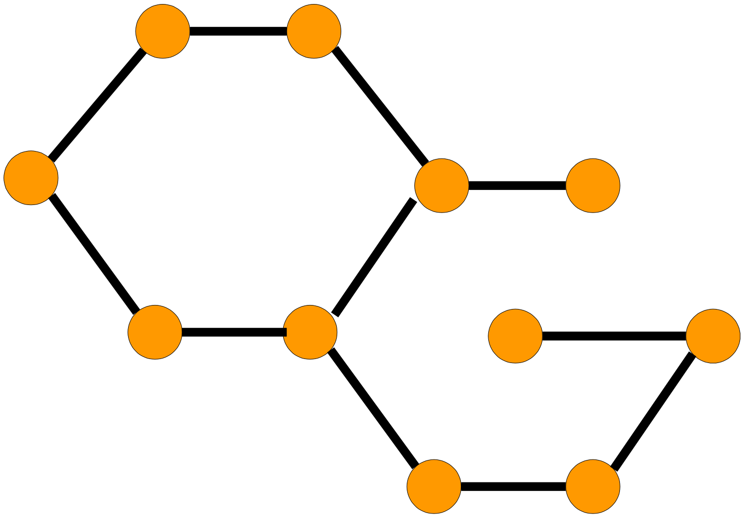

Graph-Based Anomaly Detection (GBAD)

User's Manual

Release 4.0

Table of Contents

Revision History

Release	Date	Revisions
1.0	December 15, 2009	Initial version of GBAD
2.0	September 2, 2011	Added GBAD-FSM to the GBAD test suite
3.0	December 2, 2011	Created common graph input file format
3.1	December 18, 2012	Added –noOpt option
3.2	June 18, 2014	Clarified file input format.
3.3	October 5, 2016	Fixes to gbad_mdl source and explanation of GUI.properties
4.0	February 21, 2020	Added option to run all algorithms in one run; changed names of algorithm parameters to match type of anomaly; added graph/substructure filtering capability; removed outdated GUI.

Preface

Conventions

The following documentation conventions are followed within this document.

<u>bold underlined text</u> 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 tool suite.

1.1 Overview

The GBAD graph-based anomaly detection tool suite discovers structural anomalies in data represented as a graph. The normative pattern discovery aspects of the GBAD-MDL system is based upon the SUBDUE graph-based pattern learning system, and the normative pattern discovery aspects of the GBAD-FSM system is based upon the GASTON frequent subgraph mining tool. (Details of how SUBDUE works internally can be found on the SUBDUE web-site; details for GASTON can be found on the GASTON 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 tools. 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: various notes and issues regarding the GBAD application
- Appendix A: terminology

1.2 Reference Documents

- ***GBAD Information:*** www.gbad.info
- ***SUBDUE Home Page:*** http://www.subdue.org/
- ***AT&T Labs GraphViz:*** http://www.graphviz.org/
- ***GASTON Home Page:*** http://www.liacs.nl/~snijssen/gaston/

2. Downloading and Installing

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

2.1 Download

The GBAD tool suite, 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 GBAD. 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 GBAD. 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 GBAD applications, you should perform the following steps on each of the GBAD tools:

2.3.1 GBAD-MDL

1. Change directory to *gbad-<release>/src*
2. At the command prompt, enter: *make*. This will compile the GBAD-MDL program.
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).

2.3.2 GBAD-FSM

1. Change directory to *gbad-fsm_<release>*
2. At the command prompt, enter: *make*. This will compile the GBAD-FSM program. The executable will be created in this directory.

3. Data Format

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

3.1 Input

The input to GBAD 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.[1] The *"XP"* is followed by a "#" and a unique example ID which must start at 1 and increase by one for each *"XP"*.

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. All labels 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. All labels 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.

Note: GBAD-FSM treats all edges as undirected, so directed edges are only allowed with the GBAD-MDL tool.

3.1.1.3 Comments

You can also choose to put comments in your graph input file. Comments are designated by the double slashes “//”. Anything after “//” 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).

