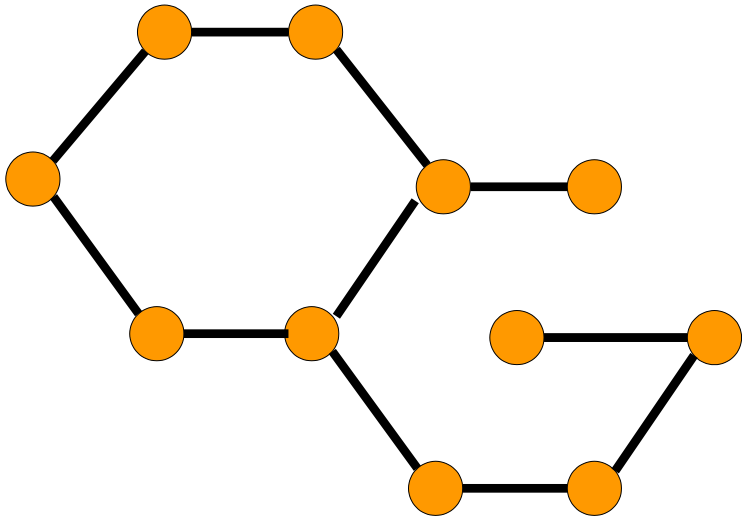

GBAD Manual

Version 1.0
Release 2.2

Table of Contents

Revision History

Release Version	Date	Revisions
1.0	December 15, 2009	Initial version of manual

Preface

Conventions

The following documentation conventions are followed within this document.

bold underlined text signifies notes or comments to the reader.

Italicized text signifies file names, directories or programs.

Bold italicized text signifies a reference to another document.

1. Introduction

The following document provides a manual on how to use the GBAD system.

1.1 Overview

The GBAD graph-based anomaly detection system discovers structural anomalies in data represented as a graph. The normative pattern discovery aspects of the GBAD system is based upon the SUBDUE graph-based pattern learning system. (Details of how SUBDUE works internally can be found on the SUBDUE web-site; details of the specific GBAD algorithms can be found on the GBAD web-site.)

This document will provide you with the specifics on how to install and run the GBAD application. This document contains the following sections:

- Chapter Two: instructions on how to download and install GBAD
- Chapter Three: layout of the required graph input file
- Chapter Four: instructions on running GBAD
- Chapter Five: GBAD Graphical User Interface
- Chapter Six: various notes and issues regarding the GBAD application
- Appendix A: terminology

1.2 Reference Documents

- ***GBAD Publications:*** www.gbad.info
- ***SUBDUE Home Page:*** http://www.subdue.org/
- ***AT&T Labs GraphViz:*** http://www.graphviz.org/

2. Downloading and Installing

In order to build and run the GBAD application, you must first download the appropriate files.

2.1 Download

The GBAD system, including documentation, papers, and research, can be found on the GBAD web-site page (www.gbad.info).

In order to get the latest copy of the application, you must choose the Download option located on the left-hand side of the GBAD home page. After clicking the Download link, you will be redirected to the "Download" page, which provides the requestor with latest source code for the GBAD application. After completing a brief informational form, you will be provided with an acknowledgement message on the web-page with a link to download the latest version of GBAD.

2.2 Files

Once you have downloaded the GBAD archive, and unzipped the files, the following directory/file structure is created:

./bin/ -- directory of executables (initially empty)

./COPYRIGHT -- file containing the GBAD copyright notice

./docs/ -- directory containing this manual

./graphs/ -- directory containing some sample graph input files

./README – file containing brief directions on how to build and run GBAD

./*RELEASE_NOTES* – file containing version histories

./src/ - directory containing the source code and make file

2.3 Install

After downloading and unzipping the files, you can now install the GBAD application. Installation consists of actually building the application so that it is now native to your Unix system.

GBAD uses the standard *make* facility to build its application. In order to build the application, you should perform the following steps:

1. Change directory to *gbad-<release>/src*
2. At the command prompt, enter: *make*. This will compile the GBAD programs.
3. At the command prompt, enter: *make install*. This will copy the executables to the *gbad-<release>/bin* directory
4. At the command prompt, enter: *make clean*. This will clean up the *src* directory (removing object files).

3. Data Format

The following section describes the format of the input graph that must be supplied in order to run the GBAD application.

3.1 Input

The input to the GBAD application is comprised of a textual representation of a graph.

3.1.1 Graph Format

The input file can consist of one or more graphs. Each graph is prefaced (on a line by itself) by an "*XP*", indicating a positive example. However, if the first (or only) graph in the file is positive, then the "*XP*" can be omitted.[1]

3.1.1.1 Vertices

Each graph is a sequence of vertices and edges. A vertex is defined as:

```
v <#> "<label>"
```

where <#> is a unique vertex ID for the graph and <*label*> is any string or real number. Strings containing white-space or the comment character (see below) must be surrounded by double-quotes. Vertex IDs for a graph must start at 1 and increase by 1 for each successive vertex.

It should also be noted that there must be at least one vertex defined before any edges are defined.

3.1.1.2 Edges

An edge is defined as one of the following:

e <vertex 1 #> <vertex 2 #> " <label> "

d <vertex 1 #> <vertex 2 #> " <label> "

u <vertex 1 #> <vertex 2 #> "<label> "

where <*vertex 1 #*> and <*vertex 2 #*> are the vertex ID's for the source vertex and the target vertex respectively, and <*label*> is any string or real number. Strings containing white-space or the comment character (see below) must be surrounded by double-quotes. Edges beginning with "*e*" are assumed directed unless the option "*-undirected*" is specified at the command line (see next section), in which case all "*e*" edges become undirected. Edges beginning with "*d*" are always directed, and edges beginning with "*u*" are always undirected.[2]

[1] Currently, only positive graphs are handled in GBAD.

[2] Currently, GBAD is expecting all labels to begin and end with quotation marks.

3.1.1.3 Comments

You can also choose to put comments in your graph input file. Comments are designated by the percent "*%*" sign. Anything after a "*%*" until the end of the line will be ignored (unless the "*%*" is part of a quoted label).

3.1.1.4 Example

As an example, if you were trying to represent that a *cat* is an *animal*, the graph might look like the following:

```
% Cat
v 1 cat
v 2 animal
d 1 2 is-a
```

However, if the edge were directed the other way (eg. d 2 1 is-a), that would imply that the animal is a cat, which is not necessarily true. It should be noted that GBAD would not complain if you made that relationship, but the results would probably not be what you desired.

3.2 Output

The output from executing GBAD, which will be discussed in more detail in the following section, consists of textual information that is essentially represented in the same format as the input. In addition to the patterns and anomalies that are discovered, the output includes options, parameters and other information about the run. By default, the output is displayed to the user's screen, or to wherever the user directed the output (for example, with the Unix ">" command).

3.3 Example

To illustrate what an input file might look like, **Figure 1** shows a segment of the GBAD input file for for this example, and **Figure 2** shows a segment of a pictorial representation of this example.

```
v 1 v10
v 2 v9
v 3 v8
v 4 v7
v 5 v6
v 6 v5
v 7 v4
v 8 v3
v 9 v2
v 10 v1
...
e 7 1 e9
e 7 2 e8
e 8 3 e7
e 8 4 e6
e 9 5 e5
e 9 6 e4
e 10 7 e3
e 10 8 e2
e 10 9 e1
...
```

Figure 1: Segment of GBAD input graph format for example

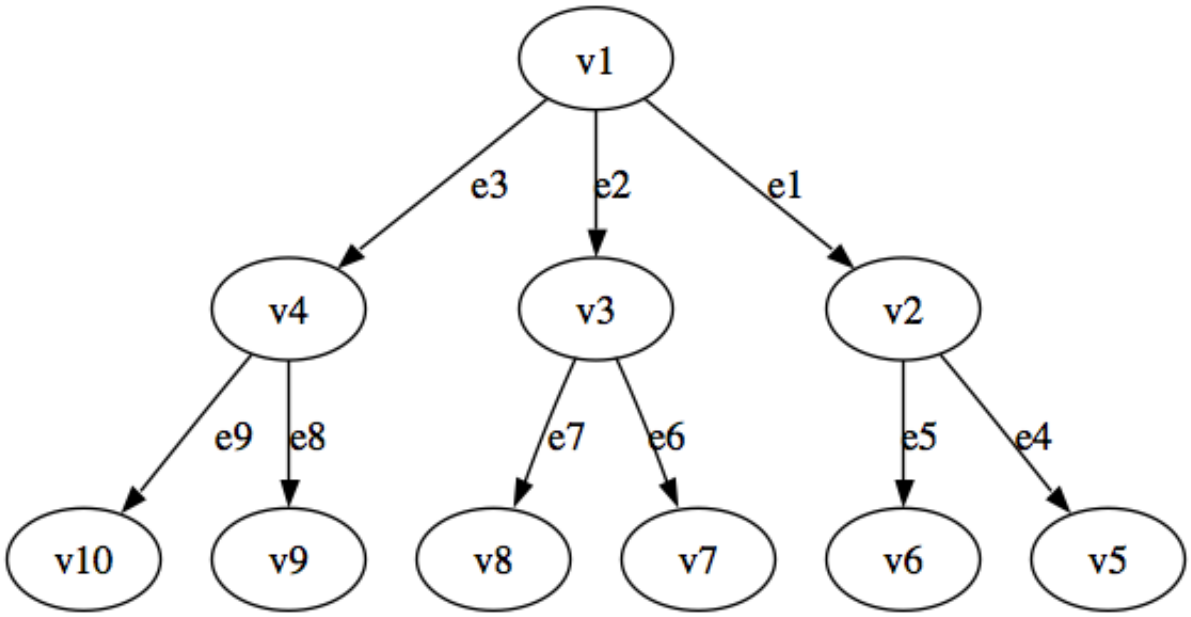

Figure 2: Segment of graphical representation for example

This graph example is also included in the GBAD kit in the file *graph_with_modification.g*.

4. Executing

The purpose of the GBAD application is to discover anomalous substructure instances (and their specific anomalous vertices/edges) in a specified input graph. The ability to discover these anomalies is controlled by various user-specified parameters, which control the algorithms that are used, as well as the length and size of the discovery space (among other things).

4.1 Command

GBAD has what is known as a "command-line" interface. In order to run GBAD, you must be logged on to the Unix machine where the application was downloaded and installed. From the Unix prompt, the command would be as follows:

```
gbad <options> <graph input file>
```

There are several points which should be noted here:

gbad is the name of the executable. The above example assumes that you are running the application from the same directory where the executable resides (which is probably in *./bin/*). If the desire is to run the application from another directory, *gbad* will have to be "pathed".

<*options*> will be discussed in the next section

<*graph input file*> is the name (and path) of the graph input file (ex. *graph_with_deletion.g*)

4.1.1 Options

Because of the nature of graphs, and the varying ways that graphs can be dissected and analyzed, there are several command-line options available to be used. Each of these options can result in different results when used together or by themselves.

This document will not go into graph theory, or even some of the more common algorithmic functions used in computers, and will leave that up to you to investigate. Each of the options will be explained, but it is assumed that there is some knowledge of the subject being discussed.

4.1.1.1 -beam <#>

This parameter specifies the beam width of GBAD's *normative pattern*[3] search. Only the best beam substructures (or all the substructures with the best beam values) are kept on the frontier of the search. The exact meaning of the beam width is determined by the *-valuebased* option described below. The default value for this setting is 4.

