Undergraduate Research Opportunity Research

(UROP) Project Report

Linear Representation of Graphs

By
Lim Junliang Kevin

Department of Computer Science
School of Computing
National University of Singapore
2007/2008

Undergraduate Research Opportunity Program
(UROP) Project Report

Linear Representation of Graphs

By
Lim Junliang Kevin

Department of Computer Science
School of Computing
National University of Singapore
2007/2008

Project No: U114080
Advisor: Prof Wong Limsoon
Deliverables:
Report: 1 Volume

Abstract

Chemical structures are usually stored in a database using unique keys as an identifier. However, many of the present chemical database systems use keys that may not allow the chemical structure to be derived. This is an aspect of usefulness of the unique keys. In contrast, some keys that are useful may not be unique enough. In a recent publication, the SMILES algorithm is suggested to be producing non-unique identifiers (Robert Grossman et al, 2005). In this paper, we debate the usefulness and uniqueness of SMILES algorithm to a certain extent, show how the SMILES algorithm fails to produce unique identifiers and show how we can adapt from the SMILES algorithm to produce identifiers that are unique and useful. We also developed an algorithm named Graph Linear Representer (GLR) that constructs a unique identifier for chemical structures. The algorithm was tested against 10,000 random structures obtained from the national cancer institute database. In this paper, we also show that the new algorithm produces unique keys that are useful , unique and flexible.

Subject Descriptor:

G.1.0 General

J.3 Life and Medical Sciences

Keywords:

Chemical Databases, Biological Data, Linear Representation of Graphs, Graph Databases.

Acknowledgements

I hereby thank all those who have contributed and rendered help towards making this project possible. I would like to thank Prof Wong Limsoon, my advisor who has provided great insights and valuable advice. I would also like to thank Chen Xian Kun for providing me with randomly generated graphs for testing.

Table of Contents

1 Introduction

The amount of biological and chemical information has been increasing at a very fast rate (Robert Grossman et al 2005). As of 2008, the national cancer institute database has 250,251 structures, the chemical abstracts service database has close to 29 million structures and increasing at a rate of 3,000 structures per day (sources taken from cas website), and the pubchem database has close to 19 million compounds and 40 million substances both growing dynamically. These statistics infer the need for efficient and effective ways to store these data. One of the problems associated to these concerns is related to assigning unique identifiers for each chemical structure. The reason is that a mistake in entering two identical structures into a database can have a heavy impact, implying that data is increasing at an unnecessary rate. This report is concerned with assigning unique identifiers for chemical substances using features present in chemical structures, addressing the concerns mentioned above.

1.1 Research Background

As of today, chemical databases employ different ways to identify the chemical structures. For example, the chemical abstracts service uses a unique CAS number as foreign key (ibid), while other databases may depend on the CAS number or use other commonly used unique keys like the INChi chemical identifier or the SMILES unique keys. A recent study has been done by Robert Grossman et al to access these unique identifiers (ibid). The study claimed a few counter examples for the SMILES unique keys and also identified a few issues concerning the usefulness of other methods. For example, a naïve method to distinguish two identical structures would be to compare two graphs with a graph isomorphism algorithm. However such a method would not be useful because the discrimination between two graphs requires a lot of computing and does not provide any unique identifier. In addition, a useful identifier should be one such that the chemical structure can be derived. In view of this, the CAS number may not be a useful chemical identifier.

1.2 Research Motivation

The motivation for research arises from the counter examples provided by Robert Grossman et al. The study showed that although the SMILES identifier is useful because of its ability to derive the chemical compound from the identifier itself, it is not unique. Hence, it is worthy

to understand what had failed in the SMILES algorithm and further identify how an identifier can be designed to so that it is both useful and unique. In addition, we have also identified some mistakes made by Robert Grossman et al when attempting to show that the SMILES algorithm does not provide a unique identifier. This allows us further insights to the real reason behind why the SMILES algorithm did not produce a unique identifier.