4. Running GBAD

The purpose of GBAD 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>
```

or

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

There are several points which should be noted here:

gbad is the name of the GBAD-MDL executable, and *gbad-fsm* is the name of the GBAD-FSM 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.

The options are divided into three categories: Those that can be used by the GBAD-MDL system, those that can be used by the GBAD-FSM system, and those that can be used by either system.

4.1.1.1 GBAD-MDL Options

4.1.1.1.1 -add <#>

This option initiates the *prob* (probabilistic) algorithm to search for **anomalous additions** as part of the anomaly detection phase. The *prob* algorithm examines all extensions to the normative substructure with the lowest probability, and hence the most likely to be an anomaly. The *prob* algorithm 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 *–add* 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 *prob* algorithm, a minimum value of 2 must be specified.

4.1.1.1.2 -all <#.#>

This option initiates ALL of the anomaly detection algorithms (*add*, specified previous, *mod*, and *del* – see subsequent sections) as part of the anomaly detection phase. The parameter to the *–all* 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.

NOTE: Using the *–add* option will take longer to run than running GBAD on each of the individual anomaly types in parallel.

4.1.1.1.3 -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.

4.1.1.1.1 -del <#.#>

This option initiates the *mps* (maximum partial substructure) algorithm to search for **anomalous deletions** s part of the anomaly detection phase. The *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 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.

---

[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.

The parameter to the *–del* 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.

4.1.1.1.2 -eval <#>

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

4.1.1.1.2.1 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.

4.1.1.1.2.2 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.

4.1.1.1.1 -graphFilter <#>

This argument specifies the amount of the graph to be filtered out of consideration for possible normative patterns and anomalies. It uses an approach of removing the “murky middle”, whereby the initial list of starting vertices is purged from the initial list IF its counts are not high *and* not low. The idea being that we only care about frequent patterns in order to discover the normative pattern, and we only care about rare patterns in order to discover potential anomalies – anything “in the middle” is not important.

The parameter for the *–graphFilter* option is a number > 0.0 and < 0.99, indicating the amount of “middle” vertices to remove. For example, specifying *–graphFilter 0.4* would keep the top 0.3 of frequent vertices, and the bottom 0.3 of infrequent vertices, thus removing the middle 0.4. The default value for this to perform no graph filtering.

4.1.1.1.2 -limit <#>

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

4.1.1.1.3 -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.

4.1.1.1.4 -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.

4.1.1.1.5 -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.

4.1.1.1.6 -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.

4.1.1.1.1 -mod <#.#>

This option initiates the *mdl* (minimum description length) algorithm to search for **anomalous modifications** as part of the anomaly detection phase. The *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 *–mod* 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] In GBAD, the lower the score, the more anomalous a substructure.

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

4.1.1.1.1 -nAnoms <#>

This argument specifies the number of unique anomalies to return. The default value for this setting is 1.

4.1.1.1.2 -noOpt

This option turns off optimizations in the *mdl* and *mps* algorithms. Empirical analysis has demonstrated that in several cases, having this option on results in an order-of-magnitude run-time improvement. However, the topology of the graph does have an effect on the performance, and in some cases, there is no speed-up and may even be slower. By default, this option is on.

4.1.1.1.3 -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.

4.1.1.1.4 -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.

4.1.1.1.5 -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.

4.1.1.1.6 -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.

4.1.1.1.7 -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.

4.1.1.1.1 -subFilter <#>

This argument specifies the amount of substructures to be filtered out of consideration for possible normative patterns and anomalies. It uses an approach of removing the “murky middle”, whereby the current substructure candidates are purged IF their frequency is not high *and* not low. The idea being that we only care about frequent patterns in order to discover the normative pattern, and we only care about rare patterns in order to discover potential anomalies – anything "in the middle" is not important.

The parameter for the *–subFilter* option is a number > 0.0 and < 0.99, indicating the amount of "middle" substructures to remove. For example, specifying *–subFilter 0.4* would keep the top 0.3 of frequent substructures, and the bottom 0.3 of infrequent substructures, thus removing the middle 0.4. The default value for this to perform no substructure filtering.

4.1.1.1.2 -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.

4.1.1.1.3 -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.

4.1.1.2 GBAD-FSM Options

4.1.1.2.1 –bestSubInstOut

This option enables the output of all the best substructure instances to the *best_sub.inst* file.

4.1.1.2.2 –graph <filename>

The <*filename*> following this option specifies the fully-pathed input graph file.

4.1.1.2.3 –mst <threshold>

The value specified for *-mst* is the minimum support threshold. Which means that the substructure shown in *best_sub.g* is a subgraph that is included in at least <*threshold*> transactions (or "XP"s in the graph input file).

4.1.1.2.4 –nameAnomSub <filename>

The <*filename*> following this option specifies the file to which GBAD-FSM will write anomalies. The default filename is *anom_sub.g*.

4.1.1.2.5 –nameBestSubG <filename>

The <*filename*> following this option specifies the file to which GBAD-FSM will write the best substructure information. The default filename is *best_sub.g*.

4.1.1.2.6 –nameBestSubINST <filename>

The <*filename*> following this option specifies the file to which GBAD-FSM will write the best substructure instances information. The default filename is *best_sub.inst*.

4.1.1.2.7 *–phase <#>*

GBAD-FSM consists of two phases (determining the normative pattern, and anomaly detection) which can be run separately or together. Specifying "*-phase 1*" will run only the search for the normative pattern, and "*-phase 2*" will run only the anomaly detection. If this option is not specified, the system runs both phases.

4.1.1.2.8 *–subInstOut <filename>*

Specifying this option enables the system to output ALL of the considered substructure instances to <*filename*>. (NOTE: This file can get very large!)

4.1.1.2.9 *-mdl <#.#>*

Same as the *–mod* option, as defined in Section 4.1.1.1.1.

4.1.1.2.10 *-mps <#.#>*

Same as the *–del* option, as defined in Section 4.1.1.1.1.

4.1.1.2.11 *-prob <#>*

Same as the *–add* option, as defined in Section 4.1.1.1.1.

4.1.1.3 *Shared Options*

4.1.1.3.1 *-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 colored. The vertices and edges that are part of the normative pattern are colored blue. Any anomalies found by the *mps* algorithm are colored orange. The anomalous substructures found by the *mdl* algorithm are colored orange and the actual anomaly within the substructure is colored red. Anomalies found by the *prob* 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 publicly-available *GraphViz* package.

4.2 Example

It is now time to run GBAD. In this example, we will run just the GBAD-MDL algorithm:

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

Figure 1 shows a portion of the actual textual output of the GBAD run on this example graph. **Figure 2** shows a graphical representation of the normative pattern. **Figure 3** shows a graphical representation of the anomalous pattern, with the specific anomalous vertex.

```
GBAD 3.0

Parameters:
  Input file..................... graphs/graph_with_modification.g
  Predefined substructure file... none
  Output file.................... none
  Dot 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
  '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"
    d 219 215 "e5"
    d 219 216 "e4"
    d 220 217 "e3"
    d 220 218 "e2"
    d 220 219 "e9" <-- anomaly (original edge vertices: 220 -- 219, in
original example 1)