[3] There are basically two steps in GBAD's processing: (1) it discovers the normative substructure (or substructures) in a graph, and then (2) it applies user-specified anomaly detection algorithms based upon those normative patterns.

4.1.1.2 -dot <file_name>

This option causes GBAD to write a dot file, specified by the parameter, at the end of the GBAD run. The dot file will contain a description of the input graph in the dot language with vertices and edges color. The vertices and edges that are part of the normative pattern are colored blue. Any anomalies found by the GBAD-MPS algorithm are colored orange. The anomalous substructures found by the GBAD-MDL algorithm are colored orange and the actual anomaly within the substructure is colored red. Anomalies found by the GBAD-P algorithm are colored orange and red, where the red vertices are the most anomalous. Various programs are capable of rendering the dot file, for example, the *dot* program that comes with the *GraphViz* package.

4.1.1.3 -eval <#>

GBAD has three methods available for evaluating candidate (normative) substructures:

4.1.1.3.1 Minimum Description Length (MDL) - 1

The value of a substructure *S* in graph *G* is:

$$value(S,G) = \frac{DL(G)}{\left(DL(S) + DL(G \mid S)\right)}$$

where *DL* is the description length in bits, and (*G*|*S*) is *G* compressed with *S*.

MDL is the default evaluation method.

4.1.1.3.2 Size - 2

The value of a substructure *S* in graph *G* is:

$$value(S,G) = \frac{size(G)}{\left(size(S) + size(G \mid S)\right)}$$

where

$$size(G) = \left(\#vertices(G) + \#edges(G)\right)$$

and (*G*|*S*) is *G* compressed with *S*.

The size measure is faster to compute than the MDL measure, but less consistent.

4.1.1.4 -limit <#>

The number of different substructures to consider. The default value is computed based on the input graph as #Edges / 2.

4.1.1.5 -maxAnomalousScore <#.#>

This argument specifies the maximum anomalous score for reporting any potential anomalies.[4] If the best anomaly (i.e., the anomaly with the lowest score) has a score *greater* than this value, no anomaly will be reported. The default value for this setting is no limit on how high the score can be.

4.1.1.6 -maxsize <#>

This argument specifies the maximum number of vertices that can be in a reported substructure. Larger substructures are pruned from the search space. The default value for this setting is the number of vertices in the input graph.

4.1.1.7 -mdl <#.#>

This option initiates the GBAD-MDL minimum description length algorithm as part of the anomaly detection phase. The GBAD-MDL algorithm uses the Minimum Description Length (MDL) heuristic to discover the best substructure in a graph, and then subsequently examines all of the instances for similar patterns. Using an inexact matching approach, instances that are the "closest" (without matching exactly) in structure to the best structure (i.e., compresses the graph the most), where there is a tradeoff in the cost of transforming the instance to match the structure (matchcost), as well as the frequency with which the instance occurs, where the lower the value, the more anomalous the structure.

The parameter to the *–mdl* option is the maximum amount of *change* that the user is willing to accept. When specifying this option, you must give it a floating-point value between 0.0 and 1.0, where the closer the value to 0.0, the less change one is willing to accept as anomalous.

4.1.1.8 -minAnomalousScore <#.#>

This argument specifies the minimum anomalous score for reporting any potential anomalies.[5] If the best anomaly (i.e., the anomaly with the lowest score) has a score *lower* than this value, no anomaly will be reported. The default value for this setting is 0.0.

4.1.1.9 -minsize <#>

This argument specifies the minimum number of vertices that must be in a substructure before it is reported. The default value for this setting is 1.

4.1.1.10 -mps <#.#>

This option initiates the GBAD-MPS maximum partial substructure algorithm as part of the anomaly detection phase. The GBAD-MPS algorithm examines all instances of parent (or ancestral) substructures that are missing various edges and vertices. The value associated with the parent instances represents the cost of transformation (i.e., how much change would have to take place for the

[4] In GBAD, the lower the score, the more anomalous a substructure.

[5] In GBAD, the lower the score, the more anomalous a substructure.

instance to match the best substructure). Thus, the instance with the lowest cost transformation (if more than one instance have the same value, the frequency of the instance's structure will be used to break the tie if possible) is considered the anomaly, as it is closest (maximum) to the best substructure without being included on the normative substructure's instance list.

The parameter to the *–mps* option is the maximum amount of *change* that the user is willing to accept. When specifying this option, you must give it a floating-point value between 0.0 and 1.0, where the closer the value to 0.0, the less change one is willing to accept as anomalous.

4.1.1.11 -norm <#>

This argument specifies the normative pattern to use when applying the anomaly detection algorithms. In the case of the *–prob* algorithm, the normative pattern specified is only used for the first iteration. The default value for this setting is the best substructure, or 1.

4.1.1.12 -nsubs <#>

This argument specifies the maximum length of the list of best substructures found during the discovery. The default value for this setting is 3.

4.1.1.13 -output #

This argument controls the amount of GBAD's screen output. Valid values are:

(1) Print best substructure found at each iteration.

(2) Print *-nsubs* best substructures. (This is the default value.)

(3) Same as (2), plus prints the instances of the best substructures.

(4) Same as (3), plus prints substructure countdown and the best substructure found so far.

(5) Same as (4), plus prints each substructure considered.

4.1.1.14 -overlap

GBAD normally will not allow overlap among the instances of a substructure. Specifying this argument will allow overlap. During graph compression an *OVERLAP_<iteration>* edge is added between each pair of overlapping instances, and external edges to shared vertices are duplicated to all instances sharing the vertex. Allowing overlap slows GBAD considerably.

4.1.1.15 -prob <#>

This option initiates the GBAD-P probabilistic algorithm as part of the anomaly detection phase. The GBAD-P algorithm examines all extensions to the normative substructure with the lowest probability, and hence the most likely to be an anomaly. The GBAD-P functionality examines the probability of extensions to the normative pattern to determine if there is an instance that includes edges and vertices that are probabilistically less than other possible extensions.

The parameter to the *–prob* option is the number of iterations made over the input graph in which first, the best substructure from the previous iteration is used to compress the graph for use in the next iteration, then for subsequent iterations (if a value greater than 2 was specified), the previously discovered anomalous instance(s) are compressed for use in the next iteration, and so on. When specifying this option, you must give it an integer value, and in order to discover any anomalies with the GBAD-P algorithm, a minimum value of 2 must be specified.

4.1.1.16 -prune

This option tells GBAD to prune the search space by discarding substructures whose value is less than that of their parent's substructure. Since the evaluation heuristics are not monotonic, pruning may cause GBAD to miss some good normative substructures. However, it will improve the running time. The default is no pruning.

4.1.1.17 -undirected

GBAD assumes that edges in the input graph file defined using `e' are directed edges. Specifying this argument makes these edges undirected. Note that graph file edges defined with `u' are always undirected, and edges defined with `d' are always directed.

4.1.1.18 -valuebased

Normally, GBAD's beam width implies that only the beam best substructures are kept on the frontier of the search. If the *-valuebased* option is given, then the beam width is interpreted as keeping all the substructures with the top beam values on the frontier of the search.

4.2 Example

Continuing with our example graph from the previous chapter, it is now time to run GBAD. In this example, we will run just the GBAD-MDL algorithm:

```
bin/gbad -mdl 0.1 graphs/graph_with_modification.g
```

Figure 3 shows a portion of the actual textual output of the GBAD run on this example graph. **Figure 5** shows a graphical representation of the normative pattern. **Figure 5** shows a graphical representation of the anomalous pattern, with the specific anomalous vertex.

```
 GBAD 2.1

Parameters:
  Input file..................... graphs/graph_with_modification.g
  Predefined substructure file... none
  Output file.................... none
  Beam width..................... 4
  Compress....................... false
  Evaluation method.............. MDL
  Anomaly Detection method....... Information Theoretic
  Information Theoretic threshold 0.100000
  Max Anomalous Score............ MAX
  Normative Pattern.............. 1
  Similarity Percentage.......... 1.000000
  'e' edges directed............. true
  Iterations..................... 1
  Limit.......................... 109
  Minimum size of substructures.. 1
  Maximum size of substructures.. 220
  Number of best substructures... 3
  Output level................... 2
  Allow overlapping instances.... false
  Prune.......................... false
  Threshold...................... 0.000000
  Value-based queue.............. false

Read 1 total positive graphs

1 positive graphs: 220 vertices, 219 edges, 4166 bits

...

Anomalous Instance(s):

 from example 1:
    v 211 v10
    v 212 v9
    v 213 v8
    v 214 v7
    v 215 v6
    v 216 v5
    v 217 v4
    v 218 v3
    v 219 v2
    v 220 v9 <-- anomaly (original vertex: 220 , in original example 1)
    d 217 211 e9
    d 217 212 e8
    d 218 213 e7
    d 218 214 e6
...
```

Figure 3: Actual partial textual output of the GBAD run on the sample graph.

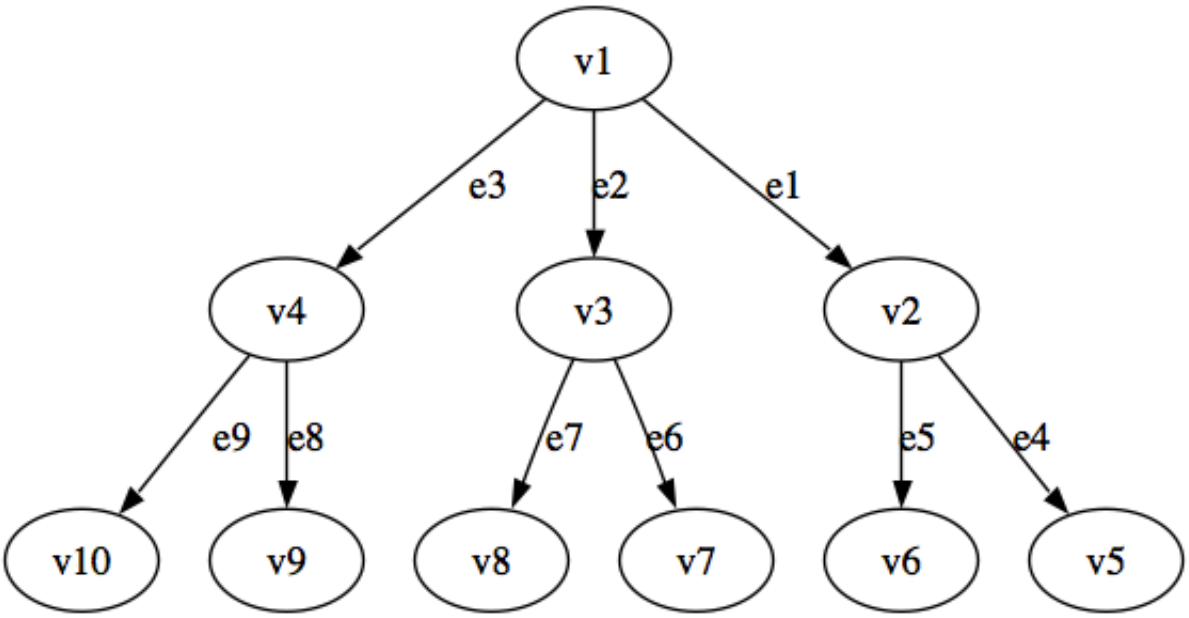

Figure 4: Graphical representation of the normative pattern.