1.3 Research Contributions

We believe that our paper makes the following research contributions:

1. We show that Grossman's conclusion about the SMILES algorithm was not correct.
2. We further explain what the problem with the SMILES algorithm was and how we can use what was working in the SMILES algorithm and discard what that did not work.
3. We produce a new algorithm to derive unique identifiers from chemical structures.

1.4 Outline of Report

The report will be organized in two following manner. Section 2 discusses the SMILES technique and how it had failed to produce a unique identifier with reference to Grossman et al, we also include our interpretation since we discovered that they had made a few mistakes. Section 3 discusses a new design of unique identifier and why the new design works. In section 4 we describe what can be done to further improve on this new design. Finally in section 5, we give some references.

2 The SMILES algorithm

The unique SMILES algorithm is divided into two stages. In the first stage, the algorithm attempts to provide a canonical labeling of the graphs' nodes (CANON) and in the second stage, the algorithm produces an identifier by producing a depth-first traversal (GENES). Grossman identified that the key procedure in defining unique identifiers lies in the canonical labeling stage of the algorithm. It is thus worthy to understand the CANON algorithm.

2.1 The CANON algorithm

The key idea behind the CANON algorithm is the use of properties in graphs that does not change in all isomorphic forms of the same graph. These are termed as graph invariants and

the CANON algorithm proposes the use of six invariants ranked in order as follows (with reference to Grossman):

1. Number of connections
2. Number of non-hydrogen bonds
3. Atomic number
4. Sign of charge
5. Absolute charge
6. Number of attached hydrogen

Briefly, the algorithm can be described as the following :

Step1: Rank each node according to graph invariants (described above), in its corresponding order.

Step2: Each rank is associated with a prime number sorted in ascending order.

Step3: For each node, a score is assigned. The score is obtained by multiplying its direct neighbor's associated prime number.

Step4: The nodes are ranked again with respect to the score obtained in step 3.

Step5: Repeat step 1 until the ranking does not change.

Step6: If the algorithm stops at step 5, check if each node is ranked uniquely. i.e. the highest rank is the total number of nodes. If the nodes are not ranked uniquely, the rank of the first node sharing the same rank is decreased by 1 and step 1-6 is repeated until each node is ranked uniquely.

2.2 Analysis of the CANON algorithm

An interesting fact behind this algorithm is that at step 5 of the algorithm, the nodes are said to be in their equivalence classes. Figure 1 provides a clear picture on equivalence classes with each node marked uniquely according to their equivalence classes. It can be seen that any two nodes sharing the same class are symmetrical to each other.

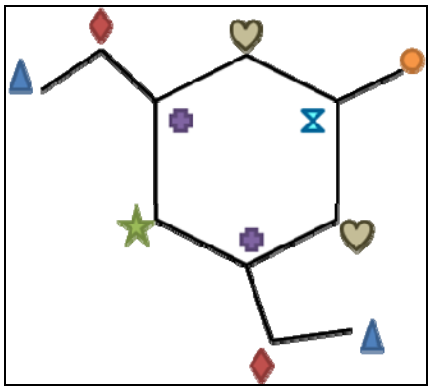


Figure 1. Equivalence class of a graph.

An ambiguity arises in step 6 where the CANON algorithm proposes to decrease the first node that shares the same rank. Grossman argued that there can be many ways to decrease these rankings depending on the initial node ordering. This could result in a non-unique identifier produced at the later stage. To further understand the implication of this ambiguity, we refer to Grossman's counter example demonstrated in table 1 and figure 2.