. . .
```

Figure 1: Actual partial textual output of the GBAD-MDL run on the sample graph.

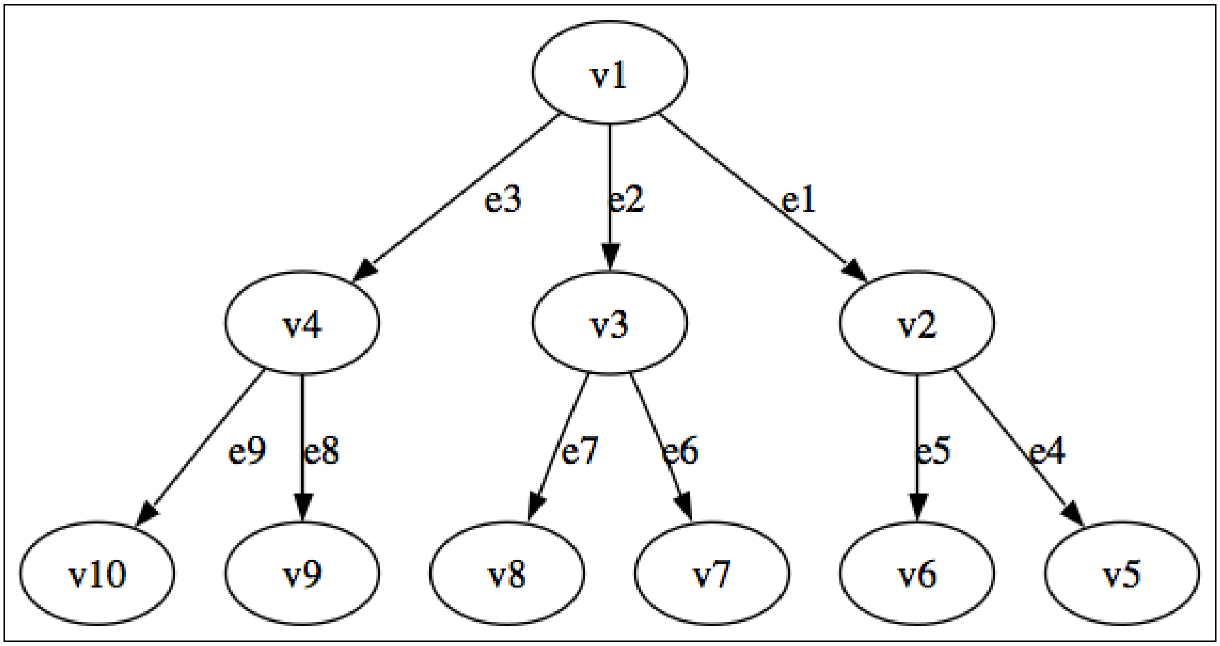

Figure 2: Graphical representation of the normative pattern.

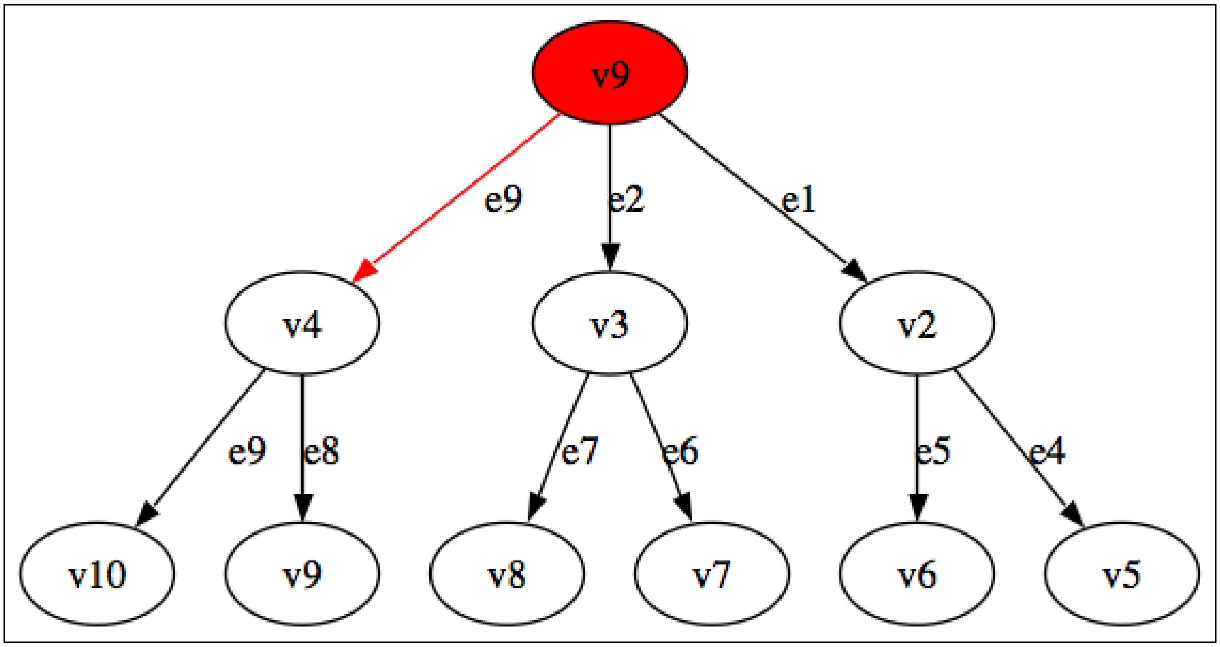

Figure 3: Graphical representation of the anomalous substructure (with specific anomaly in red) discovered in the example.

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, the size of the input graph, the number of unique labels, and the connectivity of the graph. However, 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 (Anomalous Insertion) Example

The following example graph contains several disconnected subgraphs. Every edge in the graph is given a label of ‘e.’ Four of the subgraphs are the complete graph on 4 vertices with the vertices labeled 1 through 4. Two of the subgraphs are the complete graph on 5 vertices with the vertices labeled 1 through 5. The graph also contains one additional subgraph that is a complete graphs on 5 vertices with an extension attached. The vertices on this subgraph are labeled 1 through 5 and there is an extra vertex labeled ‘V.’ This ‘V’ vertex is the target anomaly to be detected. In order to find this vertex the probabilistic algorithm is used.

4.3.1.1 Input

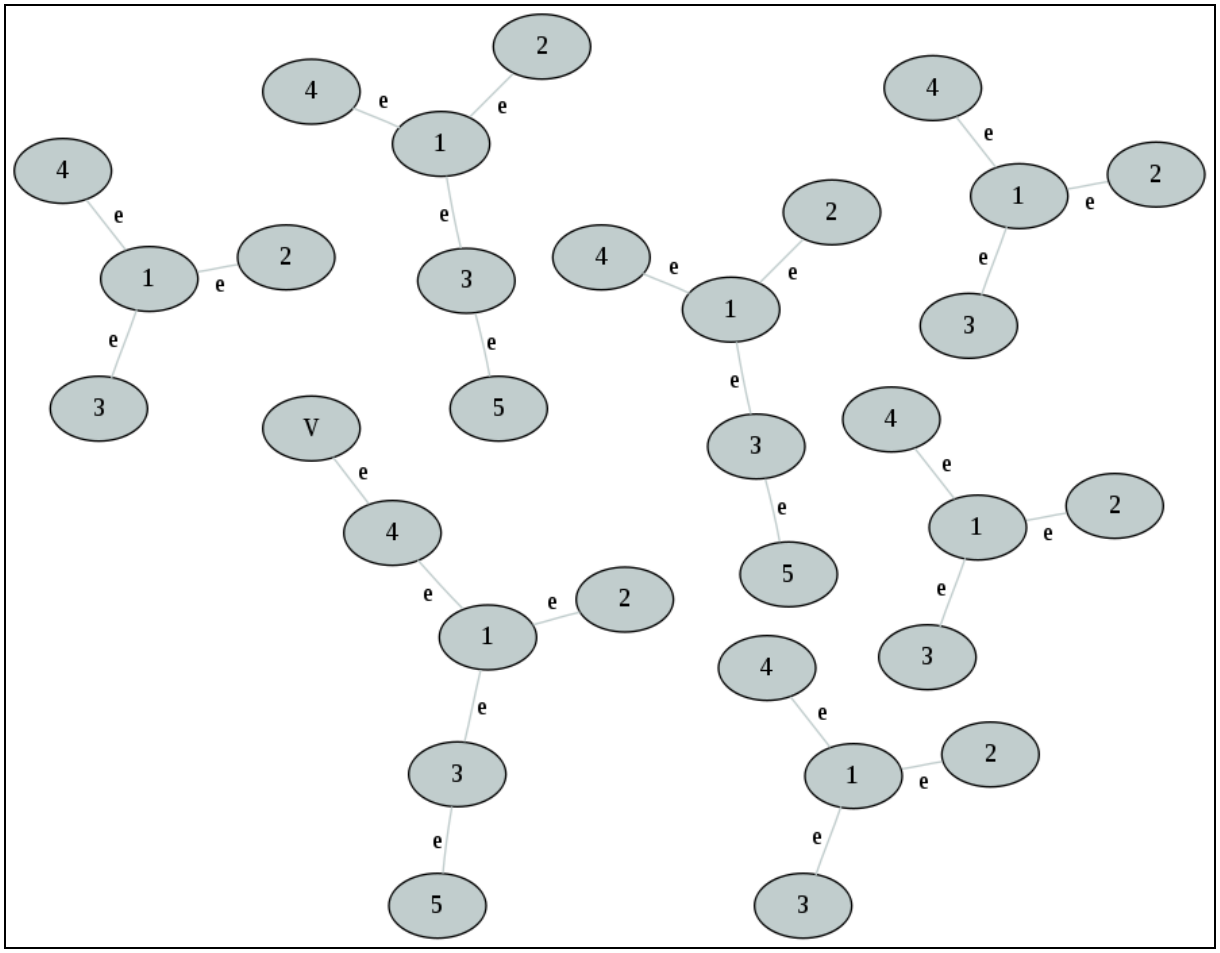

Figure 4: Graphical picture of graph input file containing an anomalous insertion.

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

Figure 5: GBAD input graph file for prob_example.g.

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

4.3.1.2 Execution

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

