can be unstable. Thus, the last method of importing a library is used. We find that simdjson (https://github.com/simdjson/simdjson) has the functionality and speed to parse the file we have. After following the instructions to install the necessary files from the simdjson GitHub, we have made some changes to the provided quickstart.cpp example program that finds the last piece of data in a twitter data set. The new program is named benign.cpp and can extract information from the benign.json file. The following is the code section for the program.

```
#include <iostream>
#include <string>
#include <fstream>

#include "simdjson.h"

using namespace simdjson;

int main(void) {
    ondemand::parser parser;
    padded_string json = padded_string::load("benign.json");
    ondemand::document packets = parser.iterate(json);
    int i = 0;
    std::ofstream myfile;
    myfile.open("benignExtracted.txt");

    for (auto value : packets) {
        std::cout << value["_source"]["layers"]["arp"]["arp.src.hw_mac"].get_stri
        << ",␣"
        << value["_source"]["layers"]["arp"]["arp.dst.hw_mac"].get_string()
        << "\n";
    }
    myfile.close();
}
```

Since simdjson reads in the file as a list, we used the built-in iterator that quickly scans through each of the MAC addresses in the packets. The C++ program outputs an executable program which can be ran to output the MAC addresses.

```
12:22:0a:01:01:64, 00:00:00:00:00:00
12:22:0a:01:01:64, 00:00:00:00:00:00
3c:a8:2a:fe:04:80, 12:22:0a:01:01:64
12:22:0a:01:01:65, 00:00:00:00:00:00
```

Then, the octets can be converted to integers using the method previously mentioned. By using simdjson, we can effectively combine the first method with the second method and output an executable that can be run at any time to find the indexed MAC addresses efficiently.

5.4 Parsing a Mounted File

The next natural step would be using simdjson on the actual file. Another challenge arises from importing the file using simdjson. The experiments above have all been done locally so far, meaning that file routes and names can be identified locally. However, simdjson does not seem to be working with a file destination that includes any folder paths. For example,

```
load("benign.json")
```

would be recognized if benign.json is in the same folder as the simdjson library. If there is any folder naming, such as

```
load("\desktop\benign.json")
```

an error occurs when compiling.

```
libc++abi: terminating with uncaught exception of
type simdjson::simdjson_error: Error reading the file.
zsh: abort      ./benign
```

There should exist ways to overcome the error, which would be a future task to explore.

5.5 Future Plans

After fixing the issue of file destination, we can mount the file to the computation server provided by TAMU. Then, we would upload the program onto the server and run the parsing process. Once the results are in the correct format, we can run GraphPi on benign Cybersecurity graph. The same process should be done to the attack Cybersecurity graph. After the graph processings, we can begin to explore the transactions of packets between users and how the vertices connect visually. Eventually, it would be beneficial to identify patterns, such as houses, apples, kites, and xboxes mentioned in "Collection Methodology" under "Peregrine Data". Future tasks will conclude the experiment on graph mining using GraphPi and Peregrine.

References

[1] K. Jamshidi, R. Mahadasa, and K. Vora. Peregrine: A pattern-aware graph mining system. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[2] T. Shi, M. Zhai, Y. Xu, and J. Zhai. Graphpi: High performance graph pattern matching through effective redundancy elimination. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020.

[3] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga. Arabesque: A system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 425–440, New York, NY, USA, 2015. Association for Computing Machinery.