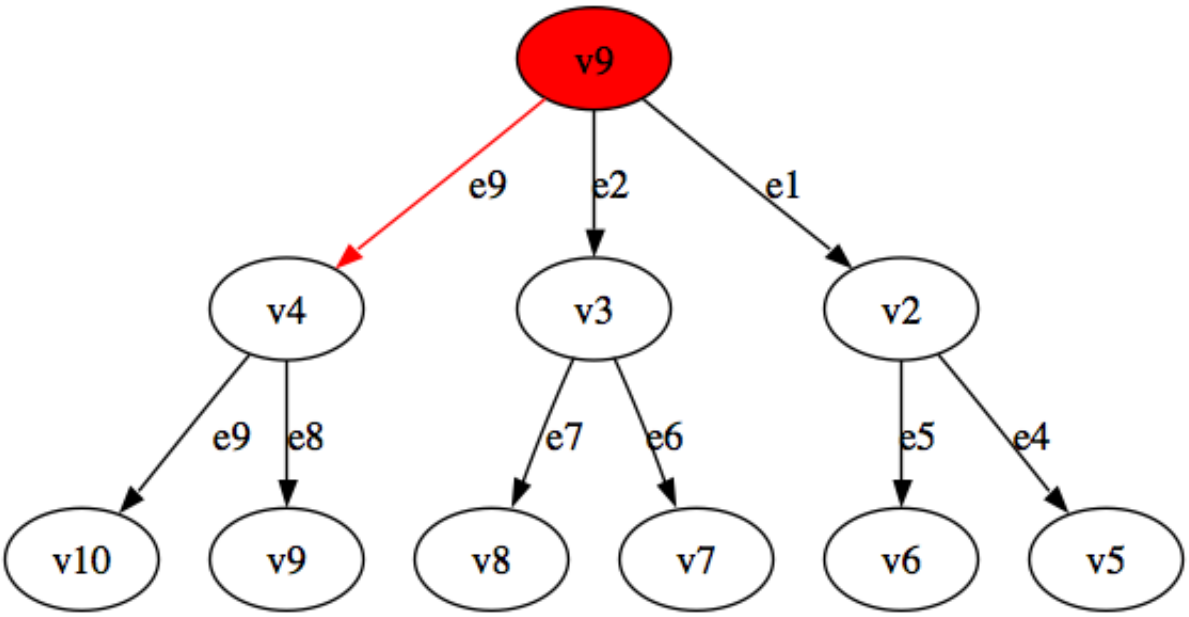

Figure 5: Graphical representation of the anomalous substructure (with specific anomaly in red) discovered in the example.

The first part of GBAD's output indicates the parameter settings for this run. These parameters were mentioned previously, but we discuss some of them in more detail here. All of the parameters are set to their default values. GBAD's default *evaluation* method is based on the minimum description length (MDL) principle, which essentially says that the best pattern (or substructure) is the one that best trades off the size of the pattern and the size of the input graph after compressing away all the instances of the pattern.

The *limit* parameter controls the extent of GBAD's search by limiting the number of different substructures the SUBDUE engine considers for expansion, i.e., it is an upper bound on the portion of the search space considered by GBAD. The *limit* defaults to half the number of edges in the positive graphs. This default value tends to be higher than necessary, as the SUBDUE discovery engine typically finds the best substructure early on. After experience with running GBAD in a particular domain, the *limit* parameter can be decreased to a value closer to when GBAD actually finds the best substructure, which can be determined by setting the *output level* to 5 so that GBAD outputs whenever it finds a new best substructure.

The *evaluation method* and *limit* parameters allow significant control over the efficiency and effectiveness of GBAD. Other parameters (*beam*, *iterations*, *prune*, *valuebased*) exert additional control over the amount of search, while still others (*overlap*) allow the introduction of additional capabilities (see previous section on ***Options***, and subsequent examples).

In this simple example, we see that the input graph has 1 (positive) graph. The total number of vertices and edges are given, along with the description length in bits of this graph, according to the MDL encoding used by the SUBDUE engine.

At this point, the SUBDUE engine begins its search for the substructure maximizing the chosen evaluation method. After determining the normative substructures, GBAD applies whatever anomaly detection algorithms were specified by the user – in this case, the *–mdl* option was chosen. After completing its discovery process, GBAD returns the instance of the most anomalous substructure (or substructures), followed by the list of normative substructures, ordered from best to worse (omitted from this figure due to space).

While not shown in this example, at the bottom of the output file, GBAD indicates the amount of CPU time spent processing the graph for anomalies (in this example, around 1 second). As discussed above, the running time of GBAD depends on a number of parameters. For extremely large graphs, on the order of a 100K vertices, GBAD's running time can be quite long, sometimes on the order of days. However, these long running times can be addressed by tweaking parameters, or possibly redesigning the graph representation to remove information not relevant to the learning task.

4.3 More Examples

The following sections show results when running the other anomaly detection algorithms, as well as when we run all of the algorithms simultaneously.

4.3.1 Probabilistic Example

The following example graph contains several disconnected subgraphs. Every edge in the graph is given a label of 'e.' Three of the subgraphs are the complete graph on 3 vertices with the vertices labeled 1 though 3. Four of the subgraphs are the complete graph on 4 vertices with the vertices labeled 1 though 4. The graph also contains three additional subgraphs that are complete graphs on 4 vertices with extensions attached. The vertices on these subgraphs are labeled 1 though 3 and what would have been labeled 4 was changed to 'V.' This 'V' vertex is connected to a vertex labeled '5.' Two of the vertices labeled with a '5' are connected to a vertex labeled '6' while the other one is connected to a vertex labeled 'v.' This last vertex labeled with the lowercase 'v' is the target anomaly to be detected. In order to find this vertex the probabilistic algorithm is used.

4.3.1.1 Input

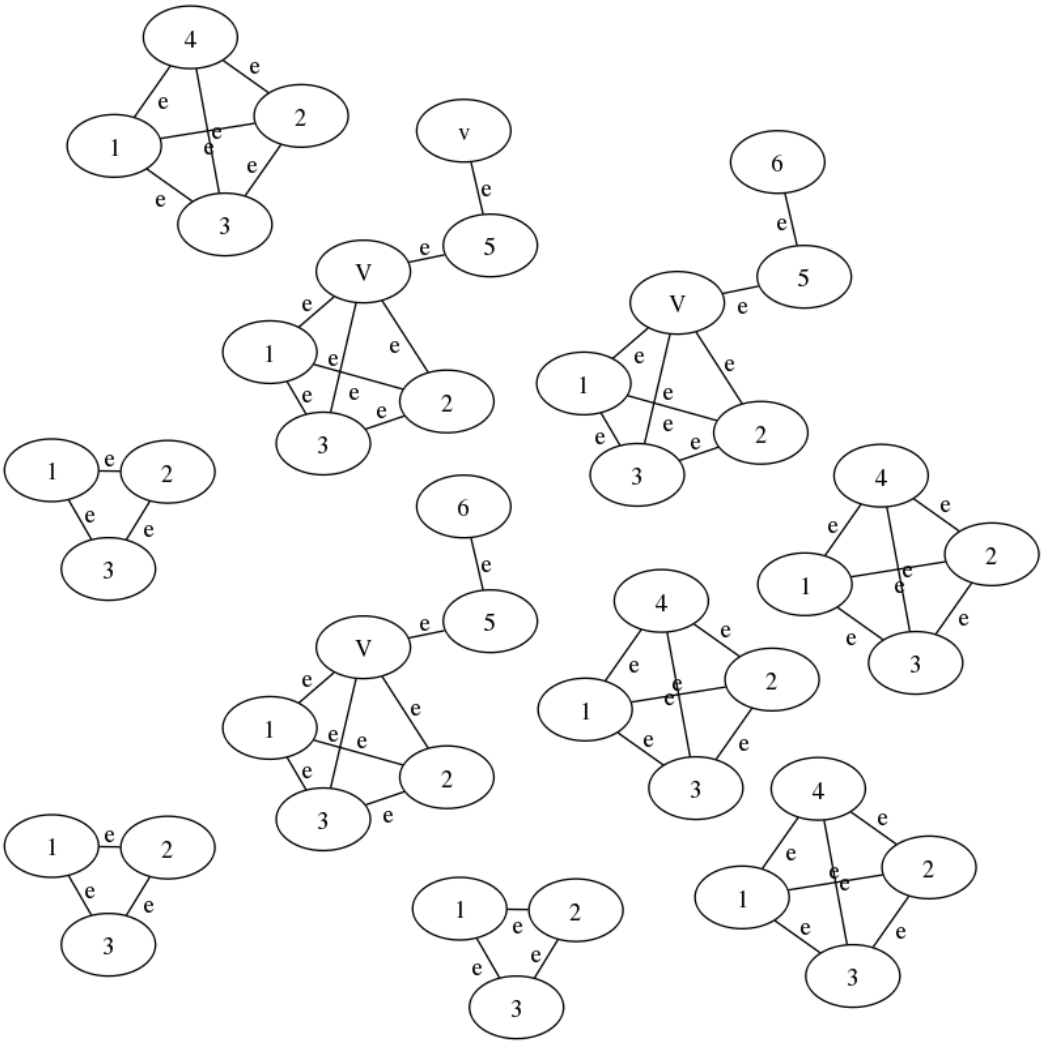

Figure 6: Graphical picture of graph input file containing an anomalous insertion.

```
...
XP
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "4"
u 1 2 "e"
u 1 3 "e"
u 1 4 "e"
u 2 3 "e"
u 2 4 "e"
u 3 4 "e"
...
XP
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "V"
v 5 "5"
v 6 "6"
u 1 2 "e"
u 1 3 "e"
u 1 4 "e"
u 2 3 "e"
u 2 4 "e"
u 3 4 "e"
...
```

Figure 7: GBAD input graph file for prob.g.

This graph example is also included in the GBAD kit in the file *prob.g*.

4.3.1.2 Execution

Now that the data is in the proper format, we can run GBAD using the following command (assuming the graph file is located where you are running the application):

```
bin/gbad -prob 6 prob.g
```

Figure 8 shows a graphical depiction of the normative substructure discovered by GBAD after the first iteration. **Figure 9** shows the textual output of GBAD after the second iteration. **Figure 10** shows the graphical depiction of the anomaly detected by the probabilistic algorithm.

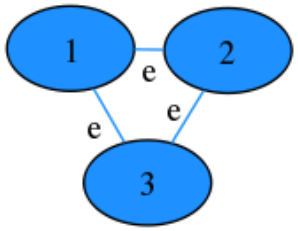

Figure 8: Graphical depiction of normative substructure discovered by GBAD after the first iteration.

```
...
----- Iteration 2 -----

10 positive graphs: 23 vertices, 27 edges, 277 bits
7 unique labels

5 initial substructures
Normative Pattern:
Substructure: value = 1.12182, instances = 21
  Graph(2v,1e):
    v 1 SUB_1
    v 2 "4"
    u 1 2 "e"

Anomalous Instance(s):

 from positive example 3:
    v 8 SUB_1
    v 15 "V" <-- anomaly (original vertex: 4 , in original example 8)
    u 8 15 "e" <-- anomaly (original edge vertices: 1 -- 4, in original
example 8)
    (probabilistic anomalous value = 0.900000 )

 from positive example 3:
    v 8 SUB_1
    v 15 "V" <-- anomaly (original vertex: 4 , in original example 8)
    u 8 15 "e" <-- anomaly (original edge vertices: 2 -- 4, in original
example 8)
    (probabilistic anomalous value = 0.900000 )

 from positive example 3:
    v 8 SUB_1
    v 15 "V" <-- anomaly (original vertex: 4 , in original example 8)
    u 8 15 "e" <-- anomaly (original edge vertices: 3 -- 4, in original
example 8)
...
```

Figure 9: Actual partial textual output of the GBAD-P run on the sample graph after the second iteration.