```
gbad-mdl/bin/gbad -add 2 graphs/prob_example.g
```

Or similarly, we can run GBAD-FSM using this command:

```
gbad-fsm/gbad-fsm –prob 2 –mst 1 –graph graphs/prob_example.g
```

Figure 6 shows a graphical depiction of the normative substructure discovered by GBAD after the first iteration. **Figure 7** shows the textual output of GBAD-MDL after the second iteration. **Figures 8** and **9** show the output from GBAD-FSM. **Figure 10** shows the graphical depiction of the anomaly detected by the probabilistic algorithm.

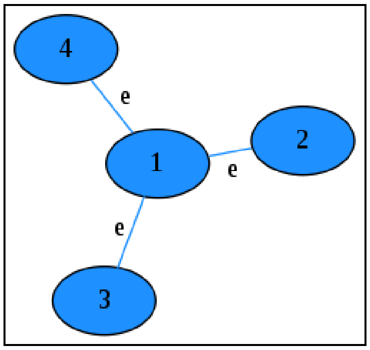

Figure 6: Graphical depiction of normative substructure discovered by GBAD after the first iteration.

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

7 positive graphs: 11 vertices, 4 edges, 69 bits
4 unique labels

2 initial substructures
Normative Pattern:
Substructure: value = 1.38643, instances = 4
  Graph(2v,1e):
    v 1 SUB_1
    v 2 "5"
    u 1 2 "e"

Anomalous Instance(s):

 from positive example 2:
    v 6 SUB_1
    v 10 "V" <-- anomaly (original vertex: 6 , in original example
6)
    u 6 10 "e" <-- anomaly (original edge vertices: 4 -- 6, in
original example 6)
    (probabilistic anomalous value = 0.142857 )
...
```

Figure 7: Actual partial textual output of the *prob* run on the sample graph after the second iteration in GBAD-MDL.

```
v 0 0
v 1 1
v 2 2
v 3 3
e 0 1 4
e 0 2 4
e 0 3 4
z
7
```

Figure 8: *best_sub.g* after the GBAD-FSM run on the sample graph.

```
% transaction containing anomalous structure: 6
% probability: 0.142857
v 3 3
v 5 6  <--- anomaly
v 0 0
v 2 2
v 1 1
e 0 3 4
e 0 2 4
e 0 1 4
e 3 5 4  <--- anomaly
```

Figure 9: *anom_sub.g* after the GBAD-FSM run on the sample graph.