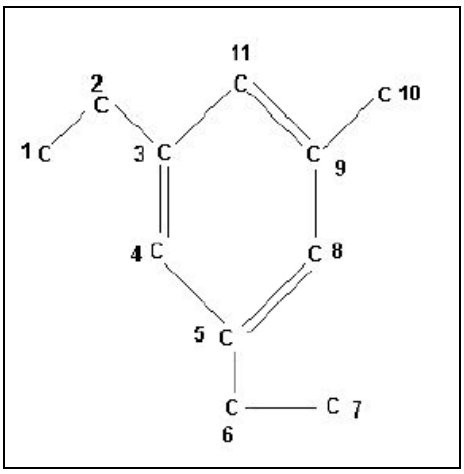


Figure 2a. A counter example proposed by Grossman

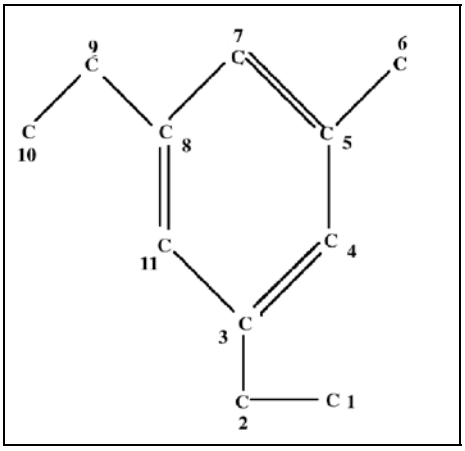


Figure 2b. An alternative numbering of graph in Figure 2a.

	Node id	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
rank	(a) Initial atomic invariants											
associate with primes	A	1,01, 06,0, 0,3	2,02, 06,0, 0,2	4,04, 06,0, 0,0	3,03, 06,0, 0,1	4,04, 06,0, 0,0	2,02, 06,0, 0,2	1,01, 06,0, 0,3	3,03, 06,0, 0,1	4,04, 06,0, 0,0	1,01, 06,0, 0,3	3,03, 06,0, 0,1
	B	1	2	4	3	4	2	1	3	4	1	3
multiply	(b) Classification by product of primes											
	B*	2	3	7	5	7	3	2	5	7	2	5
	C	3	14	75	49	75	14	3	49	50	7	49
repeat	D	1	3	6	4	6	3	1	4	5	2	4
	D*	2	5	13	7	13	5	2	7	11	3	7
	E	5	26	245	169	245	26	5	143	147	11	143
	F	1	3	7	5	7	3	1	4	6	2	4
	F*	2	5	17	11	17	5	2	7	13	3	7
	G	5	34	385	289	385	34	5	221	147	13	221
	H	1	3	7	5	7	3	1	4	6	2	4

Table 1. An instance of CANON algorithm executed graph in figure 2a. by Grossman.

According to the CANON algorithm, the algorithm pauses when the ranking does not change as demonstrated at step H of table 1. At this point, the nodes are said to be in equivalence classes. We observe that there are two nodes that belong to the equivalence class of rank 1 at the final step. Grossman et al claimed that because of this, there are two ways to choose reduce the rank in one equivalence class. He further showed that by choosing each node in the same equivalence class independently, the graphs would result in a non-unique canonical order. However, we discovered a mistake because the nodes were ranked wrongly at step F as shown, circled in table 1. This is an obvious mistake because node 9's score (147) is less than node 4's score (169) and should be given a lower rank. We present a corrected version in table 2.

G1	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Rank	1	2	4	3	4	2	1	3	4	1	3

Prime	2	3	7	5	7	3	2	5	7	2	5
Multiply	3	14	75	49	75	14	3	49	50	7	49
Rank	1	3	6	4	6	3	1	4	5	2	4
Prime	2	5	13	7	13	5	2	7	11	3	7
Multiply	5	26	245	169	245	26	5	143	147	11	143
Rank	1	3	7	6	7	3	1	4	5	2	4
Prime	2	5	17	13	17	5	2	7	11	3	7
Multiply	5	34	455	289	455	34	5	187	147	11	187
Rank	1	3	7	6	7	3	1	5	4	2	5
Prime	2	5	17	13	17	5	2	11	7	3	11
Multiply	5	34	715	289	715	34	5	119	363	7	119
Rank	1	3	7	5	7	3	1	4	6	2	4
Prime	2	5	17	11	17	5	2	7	13	3	7
Multiply	5	34	385	289	385	34	5	221	147	13	221
Rank	1	3	7	6	7	3	1	5	4	2	5

Table 2. A corrected instance of CANON algorithm on graph in figure 2a.