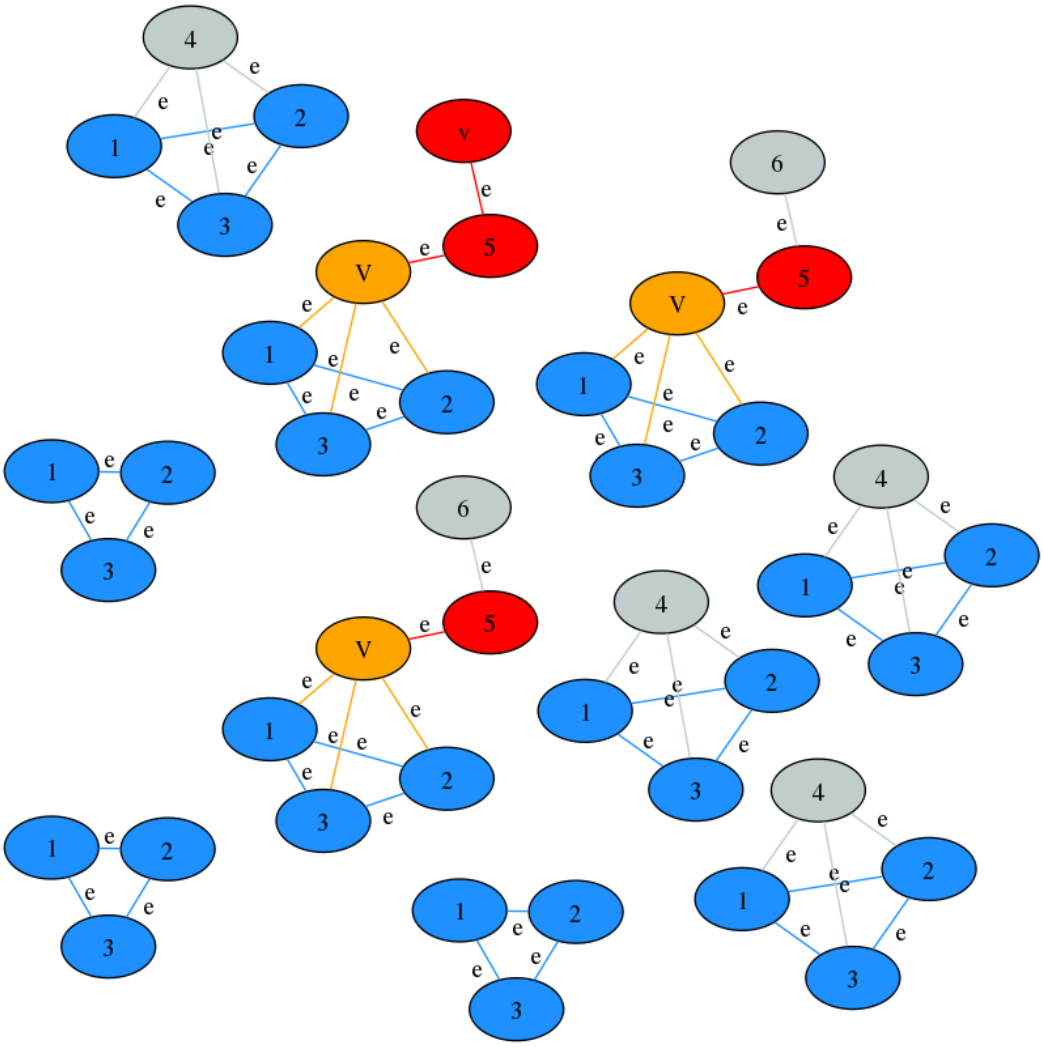

Figure 10: Graphical depiction of the normative pattern and anomaly reported by GBAD-P.

4.3.1.3 Analysis

In this run of GBAD, the complete graph of 3 vertices was found to be the normative pattern. In the second iteration, the probabilistic algorithm identifies the 'V' vertex extension off the normative pattern as having the lowest probability of being present. In the third iteration it determines that the only extension from the 'V' vertex is the '5' vertex and chooses it to be the most anomalous. In the fourth iteration, the algorithm determines that the lowercase 'v' vertex has the lowest probability of being present.

4.3.2 MDL Example

The following example graph contains several disconnected subgraphs. Four of the subgraphs are the complete graph of 4 vertices and three of the subgraphs are the complete graph of 3 vertices. All of the vertices have labels ranging from 1 to 4. All of the edges are labeled 'e.' There exists one vertex in the graph labeled 'V' and one edge in the graph labeled 'edge.' These are the two target anomalies to be detected. In order to find these modified labels, the minimum description length algorithm is used.

4.3.2.1 Input

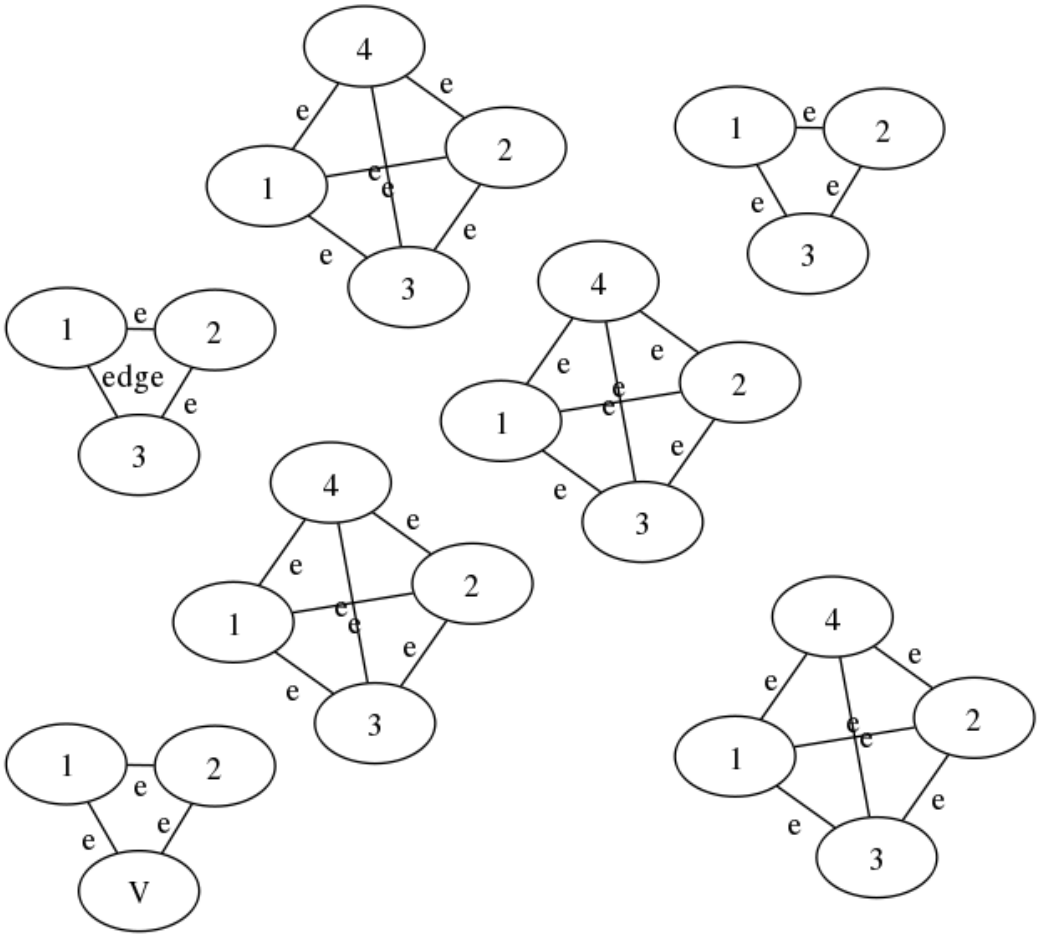

Figure 11: Graphical picture of a graph input file containing an anomalous modification.

```
...
XP
v 1 "1"
v 2 "2"
v 3 "V"
u 1 2 "e"
u 1 3 "e"
u 2 3 "e"
XP
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "e"
u 2 3 "e"
XP
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "edge"
u 2 3 "e"
...
```

Figure 12: GBAD input graph file for mdl.g.

This graph example is also included in the GBAD kit in the file *mdl.g*.

4.3.2.2 Execution

Now that the data is in the proper format, we can run GBAD using the following command (assuming the graph file is located where you are running the application):

```
bin/gbad -mdl 0.3 mdl.g
```

Figure 13 shows a graphical depiction of the normative substructure discovered by GBAD. **Figure 14** shows the textual output of GBAD. **Figure 15** shows the graphical depiction of the anomaly detected by the minimum description length algorithm.

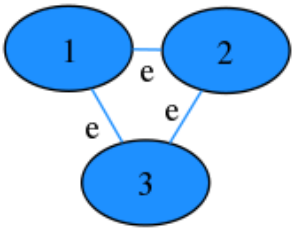

Figure 13: Graphical depiction of normative substructure discovered by GBAD.

```
...
Read 7 total positive graphs

7 positive graphs: 25 vertices, 33 edges, 387 bits
7 unique labels

5 initial substructures
Normative Pattern (1):
Substructure: value = 1.54572, instances = 5
  Graph(3v,3e):
    v 1 "1"
    v 2 "2"
    v 3 "3"
    u 1 2 "e"
    u 1 3 "e"
    u 2 3 "e"

Anomalous Instance(s):

 from example 1:
    v 1 "1"
    v 2 "2"
    v 3 "V" <-- anomaly (original vertex: 3 , in original example 1)
    u 1 2 "e"
    u 1 3 "e"
    u 2 3 "e"
    (information_theoretic anomalous value = 1.000000 )

 from example 3:
    v 7 "1"
    v 8 "2"
    v 9 "3"
    u 7 8 "e"
    u 7 9 "edge" <-- anomaly (original edge vertices: 1 -- 3, in original
example 3)
    u 8 9 "e"
...
```

Figure 14: Actual partial textual output of the GBAD-MDL run on the sample graph.