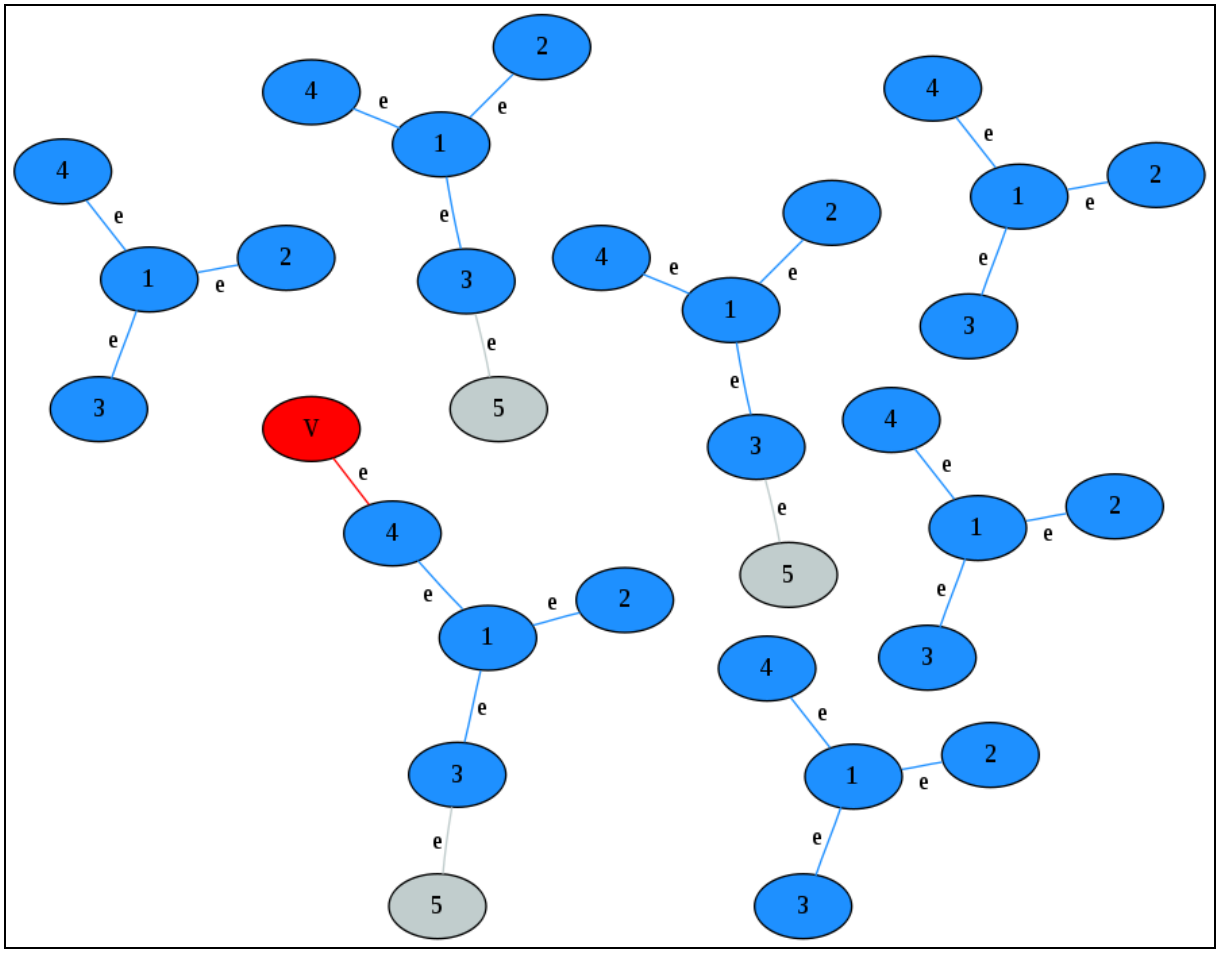

Figure 10: Graphical depiction of the normative pattern and anomaly reported by running the *prob* algorithm.

4.3.1.3 Analysis

In this run of GBAD, the complete graph of 4 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.

4.3.2 MDL (Anomalous Modification) 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 5 vertices. All of the vertices have labels ranging from 1 to 5. 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

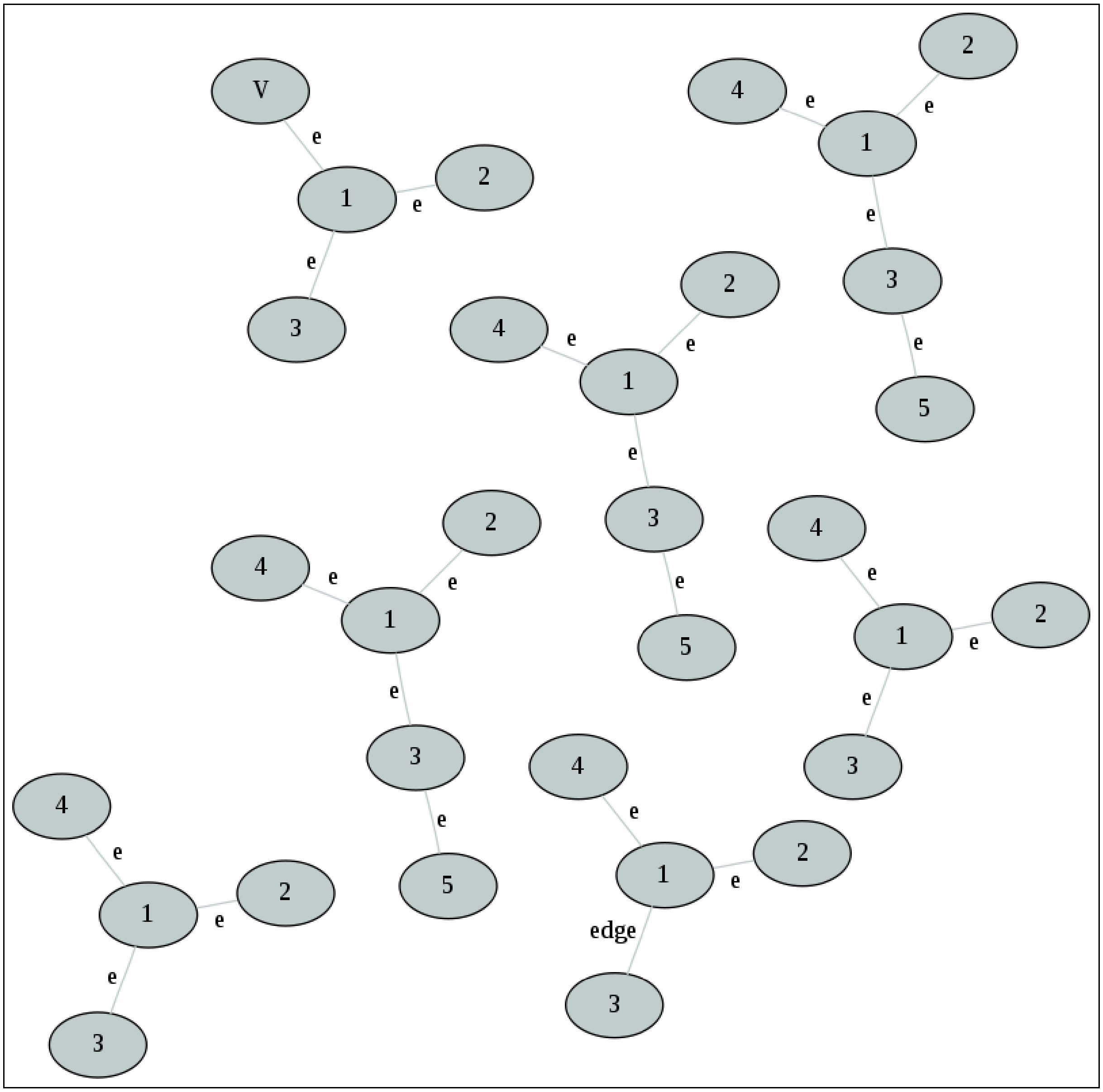

Figure 11: Graphical picture of a graph input file containing an anomalous modification.

```
...
XP # 1
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "V"
u 1 2 "e"
u 1 3 "e"
u 1 4 "e"
XP # 2
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "4"
u 1 2 "e"
u 1 3 "edge"
u 1 4 "e"
XP # 3
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "4"
u 1 2 "e"
u 1 3 "e"
u 1 4 "e"
...
```

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

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

4.3.2.2 Execution

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

```
gbad-mdl/bin/gbad -mod 0.2 graphs/mdl_example.g
```

Similarly, we can run GBAD-FSM using this command:

```
gbad-fsm/gbad-fsm –mdl 0.2 –mst 1 –graph graphs/mdl_example.g
```

Figure 13 shows a graphical depiction of the normative substructure discovered by both GBAD systems. **Figure 14** shows the textual output of GBAD-MDL. **Figures 15** and **16** show the contents of *best_sub.g* and *anom_sub.g* produced by GBAD-FSM. **Figure 17** shows the graphical depiction of the anomaly detected by the minimum description length algorithm.