This correction allows us to have further insights to the CANON algorithm. To further analyze the uniqueness of the canonical labeling, we first present the two final canonical labeling from Grossman's analysis compared to our corrected version presented in figure 3a/b. The graph on the left in figure 3 was obtained after node 1's rank was reduced. The graph on the right in figure 3 was obtained after node 7's rank was reduced. It is easy to see that the graphs from the corrected canonical node labeling in figure 3b are identical and can be seen by a 180° rotation along the x axis followed by a 60° rotation along the z axis. This information allowed us to conclude that Grossman's claim was wrong because we are able to show that the CANON algorithm could indeed produce a unique canonical label no matter which node's rank is reduced if and only if they are in the same equivalence class and consistently chosen. We explain this consistency following.

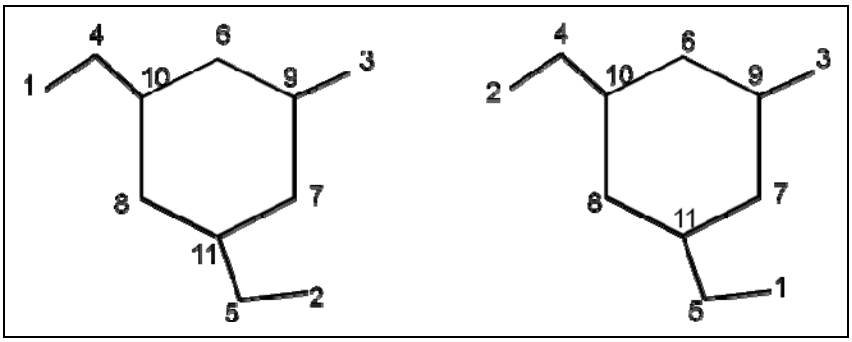


Figure 3a. Grossman's canonical node labeling.

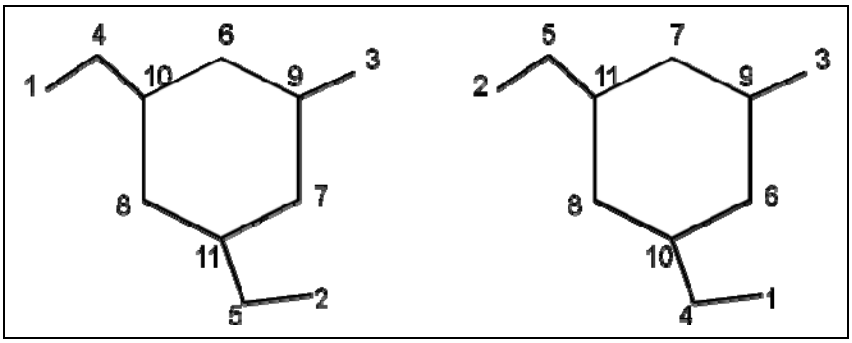


Figure 3b. A corrected canonical node labeling.

On further examination, the CANON algorithm only failed to present a canonical labeling when the algorithm is not consistent in choosing one of the two nodes in the same equivalence class, in which case, the resulting graphs were similar to those that Grossman had produced as shown in figure 3a. This is because when steps 1-5 were repeated, the algorithm pauses a couple of times. At each pause, the algorithm determines one node from a symmetry class to reduce its rank. If this node is not consistently chosen at each pause, the algorithm fails to produce a canonical labeling.

From this point of view, we have identified the disadvantages of the CANON algorithm. The disadvantage is clearly that it does not always produce a unique canonical graph labeling. However, it is also worthy to highlight the use of symmetry will still work if there is a way to consistently choose amongst those in the same symmetry classes.

3 Graph Linear Representer – The GLR algorithm

The GLR algorithm was created in an attempt to preserve what which was supposed to work in the CANON algorithm and discard what did not work in it. To reiterate, the main idea was to find a way to consistently differentiate which of the nodes to choose within a symmetric class. It is also worthy to first discuss the other motivations and original intentions to create a unique identifier before proceeding with the GLR algorithm.