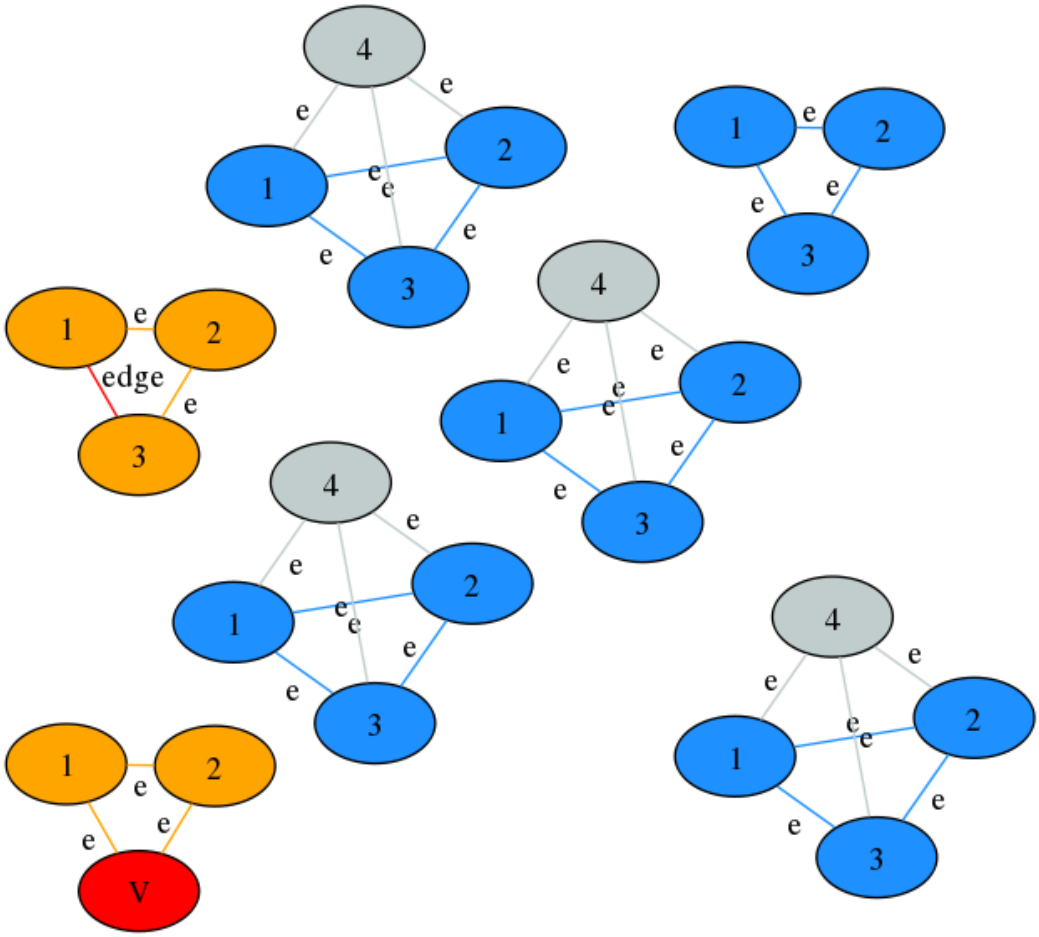

Figure 15: Graphical depiction of the normative pattern and anomaly reported by GBAD-MDL.

4.3.2.3 Analysis

In this run of GBAD, the complete graph of 3 vertices was found to be the normative pattern. The MDL algorithm found the subgraph with the vertex labeled 'V' and the subgraph with the edge labeled 'edge' required the fewest number of changes to transform these two structures into the normative pattern. These transformations both required a single label change, resulting in an anomalous value of 1.0 as seen in **Figure 14.**

4.3.3 Maximum Partial Substructure Example

The following example graph contains several disconnected subgraphs. Four of the subgraphs are the complete graph of 4 vertices and two of the subgraphs are the complete graph of 3 vertices. All of the vertices have labels ranging from 1 to 4. All of the edges are labeled 'e.' There is also another subgraph in the graph that consists of 3 vertices and 2 edges. If an edge were inserted between the vertex labeled '2' and the vertex labeled '3' the structure would be the complete graph on 3 vertices. This missing edge is the target anomaly to be detected. In order to find this missing edge, the maximum partial substructure algorithm is used.

4.3.3.1 Input

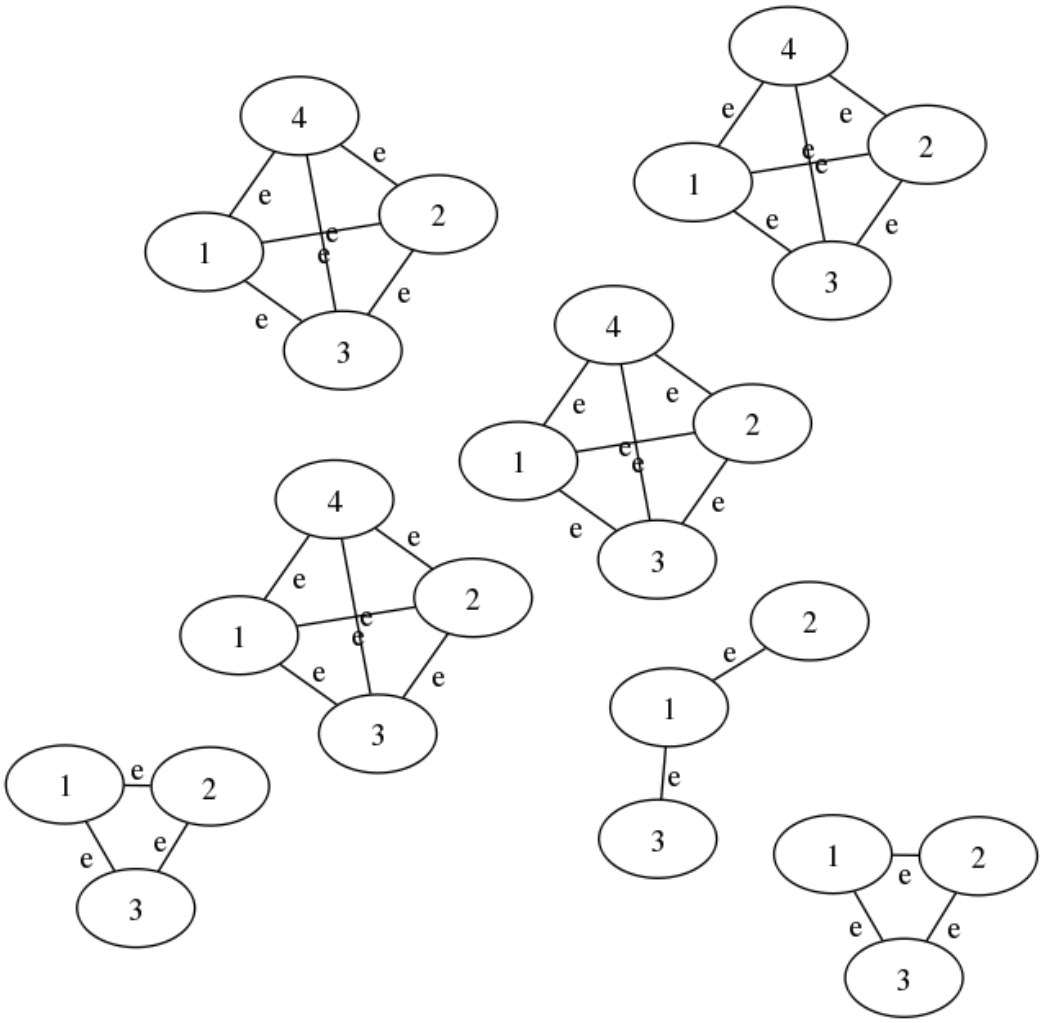

Figure 16: Graphical picture of graph input file containing anomalous deletion.

```
...
XP
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "e"
u 2 3 "e"
XP
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "e"
u 2 3 "e"
XP
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "e"
...
```

Figure 17: GBAD input graph file mps.g.

This graph example is also included in the GBAD kit in the file *mps.g*.

4.3.3.2 Execution

Now that the data is in the proper format, we can run GBAD using the following command (assuming the graph file is located where you are running the application):

```
bin/gbad -mps 0.3 mps.g
```

Figure 18 shows a graphical depiction of the normative substructure discovered by GBAD. **Figure 19** shows the textual output of GBAD. **Figure 20** shows the graphical depiction of the anomaly detected by the maximum partial substructure algorithm.

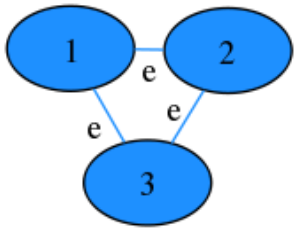

Figure 18: Graphical depiction of normative substructure discovered by GBAD.

```
...
Read 7 total positive graphs

7 positive graphs: 25 vertices, 32 edges, 351 bits
5 unique labels

5 initial substructures
Normative Pattern (1):
Substructure: value = 1.54572, instances = 5
  Graph(3v,3e):
    v 1 "1"
    v 2 "2"
    v 3 "3"
    u 1 2 "e"
    u 1 3 "e"
    u 2 3 "e"


Anomalous Instance(s):

 from example 6:
    v 12 "1"
    v 13 "2"
    v 14 "3"
    u 12 13 "e"
    u 12 14 "e"
    (max_partial_substructure anomalous value = 1.000000 )
...
```

Figure 19: Actual partial textual output of the GBAD-MPS run on the sample graph.

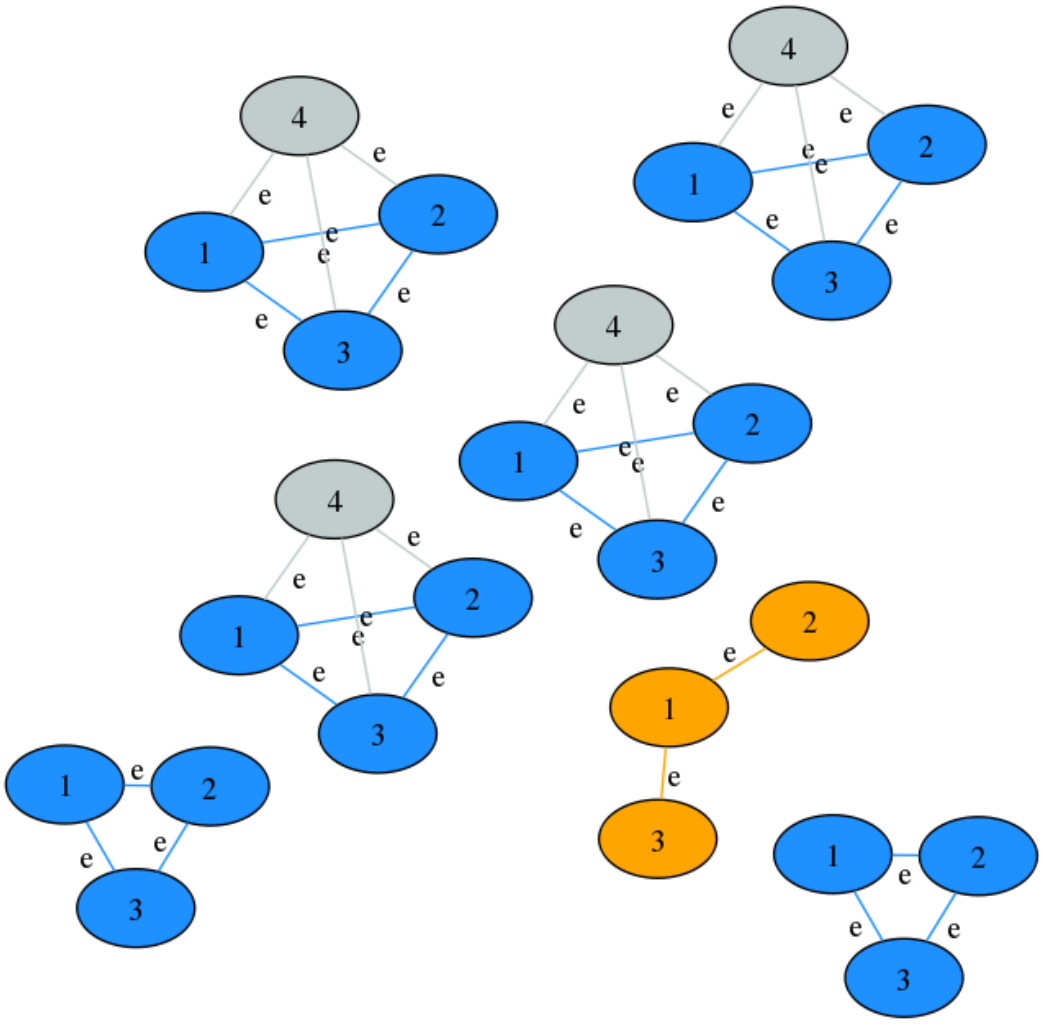

Figure 20: Graphical depiction of the normative pattern and anomaly reported by GBAD-MPS.