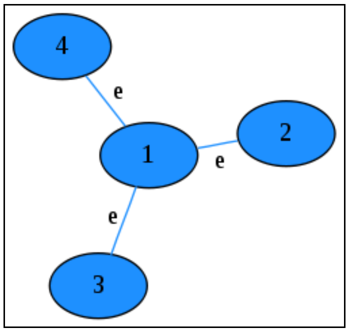

Figure 13: Graphical depiction of normative substructure discovered by GBAD.

```
...
6 initial substructures
Normative Pattern (1):
Substructure: value = 1.78426, instances = 5
  Graph(4v,3e):
    v 1 "1"
    v 2 "2"
    v 3 "3"
    v 4 "4"
    u 1 2 "e"
    u 1 3 "e"
    u 1 4 "e"

Anomalous Instance(s):

 from example 2:
    v 5 "1"
    v 6 "2"
    v 7 "3"
    v 8 "4"
    u 5 6 "e"
    u 5 7 "edge" <-- anomaly (original edge vertices: 1 -- 3, in
original example 2)
    u 5 8 "e"
    (information_theoretic anomalous value = 1.000000 )

 from example 1:
    v 1 "1"
    v 2 "2"
    v 3 "3"
    v 4 "V" <-- anomaly (original vertex: 4 , in original example
1)
    u 1 2 "e"
    u 1 3 "e"
    u 1 4 "e"
    (information_theoretic anomalous value = 1.000000 )
...
```

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

```
v 0 0
v 1 2
v 2 5
v 3 1
u 0 1 4
u 0 2 4
u 0 3 4
z
5
```

Figure 15: The contents of *best_sub.g* after the GBAD-FSM run on the sample graph.

```

% transaction containing anomalous structure: 2
% anomalous score: 0.142857
v 0 0
v 2 2
v 3 5
v 1 1
u 0 2 6 <---- anomaly
u 0 3 4
u 0 1 4
%
%
% transaction containing anomalous structure: 1
% anomalous score: 0.142857
v 0 0
v 2 2
v 3 3 <---- anomaly
v 1 1
u 0 2 4
u 0 3 4
u 0 1 4
```

Figure 16: The contents of *anom_sub.g* after the GBAD-FSM run on the sample graph.

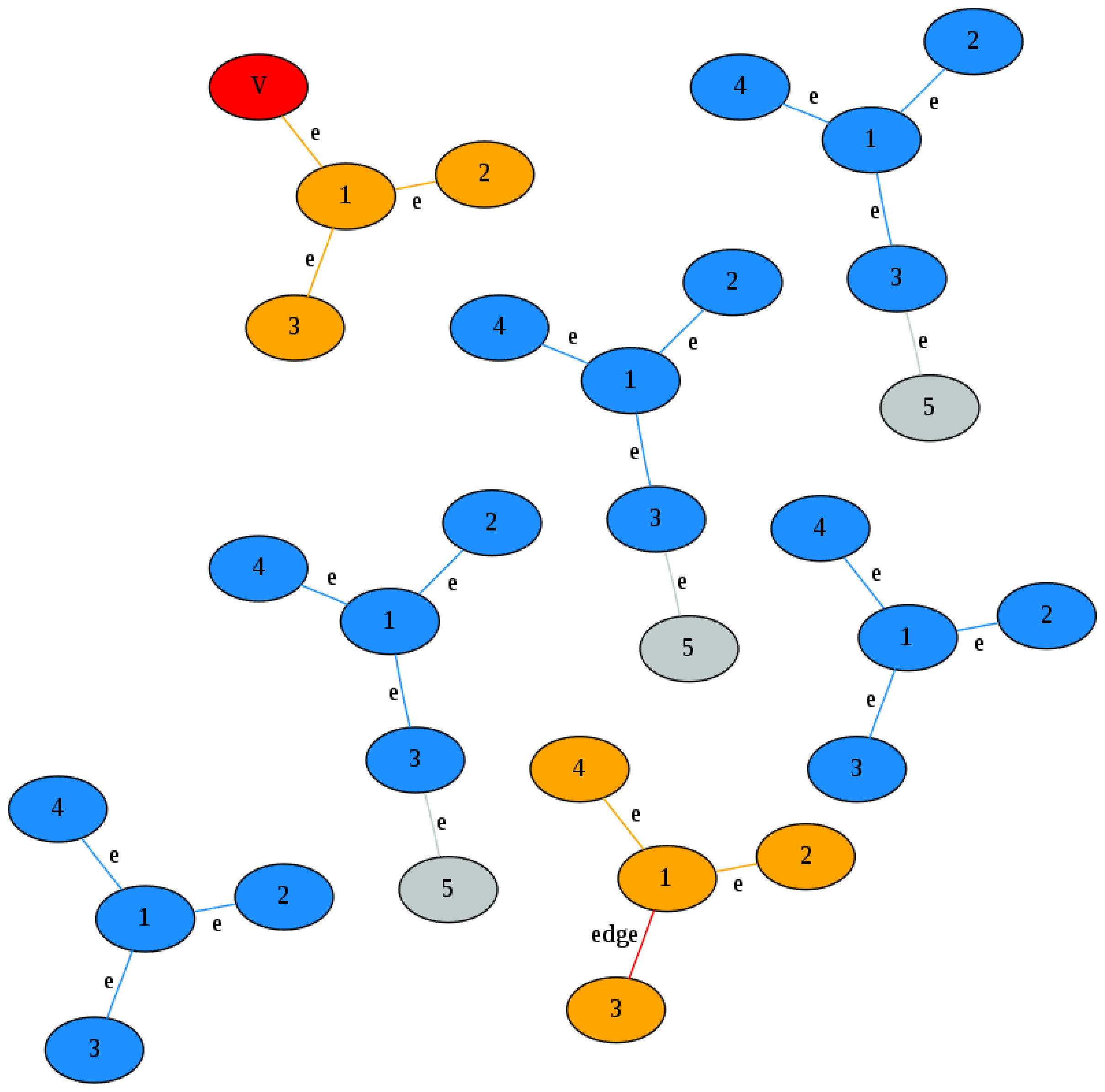

Figure 17: Graphical depiction of the normative pattern and anomaly reported by GBAD-MDL and GBAD-FSM.