3.1 Motivation to the GLR Algorithm

The motivation towards the initial idea was related to graph searching. In a recent work, GString, Jiang et al described a way to index chemical structures by converting them into linear strings (Haoliang Jiang et al, 2007). The goals in graph searching and graph identification are different. This is in part due to the different problem requirements. A

typical graph search problem is concerned with finding a way to quantify similarities in graphs. In the GString algorithm, basic and frequently occurring subgraphs were chosen as features to be extracted out from a graph. These features were converted to strings, which were in turn, a basis for graph indexing. Following up on this idea, one could generate common subgraphs up to a certain threshold and search them on any given graph using a valid subgraph isomorphism algorithm. Following that, the problem is reduced to representing the order of subgraphs in linear form. The disadvantage of this method is that subgraph isomorphism is an NP complete problem requiring a lot of computing. In addition, there is much ambiguity when generating common subgraphs because some areas of the graphs may not be detected by these subgraphs. Moreover, to produce a unique identifier based on the location of subgraphs, one has to first anchor these subgraphs by providing a canonical labeling of the nodes. This method did not work in part because it was adapted from a different problem requirement. In contrast, it worked well in a graph search problem because graph indexing did not require a strict identity between two graphs as compared to creating unique identifiers. However, this setback allowed us to gain a major insight towards finding unique graph identifiers that it suffices to say that finding a unique graph identifier cannot be done without first providing a canonical labeling of the nodes in a graph.

3.2 The GLR Algorithm

It is essential to first provide a few definitions and concepts based on our aforementioned motivations.

3.2.1 Definitions and Theorems

Definition 3.2.1.1: A node in a graph that has degree higher than 3 is a highly connected node (or hicon in short).

Definition 3.2.1.2: A subgraph neighborhood of a hicon node **a** in a graph is defined as all nodes in all paths starting from node **a** to its nearest hicon neighbor or to a node that terminates at its end.

Definition 3.2.1.3: The condition local symmetry is defined on two hicon nodes **a** and **b** such that the subgraph neighborhood of **a** and **b** is the exactly isomorphic to each other.

Notes on definition 3.2.1.3: If **a** and **b** are symmetric then our goal is to find a characteristic 1-to-1 mapping from the subgraph neighborhood of **a** and **b** to a unique number.

Definition 3.2.1.4: The condition global symmetry is defined on two hicon nodes **a** and **b** such that the graph centered on a and the graph centered on b are isomorphic to each other.

Definition 3.2.1.5: A unique number that is characteristic of the subgraph neighborhood of a hicon node **a**, is called a sufficient function of **a**, such that any other hicon node **b** has its sufficient function producing a same number as that of a if and only if they have the same subgraph neighborhood. In other words, **a** and **b** are locally symmetric if they have the same number.

Theorem 3.2.1.1: The fundamental theorem of arithmetic states that any positive integer can only be represented by only one way as a product of one or more prime numbers.

$$p_1^{\alpha}.p_2^{\beta}.p_3^{\gamma}....=u$$

The theorem can also be written in an equivalent manner.

$$\forall a,b,c,d,\alpha,\beta,\gamma,\delta \qquad p_1^a.p_2^b.p_3^c.p_4^d = p_1^{\alpha}.p_2^{\beta}.p_3^{\gamma}.p_4^{\delta} \Leftrightarrow a=\alpha,b=\beta,c=\gamma,d=\delta$$

Corollary 3.2.1.1: A function that depends on the multiplication of primes can be a sufficient function.

Proof to corollary 3.2.1.1: Let α, β, γ, δ… be the lengths of all possible paths in a subgraph neighborhood of a hicon node **a**, sorted in ascending order, then its sufficient function is $s(a)=2^{\alpha}.3^{\beta}.5^{\gamma}.7^{\delta}...$ From the fundamental theorem of arithmetic, s(a) is a unique 1-to-1 characteristic score of node **a** and hence any other node that has the same score must have the same subgraph neighborhood as that of **a**.