4.3.3.3 Analysis

In this run of GBAD, the complete graph of 3 vertices was found to be the normative pattern. The maximum partial substructure algorithm found that the subgraph with 3 vertices and only 2 edges was the structure that required the fewest number of vertex/edge insertions to transform it into the normative pattern. In this case only one edge insertion was needed, resulting in an anomalous value as shown in **Figure 19**.

5. GBAD Graphical Interface

The GBAD kit includes a Java graphical user interface that can be used to start GBAD runs and view results. The interface is provided as a Java Jar file (GUI.jar) and requires Java 1.5 or higher to run.

The Java interface recognizes ".g" extensions as graph input files and creates GBAD results files using ".result" extensions. File associations can be set through the GBAD GUI (see below) allowing these files to automatically be opened using the Java interface.

In order to make use of the basic Java interface functionally, the gbad executable must already be installed. To make use of the graph visualization functionally in the Java interface, GBAD's graph2dot executable and GraphViz's circo, dot, neato, twopi, fdp, and sfdp executables need to be installed as well (or just the ones you wish to use). If these executables are not in the system path they must be added to the users preferences (see Setting Preferences).

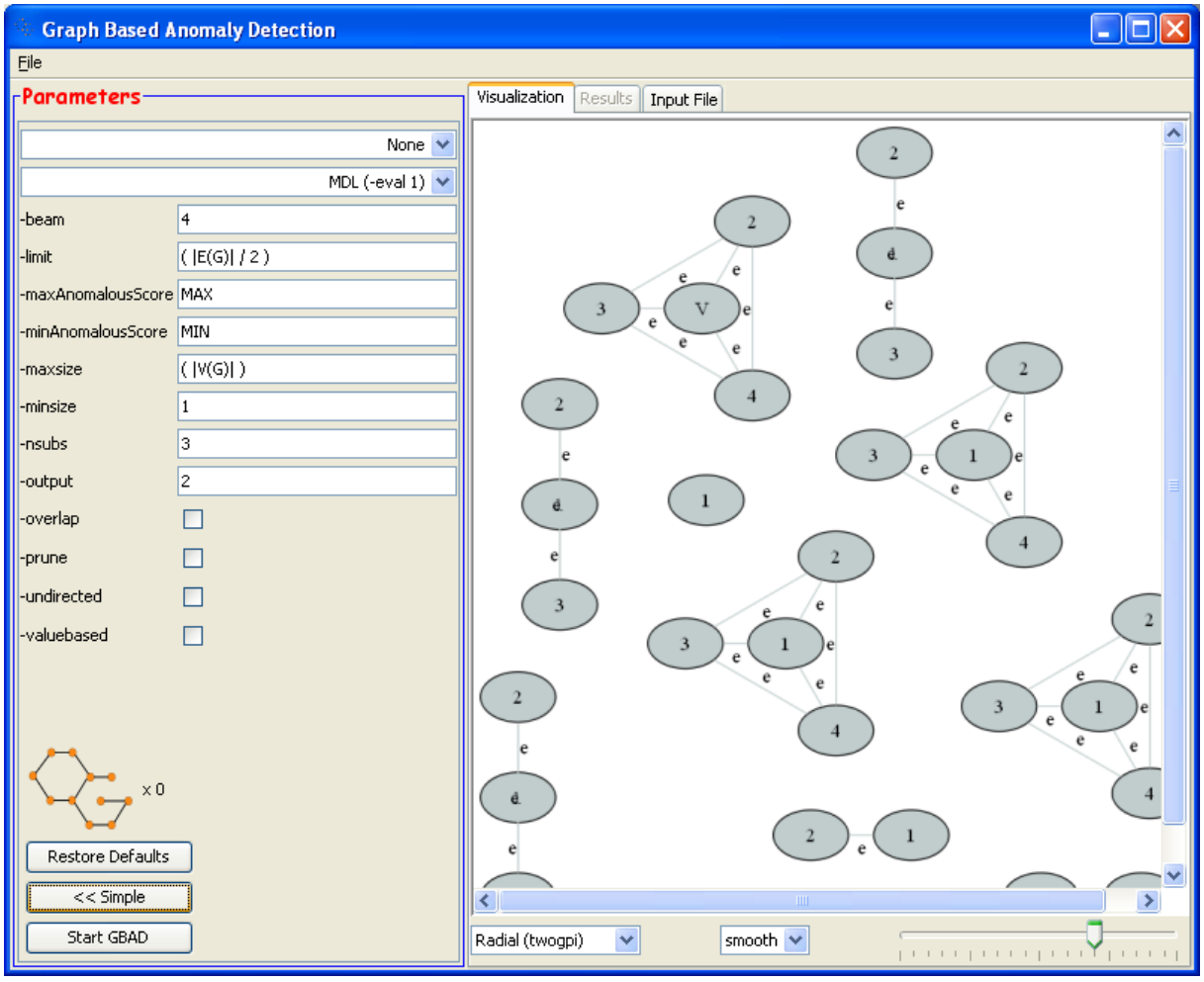

Figure 21: GBAD Graphical User Interface.

5.1 Overview

The Java interface is broken down into two main panels. The panel on the left is the "Parameter Selection" panel which is used to control all aspects of GBAD. The panel on the right is the "Graph Information" panel which is used to display all visual and textual information about the current selected graph.

5.2 Opening A Graph

To open a graph file, select "Open Graph" from the "File" menu and select the desired graph input file. After selecting the input file, a visualization of the graph file will be created and displayed in the "Visualization" tab in the "Graph Information" panel. The textual contents of the graph input file can also be viewed under the "Input File" tab.

5.3 Setting Parameters

By default, a simplified view of the GBAD parameters is available.

5.3.1 Algorithms

The GBAD anomaly detection algorithm is selected by choosing the desired algorithm from the top-most drop-down menu in the "Parameter Selection" panel. If "None" is selected then no algorithm will be run and only normative pattern discovery will be preformed.

5.3.2 Additional Parameters

All of the additional parameters available in GBAD (beam, limit, maxsize, minsize, etc.) are accessed by clicking the "Advanced >> " button towards the bottom of the "Parameter Selection" panel. Note that if the parameter view is changed back to simple (by clicking the "Simple <<" button) the values in these additional parameters are not reset and will be used when starting GBAD. The default values for all of the parameters can be reset by clicking the "Restore Defaults" button.

5.4 Running GBAD

After opening a graph file and setting any desired parameters, GBAD can be run by clicking the "Start GBAD" button at the bottom of the "Parameter Selection" panel. After starting GBAD, a "GBAD Output" window will open. The top of this window will contain the command line statement that was used to start GBAD. As output is available from GBAD, it will also be displayed in this new window as well as be written to a file. The output file will be in the same directory as the input file and the name will have the form <input_file>.<anomaly detection algorithm>.<time>.results.

If the "GBAD Output" is closed, the GBAD program will continue to run in the background, but there is no way to reopen the window after it has been closed.

The number of instances of GBAD currently running in the background is displayed next to the GBAD Icon in the "Parameter Selection" panel of the GBAD Interface. If a GBAD process completes before the GBAD interface is closed, a notification will appear and ask the user if he or she wishes to view the results. If the GBAD interface is closed before any current GBAD processes finish, the process will continue to run in the background. Results of completed processes can be open at any time (see Setting Preferences).

5.5 Viewing Results

To view the results of a completed GBAD run, select "Open Results" from the "File" menu and select the desired results file. After selecting a results file, the contents of the file will be displayed in the "Results" tab in the "Graph Information" panel. A visualization of the results will also be available in the "Visualization" tab and the graph input file will be displayed in the "Input File" tab.

The locations of dot file displayed in the "Visualization" tab and the graph input file displayed in the "Input File" tab are read from the results file. If the graph input file used to obtain results or the dot output file produced by GBAD is moved or deleted, then their respective data can not be displayed. Since opening a results file also opens the corresponding graph input file (assuming it can be found), clicking the "Start GBAD" button in the "Parameter Selection" panel will run GBAD again using the current parameter settings.

5.6 Setting Preferences

User preferences can be viewed and modified by selecting "Preferences" from the "File" menu. All user preferences are stored in the ".GBAD" directory in a file called "GUI.properties." The ".GBAD" directory can be found in the user's home directory, and is created by the GBAD GUI if it does not exist (i.e., the first time).

User preferences specify the location of the GraphViz executables (used in creating the graph visualizations), the graph2dot executable, and the gbad executable. Double clicking the file location for each of the executables allows the user to select a different executable to use. Care should be taken when changing the executables used, as the GUI will just execute whatever command that it is given. For example, if the GBAD executable is set to the UNIX *rm* executable, the result will be the deletion of the input graph when attempting to run GBAD.

In order for any preference modifications to be saved, the "Save" button in the lower left hand corner of the "Preferences" window must be clicked. Closing the window or clicking "Cancel" will discard any unsaved preference changes.

5.7 Brief Tutorial

The following section provides a walk though on how to use the Java interface to run GBAD on the sample *mps.g* file and view results.

When the Java interface starts, it will look similar to the display shown in **Figure 22**.

To open the *mps.g* file, select "Open Graph" from the "File" menu and select the file from the ./graphs directory found in GBAD kit installation directory. After opening the file, a visualization of the *mps.g* file will appear in the "Visualization" tab. At this point, the Java interface will look similar to **Figure 23**.

The default visualization style used to render the graph is "Directed (dot)". Other styles can be selected to change the way the graph is rendered from the drop down menu at the bottom middle of the screen. For example, changing the style to "Spring (neato)" will result in the Java interface looking similar to **Figure 24**.

To run the MPS algorithm on the *mps.g* file, select MPS from the drop down box at the top of the Parameters window pane. The deviation value is set to 0.3 by default, which is enough to detect the anomaly present in the mps.g file. Click "Start GBAD" to run GBAD on the *mps.g* graph. A new window called "GBAD Output" should appear. The top of the window will contain the command line

arguments used when running the *gbad* executable. As output is available from GBAD, it will be displayed in this new window. Since the *mps.g* contains a small graph, GBAD should finish in a second or two. At this point the Java interface will look similar to **Figure 25**.

After viewing the GBAD output, close the "GBAD Output" window. If the "GBAD Output" window is closed before GBAD finishes, GBAD will continue to run in the background. After GBAD finishes or the output window is closed (which ever happens last), the main Java interface will pop up a notification that a GBAD job has finished. The notification will ask the user if they want to view the results now. The results can always be opened later by clicking on "File > Open Results" and selecting the results. For this example, click "yes" and the result file will be opened and displayed in the "Results" tab. The Java interface will look similar to **Figure 26** at this point.

Two view a visualization of the anomaly found, click the "Visualization" tab. The Java interface will now look similar to **Figure 27**.