A. Appendix

A.1 isomorpher.cpp

```cpp
// Author: Anuj Ketkar
// Aggie Research Program: Graph Mining and Cybersecurity
// Exhuastivly generates all isomorphic patterns to the input graph
// and creates a LaTeX pdf of variants if there is a coordinate file

#include <iostream>
#include <string>
#include <vector>
#include <unordered_set>
#include <sstream>
#include <fstream>

using namespace std;

// Global variables
int numEdges;
int numVertices;

// Helper functiton to swaps two vertices
void swap(vector<int> &a, vector<int> &b, int first, int second)
{
    for (int i = 0; i < numEdges; ++i)
    {
        if (a[i] == first) a[i] = second;
        else if (a[i] == second) a[i] = first;

        if (b[i] == first) b[i] = second;
        else if (b[i] == second) b[i] = first;
    }
}

// Displays current graph as string
string display(vector<int> &a, vector<int> &b)
{
    ostringstream stream;
    for (int i = 0; i < numEdges; ++i)
        stream << a[i] << " " << b[i] << "\n";
    string text(stream.str());
    return text;
}

// Exhaustivly calculates all isomorphic graphs recursively
void exhaustive(unordered_set<string> &permutations, vector<int> &a,
  vector<int> &b, int prev, int callStack)
{
```

```
    for (int i = 1; i <= numVertices; ++i)
    {
        swap(a, b, prev, i);
        string key = display(a, b);
        permutations.insert(key);
        if (callStack < numVertices)
            exhaustive(permutations, a, b, i, callStack + 1);
    }
}

//Appends each entry to the LaTeX code
void latexAppend(ofstream& latexWriter, vector<int> a, vector<int> b,
  vector<pair<double, double> > coordinates, vector<int> map)
{
    latexWriter << "\t\\begin{tikzpicture}" << endl;
    latexWriter << "\t\\tikzstyle{vertex}=[circle,fill=black!25,"
    "minimum size=12pt,inner sep=2pt]" << endl;

    for(int i = 0; i < numVertices; ++i)
    {
        string xcoord = to_string(coordinates[i].first);
        xcoord.erase(xcoord.find_first_of('.') + 3, string::npos);
        string ycoord = to_string(coordinates[i].second);
        ycoord.erase(ycoord.find_first_of('.') + 3, string::npos);
        latexWriter << "\t\\node[vertex] (" << (i+1) <<  ") at ("
        << xcoord  << "," << ycoord << ") {" << map[i+1] << "};" << endl;
    }
    for(int j = 0; j < numEdges; ++j)
        latexWriter << "\t\\draw (" << a[j] << ") -- (" << b[j] << ");" << endl;

    latexWriter << "\t\\end{tikzpicture}" << endl;
}

//Finds the alterations between the original and the given input
vector<int> mapAlterations(string input, int varNum, vector<int> &original)
{
    ifstream scanner;
    scanner.open(input + to_string(varNum) + ".graph");
    vector<int> altered;

    while(scanner)
    {
        int temp;
        scanner >> temp;
        if (scanner.fail())
            break;
        altered.push_back(temp);
    }
    scanner.close();