Notes to corollary 3.2.1.1: Note that corollary 3.2.1.1 is only defined on a subgraph neighborhood of **a** and does not imply a stricter global symmetry condition.

Definition 3.2.1.6: A unique number that is characteristic of a hicon node **a** in its global position is called a complete function of **a**, such that another hicon node **b** has that same number if and only if they are globally symmetric to each other.

Corollary 3.2.1.2: A function that depends on the multiplication of primes can be a complete function.

Proof to corollary 3.2.1.2: Let d(x), d(y), d(z)… be the shortest path length from all hicon nodes to the node being evaluated i.e. node **a**. And let s(x), s(y), s(z)… be the sufficient functions of x,y,z… respectively. Then $s'(a)=2^{2^{d'(x)}.3^{s'(x)}}.3^{2^{d'(y)}.3^{s'(y)}}.5^{2^{d'(z)}.3^{s'(z)}}\ldots$ is a complete function where

{(d'(x), s'(x)), (d'(y), s'(y)), (d'(z), s'(z))} are pairs of path length and sufficient functions sorted in lexicographical order. It follows from the fundamental theorem of arithmetic that such a function has a unique number. Any two hicon nodes having the same number from their complete functions will be having the same shortest path length and sufficient score numbers of hicon nodes around it globally. Hence, they must be globally symmetric to each other.

Notes to corollary 3.2.1.2: Notice that although we only proved that any two hicon nodes that have global symmetry must have the same complete function number, there may be cases where non-global symmetry may have produced the same complete function number. However, at this point, no counter-examples could be found. In addition, an observant reader might also realize that the nested exponentials tend to infinity at a very fast rate. We offer an approximate solution taking the logarithm of the exponent term. This may have complicated implications because the fundamental theorem of arithmetic is not applicable to real numbers. However, we also observe that even for any two complete function numbers to be the same, the likelihood is very low because of the following reasons. Firstly, the logarithm function is also a 1-to-1 function; hence the exponent terms are essentially unique. The only cause for non-uniqueness is the multiplication of prime numbers each with an exponent term that is a real number. We cannot prove that it will always be unique but we suggest that the probability of non-uniqueness occurring is relatively low. We also have not found any counter-examples to non-uniqueness of the approximation. We will analyze the effectiveness of using such an approximation at the later sections.

3.2.2 A Review of the GLR Algorithm

At this point, we make the following assumptions. A graph only has single bonds and all elements in the graph are Carbon atoms. We propose that by the same principles of prime multiplication, the sufficient and complete functions can be extended to include multiple bonds and elements. The GLR algorithm takes as input an adjacency matrix representing all the connections in a randomly ordered graph and performs the following steps:

Step1: Identify all hicon nodes from an input adjacency matrix. This can be done by simplying summing up all the horizontal components in the matrix.

Step2: Extract all paths associated to a hicon node that is within its own subgraph neighborhood.

Step3: For each hicon node, compute its sufficient function based on the path extraction.

Step4: Extract shortest path information of all hicon nodes using Floyd's all-pairs shortest path algorithm. This can be done by dynamic programming (Cormen, 1990).

Step5: From step3 and 4, group the computed numbers as a pair associated with their respective hicon nodes. Rank them in lexicographical order.

Step 6: For each hicon node, compute its complete function.

Step 7: Number the graph starting from the hicon node with lowest complete function number. Since we assumed that any two nodes having the same complete function are globally symmetric to each other, it does not matter to choose any one node to number first. Within the subgraph neighbor of any hicon node selected, number from the one with the lowest path score first.

Step8: A depth first traversal can be employed similar to the GENES stage of the SMILES algorithm to obtain a unique identifier. Another way to obtain a unique identifier is to enumerate the edges of the canonical labeled nodes according to the smallest number.