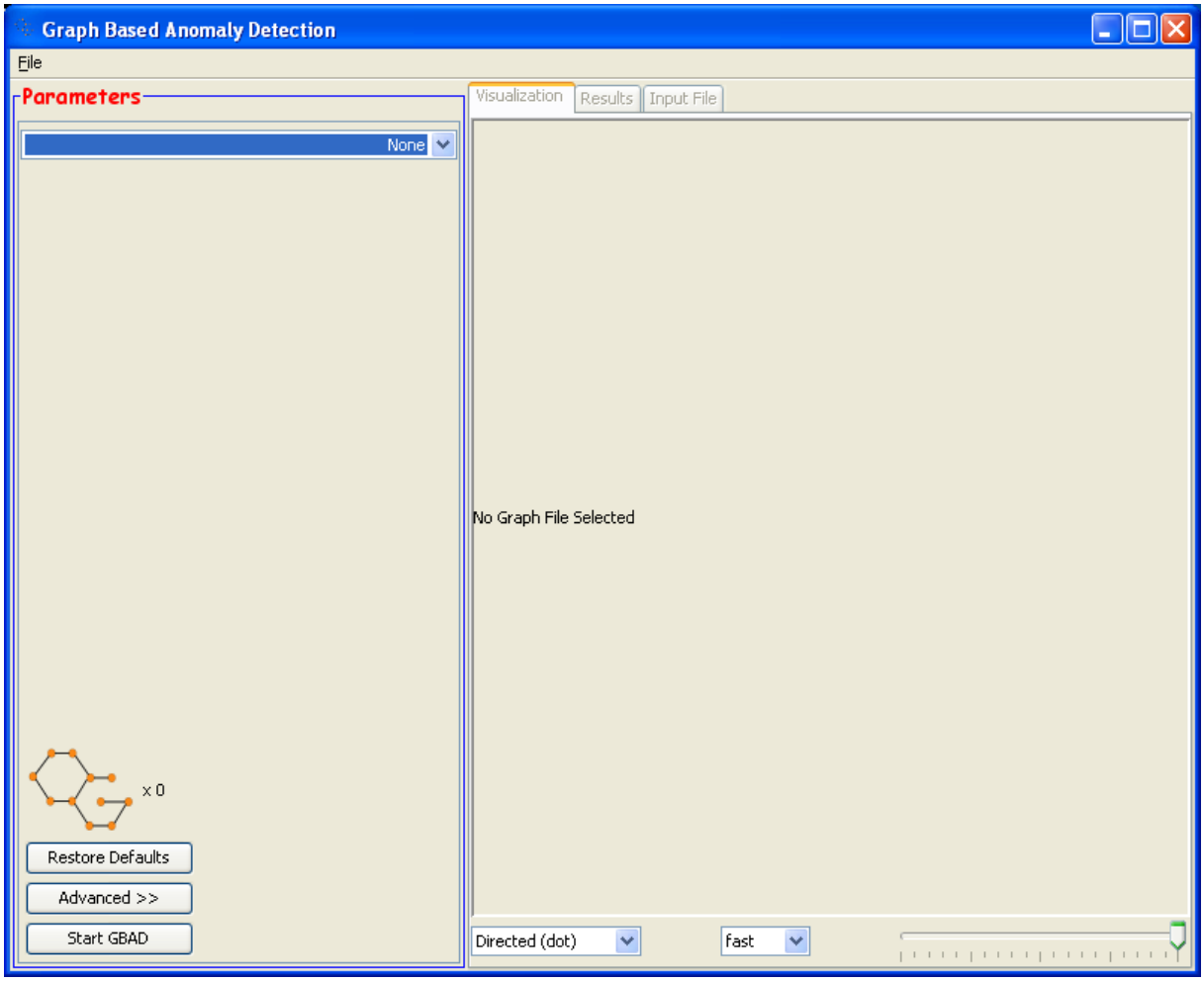

Figure 22: Initial Java interface view.

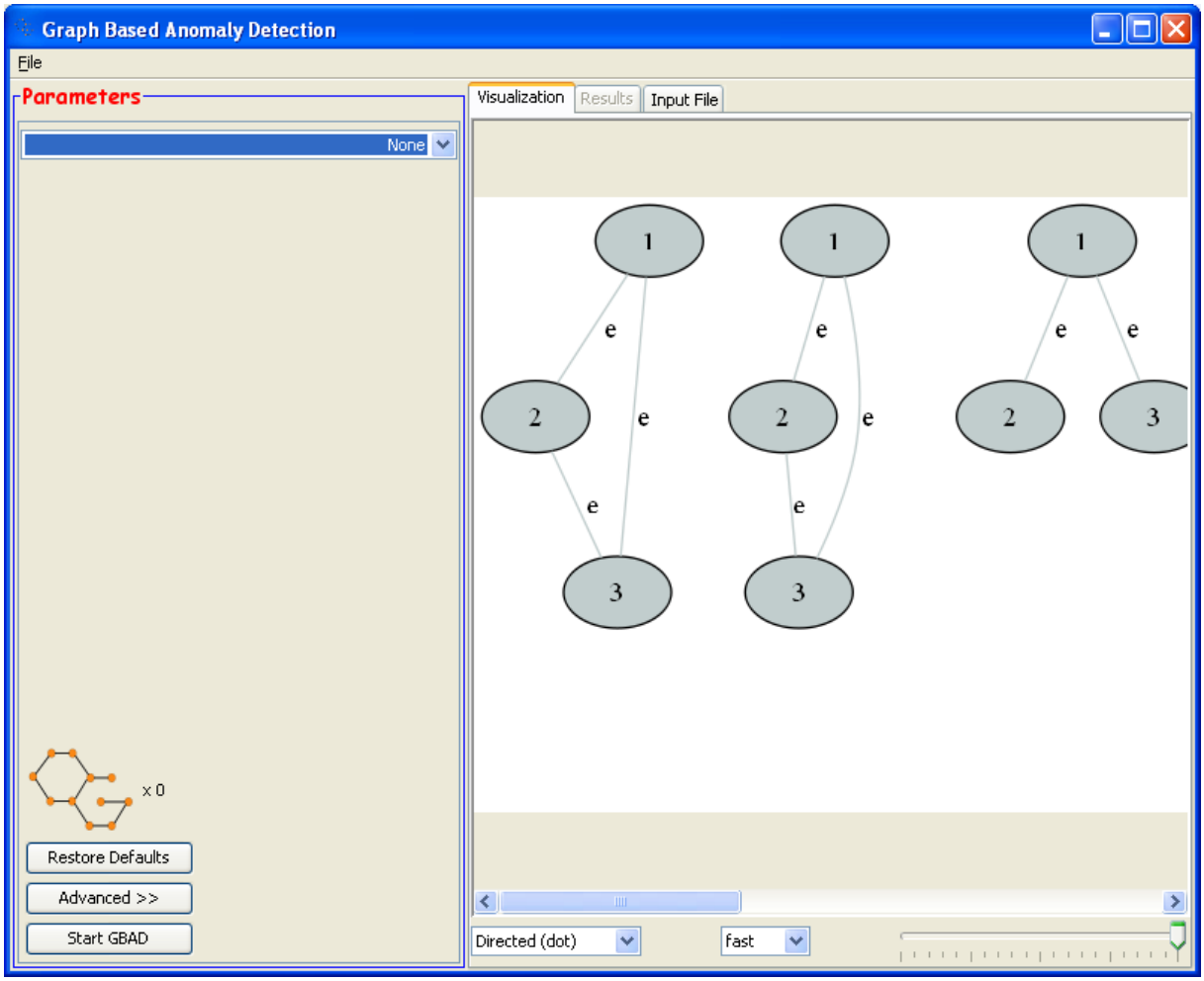

Figure 23: Graph Visualization of mps.g.

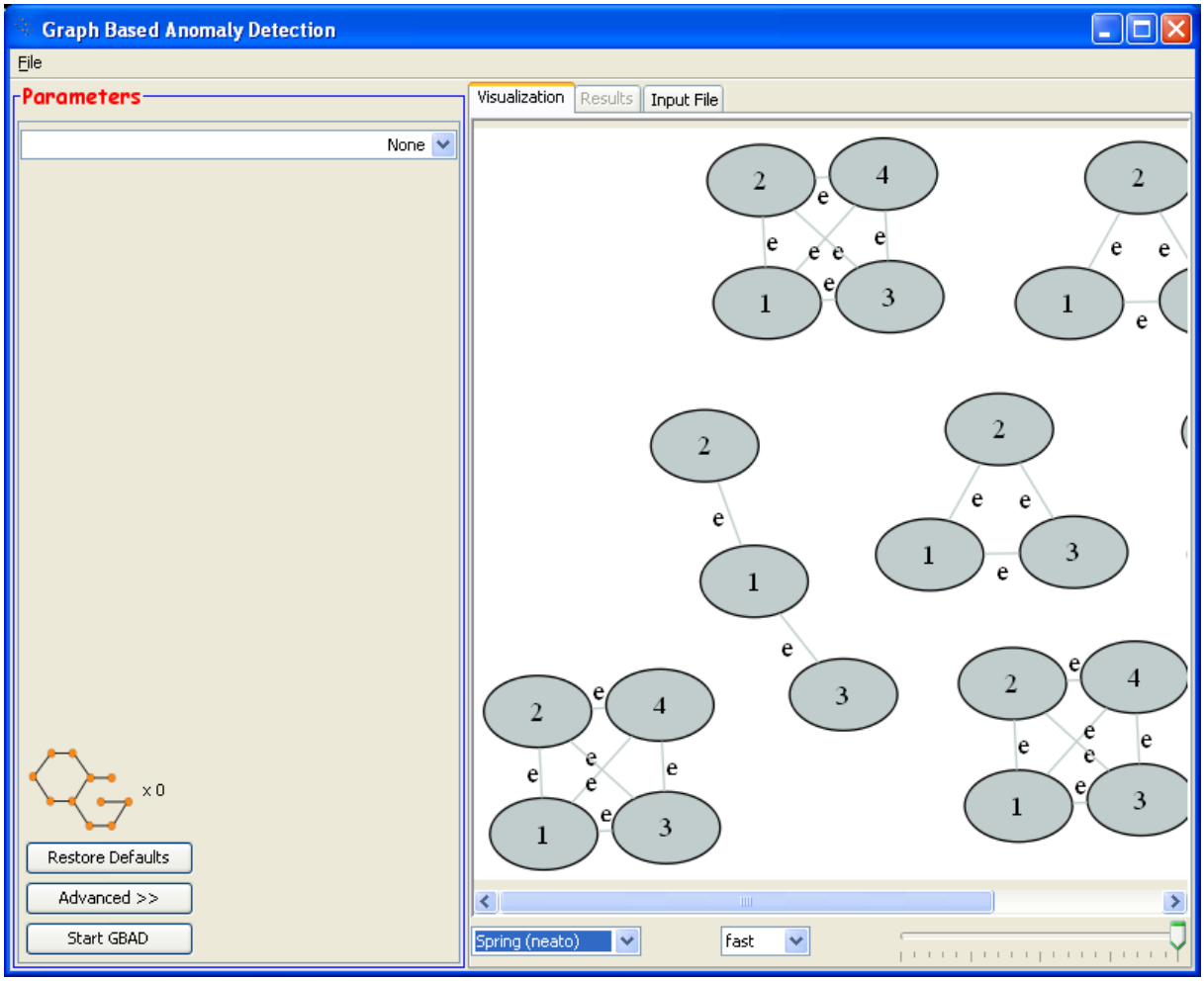

Figure 24: Graph Visualization of mps.g Using A Spring Layout.