```

```
    vector<int> map(numVertices+1,0);
    for(int i = 0; i < original.size(); ++i)
        map[original[i]] = altered[i];
    return map;
}

// Main method
int main()
{
    numEdges = 0;
    numVertices = 0;

    // File input
    string input;
    cout << "Enter graph name (without extension): ";
    cin >> input;
    ifstream fileReader;
    fileReader.open(input + ".graph");
    if (!fileReader.is_open())
    {
        cout << "Could not open file." << endl;
        return 1;
    }

    // Vectors that will hold the graphs
    vector<int> a;
    vector<int> b;
    vector<int> original;

    // Extract information from graph
    while (fileReader)
    {
        int temp;
        fileReader >> temp;
        if (fileReader.fail())
            break;

        a.push_back(temp);
        original.push_back(temp);
        if (temp > numVertices)
            numVertices = temp;

        fileReader >> temp;
        b.push_back(temp);
        original.push_back(temp);
        if (temp > numVertices)
            numVertices = temp;
        numEdges++;
    }
    fileReader.close();
```

```

    //Store originals for mapping differences
    vector<int> sourceA = a;
    vector<int> sourceB = b;

    //Exhasutively generate the permutations
    unordered_set<string> permutations;
    exhaustive(permutations, a, b, 1, 1);

    //Output files
    cout << "Generated all " << permutations.size() << " isomorphic variants of"
    "the input graph." << endl;
    cout << "Write all the files in the current directory? (y/n)" << endl;
    char response1;
    cin >> response1;
    if (! (response1 == 'y') && ! (response1 == 'Y'))
        return 0;
    int count = 1;
    for (unordered_set<string>::iterator iter = permutations.begin();
    iter != permutations.end(); iter++)
    {
        string output = input + to_string(count) + ".graph";
        ofstream fileWriter(output);
        fileWriter << *iter;
        count++;
        fileWriter.close();
    }
    cout << "Successfully created all graph files.\n" << endl;

    //LaTeX Genetation
    cout << "Is there a " << input << ".txt file containing coordinates for the"
    " LaTeX display? (y/n)" << endl;
    char response2;
    cin >> response2;
    if(!(response2 == 'y') && !(response2 == 'Y'))
        return 0;

    //Store the absolute coordinates from user
    vector<pair<double,double> > coordinates;
    ifstream pointReader;
    pointReader.open(input + ".txt");
    if (!pointReader.is_open())
    {
        cout << "Could not find file." << endl;
        return 1;
    }
    for(int i = 0; i < numVertices; ++i)
    {
        pair<double,double> point;
        pointReader >> point.first;
```

```
        pointReader >> point.second;
        coordinates.push_back(point);
    }
    pointReader.close();

    //Create header in LaTeX code
    string name =  input + ".tex";
    ofstream latexWriter(name);
    latexWriter << "\\documentclass{article}" << endl;
    latexWriter << "\\usepackage[utf8]{inputenc}" << endl;
    latexWriter << "\\usepackage{nopageno}" << endl;
    latexWriter << "\\usepackage{tikz}" << endl;
    latexWriter << "\\begin{document}" << endl;

    //Append each entry to code
    for(int i = 1; i <= permutations.size(); ++i)
    {
        latexWriter << "\\begin{center}" << endl;
        latexWriter << "\\par\\noindent " << input << i << "\\par" << endl;
        vector<int> map = mapAlterations(input,i,original);
        latexAppend(latexWriter,a,b,coordinates,map);
        latexWriter << "\\end{center}" << endl;
    }
    latexWriter << "\\end{document}" << endl;
    cout << "Created " << input << ".tex" << " to display graphs in "
    "LaTeX" << endl;

    return 0;
}
```

A.2 memusage.sh

```
#!/usr/bin/env bash
# memusage -- Measure memory usage of processes
# Usage: memusage COMMAND [ARGS]...
#
# Author: Wyatt McGinnis
# Created: 2022-06-23
# Based on: memusage.sh by Jaeho Shin <netj@sparcs.org>
############################################################################
# Copyright 2022 Wyatt McGinnis.                                           #
#                                                                          #
# Licensed under the Apache License, Version 2.0 (the "License");         #
# you may not use this file except in compliance with the License.         #
# You may obtain a copy of the License at                                  #
#                                                                          #
#     http://www.apache.org/licenses/LICENSE-2.0                           #
#                                                                          #
# Unless required by applicable law or agreed to in writing, software      #
# distributed under the License is distributed on an "AS IS" BASIS,        #
```

```
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. #
# See the License for the specific language governing permissions and      #
# limitations under the License.                                           #
############################################################################
#Config                                                                    #
#                                                                          #
#Filename                                                                  #
outfile="memprofile.txt"                                                   #
runs=2                                                                     #
############################################################################

set -um

#Check input
[[ $# -gt 0 ]] || { sed -n '2,/^#$/ s/^# //p' <"$0"; exit 1; }

#Set file
echo "Time________________|__RSS__|_Dirty_|_Total_|_Run_" > $outfile

#Global variables
glbpeakrss=0; glbpeakdirty=0; glbpeaktotal=0;
glbsumrss=0; glbsumdirty=0; glbsumtotal=0;

for i in $(seq 1 1 $runs); do
    $@ &
    pgid=$!

    #Kill process on script exit
    trap "kill $pgid 2> /dev/null" EXIT

    #Monitoring memory usage
    peakrss=0; peakdirty=0; peaktotal=0;
    sumrss=0; sumdirty=0; sumtotal=0; cnt=0;
    while kill -0 $pgid 2> /dev/null; do
        temp=`pmap -x $pgid | grep total | grep -oP "[0-9]+.*"| sed "s/\s\+/ /g"`
        total=`echo $temp | cut -d ' ' -f1`
        rss=`echo $temp | cut -d ' ' -f2`
        dirty=`echo $temp | cut -d ' ' -f3`
        let peaktotal="total > peaktotal ? total : peaktotal"
        let peakdirty="dirty > peakdirty ? dirty : peakdirty"
        let peakrss="rss > peakrss ? rss : peakrss"
        let sumtotal="sumtotal + total"
        let sumdirty="sumdirty + dirty"
        let sumrss="sumrss + rss"
        let cnt="cnt + 1"
        date=`date +20%y-%m-%d-%H-%M-%S`
        if [ ! -z $total ]; then
            printf "%-20s|%+7s|%+7s|%+7s|%+5s\n" $date $rss $dirty $total $i >> $
        fi
        sleep 1
```

```
    done
    printf "%-20s|%+7s|%+7s|%+7s|%+5s\n" Maximum: $peakrss $peakdirty $peaktotal
    printf "%-20s|%+7s|%+7s|%+7s|%+5s\n" >> $outfile
    let glbpeaktotal="peaktotal > glbpeaktotal ? peaktotal : glbpeaktotal"
    let glbpeakdirty="peakdirty > glbpeakdirty ? peakdirty : glbpeakdirty"
    let glbpeakrss="peakrss > glbpeakrss ? peakrss : glbpeakrss"
    let glbsumrss="glbsumrss + sumrss / cnt"
    let glbsumdirty="glbsumdirty + sumdirty / cnt"
    let glbsumtotal="glbsumtotal + sumtotal / cnt"
done

    let glbsumrss="glbsumrss / runs"
    let glbsumdirty="glbsumdirty / runs"
    let glbsumtotal="glbsumtotal / runs"

printf "%-20s|%+7s|%+7s|%+7s|%+5s\n" Maximum: $glbpeakrss $glbpeakdirty $glbpeakt
printf "%-20s|%+7s|%+7s|%+7s|%+5s\n" Average: $glbsumrss $glbsumdirty $glbsumtota
```