3.2.3 An Analysis of the GLR Algorithm

To analyze the whether the algorithm produces unique identifiers; it suffices to show that a canonical labeling must have been produced by step 7 of the algorithm. As an example, we produce a canonical labeling of the counter-example Grossman had used in figure 4. The graph on the left in figure 4 identifies the hicon nodes labeled as a, b and c. Note that nodes a and c are symmetric to each other. The graph in the middle is a randomly numbered graph and the algorithm always produce the same output graph shown on the right no matter a or c were chosen first.

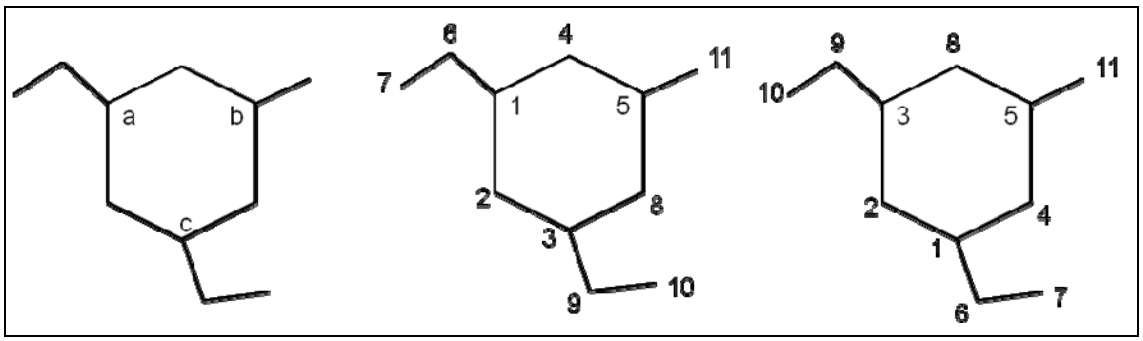


Figure 4b. Canonical labeling using the GLR algorithm. a and c are symmetric.

In contrast with the CANON algorithm, the GLR algorithm offers a unique advantage. By only considering the highly connected nodes, the GLR algorithm reduces the need to compare unnecessary symmetry classes. In figure 1, we showed all the symmetrical classes of the same test graph used above. The CANON algorithm attempts to number the graph from one symmetrical class to another, effectively considering all 7 symmetry classes. In our case, we reduced the number of symmetry classes to be considered to 2, as indicated in figure 4b. We reiterate the problem discussed in section 2 for clarity. In section 2, we have identified that symmetry in graph is an important property since we can choose to number from any two nodes that are symmetrical. The problem with the CANON algorithm only comes in when the rank of any node in a symmetry class needs be reduced consistently if the same node rankings were considered more than once by the algorithm. In our case, the algorithm only considers the nodes in the same symmetry class once. Hence, the consistency problem does not exist in our model.

Under notes to corollary 3.2.1.2 in the previous section, we highlighted two issues with our current model. Firstly, although we had shown that two node that are symmetrical to each other will always have the same score, we do not have sufficient prove to show that any two nodes that are not symmetrical to each other will not have the same the same score. Secondly, the approximation for sufficient and complete functions could have produced non-unique numbers. As a result, we have to resort to empirical testing to show the number of graphs that could be canonically labeled by our algorithm. We provide our test cases taken from the national cancer institute, as 10,000 randomly selected structures out of the total 250,251 structures. Each of the 10,000 structures was permutated 100 times such that they have different initial node orderings. In our test, all 10,000 structures produced the same canonical labeling from all 100 random initial orderings. We have included the first 100 structures and their adjacency matrix after canonical labeling together with the source code of the program for reference. Instructions on how the program should be run are found in appendix B. For a more specific example, we have included a visualization of the test example shown in figure 4b, with the original random labeling as the left graph and the canonical labeling under the GLR algorithm as the graph on the right, in appendix A.