4.3.2.3 Analysis

In this run of GBAD, the complete graph of 4 vertices was found to be the normative pattern. The *mdl* algorithm found that 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 for GBAD-MDL as seen in **Figure 14**, and 0.142857 (which is one change divided by seven total instances) for GBAD-FSM as seen in **Figure 16**.

4.3.3 Maximum Partial Substructure (Anomalous Deletion) Example

The following example graph contains several disconnected subgraphs. Three of the subgraphs are the complete graph of 4 vertices and three of the subgraphs are the complete graph of 5 vertices. All of the vertices have labels ranging from 1 to 5. 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 '4' the structure would be the complete graph on 4 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

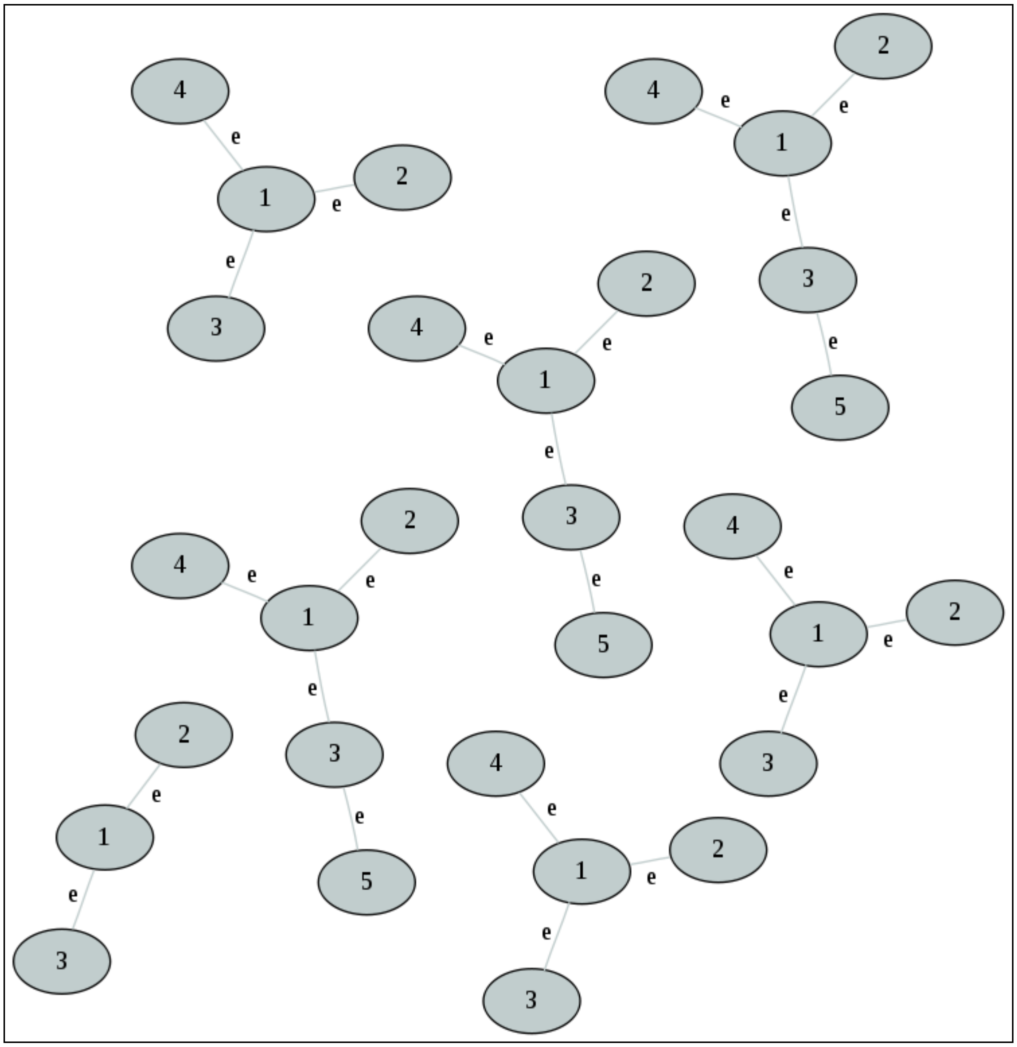

Figure 18: Graphical picture of graph input file containing anomalous deletion.

```
...
XP # 2
v 1 "1"
v 2 "2"
v 3 "3"
v 4 "4"
u 1 2 "e"
u 1 3 "e"
u 1 4 "e"
XP # 3
v 1 "1"
v 2 "2"
v 3 "3"
u 1 2 "e"
u 1 3 "e"
...
```

Figure 19: GBAD input graph file mps_example.g.

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

4.3.3.2 Execution

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

```
gbad-mdl/bin/gbad -del 0.3 graphs/mps_example.g
```

Or similarly, we can run GBAD-FSM using this command:

```
gbad-fsm/gbad-fsm -mps 0.3 -mst 1 -graph graphs/mps_example.g
```

Figure 20 shows a graphical depiction of the normative substructure discovered by GBAD-MDL and GBAD-FSM. **Figure 21** shows the textual output of GBAD-MDL, and **Figures 22** and **23** show the results reported by GBAD-FSM. **Figure 24** shows the graphical depiction of the anomaly detected by the maximum partial substructure algorithm.