A.3 testsuite.sh

```
#!/usr/bin/env bash
# testsuite -- Run a testsuite on a set of patterns locaated in the directory
# Usage: testsuite.sh PATTERN_DIR COMMAND [ARGS]...
#
# Author: Wyatt McGinnis
# Created: 2022-06-23
###########################################################################
# Copyright 2022 Wyatt McGinnis.                                          #
#                                                                         #
# Licensed under the Apache License, Version 2.0 (the "License");         #
# you may not use this file except in compliance with the License.        #
# You may obtain a copy of the License at                                 #
#                                                                         #
#     http://www.apache.org/licenses/LICENSE-2.0                          #
#                                                                         #
# Unless required by applicable law or agreed to in writing, software     #
# distributed under the License is distributed on an "AS IS" BASIS,       #
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. #
# See the License for the specific language governing permissions and     #
# limitations under the License.                                          #
###########################################################################
#Config                                                                   #
#                                                                         #
#Filename                                                                 #
outfile="testsuiteresults.txt"                                            #
memfile="memprofile.txt"                                                  #
numthreads=24                                                             #
###########################################################################

set -um
```

```

#Check input
[[ $# -gt 0 ]] || { sed -n '2,/^#$/ s/^# //p' <"$0"; exit 1; }

echo "Pattern_____________|__RSS__|_Dirty_|_Time_" > $outfile

run=0
time_avg=0
for entry in `ls $1`; do
    if [[ $@ == *"GraphPi"* ]]; then
        run="memusage.sh ${@: 2} $1/$entry"
    else
        run="memusage.sh ${@: 2} $1/$entry $numthreads"
    fi
    time=$(./$run | grep 'time' | grep -oP "[0-9]+.*"| sed "s/\s\+/ /g")
    time1=`echo $time | cut -d ' ' -f1`
    time2=`echo $time | cut -d ' ' -f2`
    time_avg=`echo "scale=5; $time1 / 2 + $time2 / 2" | bc`
    mem=$(cat memprofile.txt | grep 'Average' | grep -oP "[0-9]+.*"| tr '|' ' ' |
    rss=`echo $mem | cut -d ' ' -f1`
    dirty=`echo $mem | cut -d ' ' -f2`
    printf "%-20s|%+7s|%+7s|%+6s\n" $entry $rss $dirty $time_avg >> $outfile
    echo $entry done...
done
```