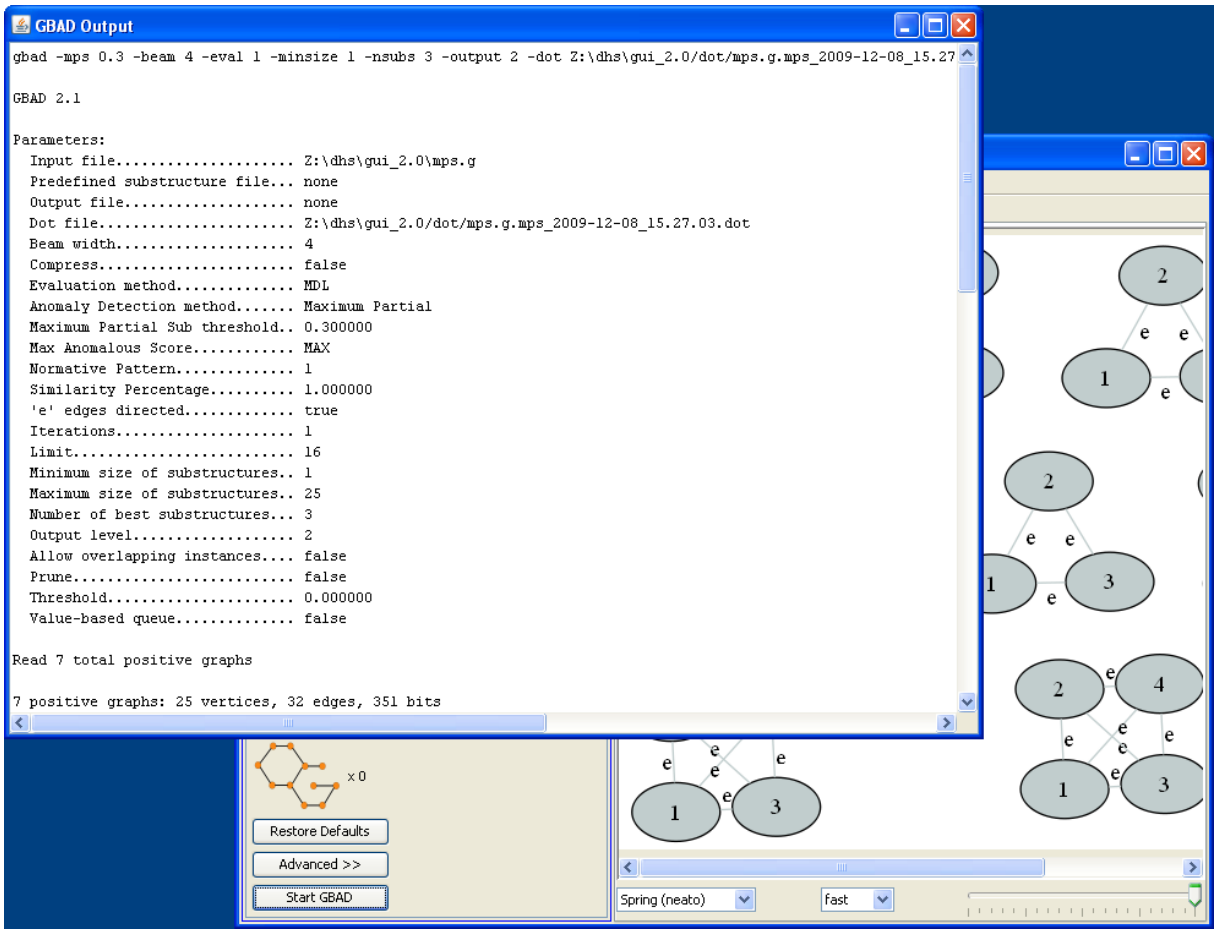

Figure 25: GBAD Output For mps.g.

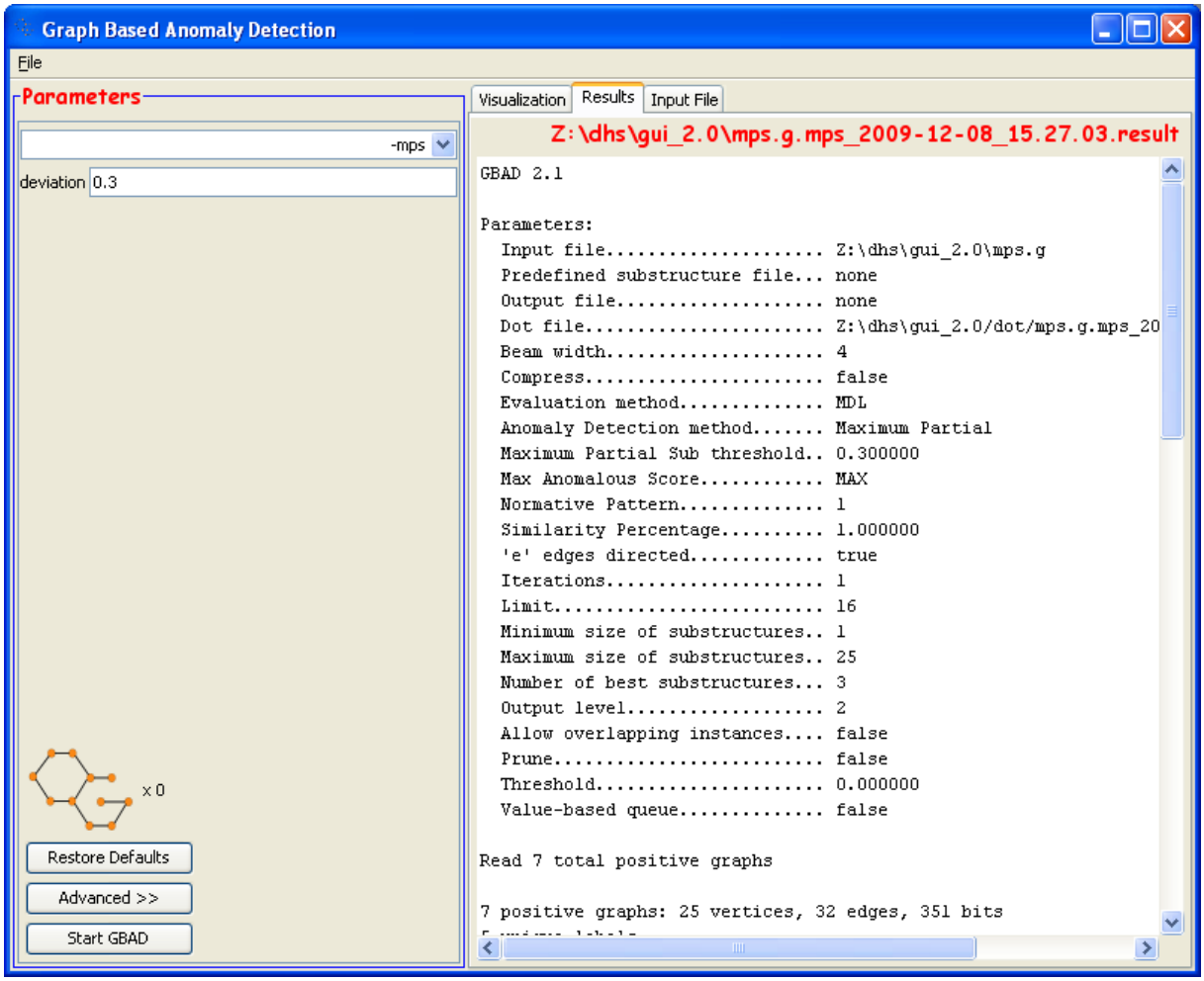

Figure 26: GBAD Results For mps.g.

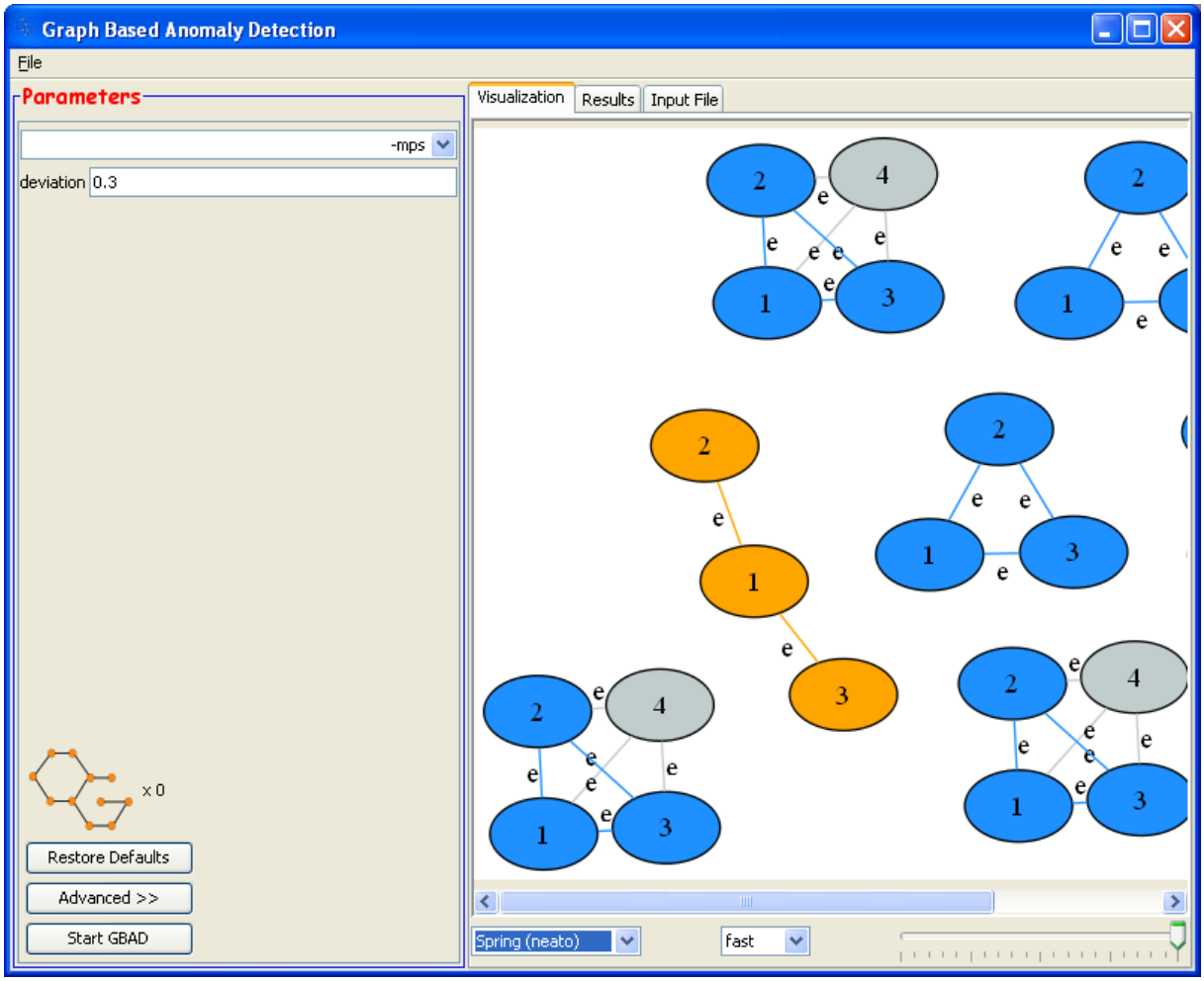

Figure 27: Visualization Of The Anomaly In The mps.g Graph.

6. Notes/Issues

The following sections represent various notes and issues.

6.1 Unix

GBAD was designed and developed to run on a Unix-based system. The application was tested on Linux, but should be compatible with any Unix system, as well as a Windows environment. GBAD was written in C, where every effort was made to use only standard ANSI C constructs and functions.

A. Appendix - Terminology

The following terminology was referenced in this document:

GBAD – Graph-Based Anomaly Detection

MDL – Minimum Description Length

MPS – Maximum Partial Substructure

SUBDUE – SUBstructure Discovery Using Examples

TBD – To Be Determined