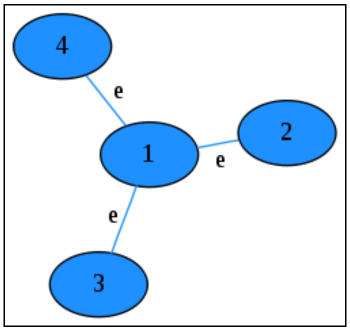

Figure 20: Graphical depiction of normative substructure discovered by GBAD.

```
...
Read 7 total positive graphs

7 positive graphs: 31 vertices, 23 edges, 328 bits
6 unique labels

5 initial substructures
Normative Pattern (1):
Substructure: value = 2.34497, instances = 6
  Graph(4v,3e):
    v 1 "1"
    v 2 "2"
    v 3 "3"
    v 4 "4"
    u 1 2 "e"
    u 1 3 "e"
    u 1 4 "e"

Anomalous Instance(s):

 from example 3:
    v 9 "1"
    v 10 "2"
    v 11 "3"
    u 9 10 "e"
    u 9 11 "e"
    (max_partial_substructure anomalous value =
2.000000 )
...
```

Figure 21: Actual partial textual output of the GBAD-MDL run on the sample graph.

```
v 0 0
v 1 1
v 2 3
v 3 2
e 0 1 4
e 0 2 4
e 0 3 4
z
6
```

Figure 22: *best_sub.g* after the GBAD-FSM run on the sample graph.

```
% transaction containing anomalous structure: 3
% anomalous value: 0.285714
v 0 0
v 1 1
v 2 2
e 0 1 4
e 0 2 4
```

Figure 23: *anom_sub.g* after the GBAD-FSM run on the sample graph.

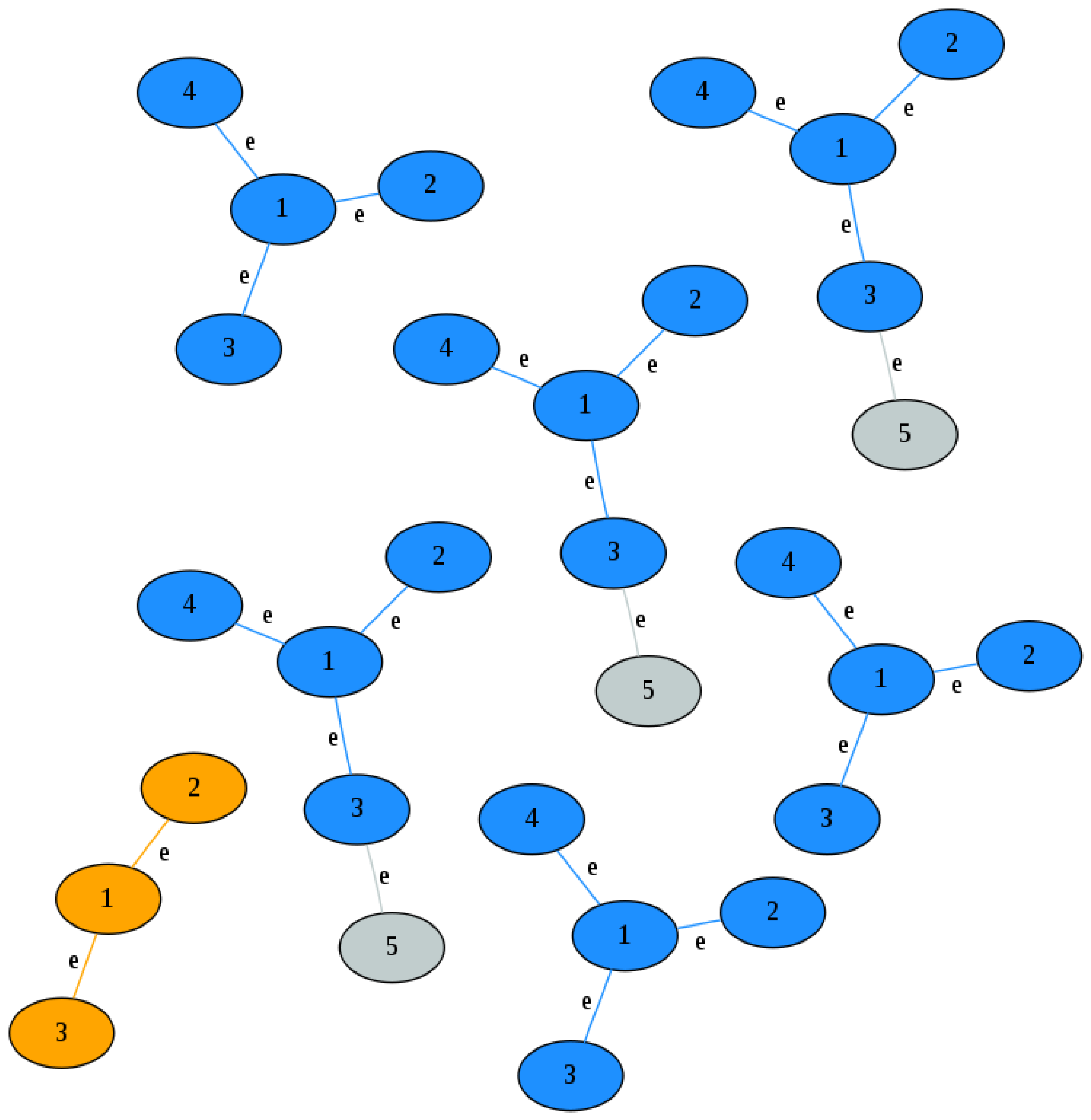

Figure 24: Graphical depiction of the normative pattern and anomaly reported by GBAD-MDL and GBAD-FSM.

4.3.3.3 Analysis

In this run of GBAD, the complete graph of 4 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 one edge and one vertex insertion were needed, resulting in an anomalous value as shown in **Figure 21** and **Figure 23**.

5. Notes/Issues

The following sections represent various notes and issues.

5.1 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/C++, where every effort was made to use only standard ANSI C/C++ constructs and functions.

5.2 GBAD-FSM versus GBAD-MDL

It should be noted that the GBAD-FSM tool works best on graphs with multiple transactions (or XPs). Because it is based on a frequent subgraph miner, graphs with single transactions (especially large ones) take much longer to process because of the internal tree that is created. Therefore, if the input graph file that you want to process is composed of a single transaction (or XP), it is best to use the GBAD-MDL tool.

5.3 Software Releases

While changes may occur to the GBAD tools, every effort has been made to minimally impact the user. The current versions of the software that go with Version 4.0 of this manual are as follows:

- gbad-mdl (4.0)
- gbad-fsm (2.1)

A. Appendix - Terminology

The following terminology was referenced in this document:

FSM – Frequent Subgraph Miner

GASTON - GrAph, Sequences and Tree extractiON

GBAD – Graph-Based Anomaly Detection

MDL – Minimum Description Length

MPS – Maximum Partial Substructure

SUBDUE – SUBstructure Discovery Using Examples

TBD – To Be Determined