The time complexity of the algorithm is the maximum of time complexity of the 7 steps described in section 3.2.2. If we assume computation of numbers to be of linear time, then the time complexity of the GLR algorithm is bounded by the time complexity required to

perform Floyd's all-pairs shortest path algorithm which is in $O(n^3)$. This is a surprising result because graph isomorphism testing is known to be in NP but neither known to be in polynomial time or NP-complete (Cormen, 1990). We suggest that these could show that our algorithm might fail due to the discussion mentioned in section 3.2.1.2. However, our test results appear to be consistent since all 1000,000 instances of which 10,000 structures were each producing 100 identical canonical labeling. In view of this, we suggest that graph isomorphism testing could be a polynomial time, but have no proof to it.

4 Suggestions for Improvement

The GLR algorithm can be modified to account for all the assumptions suggested in section 3. It can be easy to see that multiple bonds and elements can be included in a sufficient function by making use of nested exponentials as demonstrated earlier. However, the caveat to this is that a logarithm might be necessary to reduce the size of numbers and may have hidden complications as aforementioned.

We can also further improve our algorithm by taking into account other forms of chemical isomers. We show all possible chemical isomers in figure 5 and following, a discussion on what chemical isomers have already been taken into consideration by the GLR algorithm and what remains uncovered.

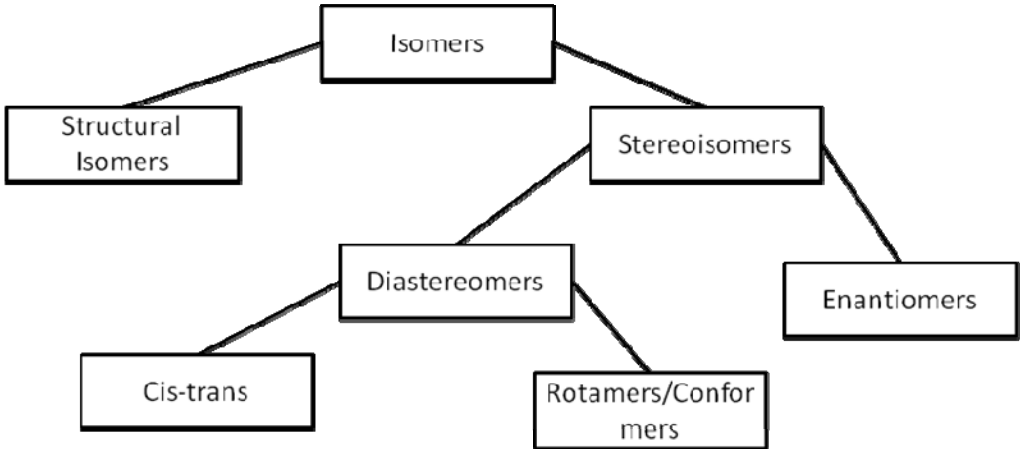


Figure 5. All possible chemical isomers

The GLR has covered structural isomers as it is intrinsically identical to the graph isomorphism model we have introduced. The only other chemical isomers that need to be considered are those of three dimensional chemical isomers for example the enantiomers. These can be also included into the GLR by modifying the sufficient and complete functions respectively. However, this also means that extra input information has to be given because there is no way to differentiate three dimensional structures from an input two dimensional adjacency matrix.

In conclusion, we suggest that the GLR algorithm is flexible because of its ability to adapt to include more features so that complicated chemical structures can also have a unique identifier. The GLR algorithm also produces a useful identifier because the chemical structure can be easily derived from a depth-first traversal list.

5 References

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990). Introduction to Algorithms, first edition, MIT Press and McGraw-Hill, pp. 558-565

D. Tchekhovskoi, S. Stein, S. Heller, The IUPAC Internationl Chemical Identifier (InChI)

David Eppstein, (1995) Subgraph isomorphism in planar graphs and related problems, Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, p.632-640, January 22-24, 1995, San Francisco, California, United States.

David Weininger, Arthur Weininger and Joseph L. Weininger. (1989) SMILES 2: Algorithm for Generation of Unique SMILES Notation, Daylight Chemical Information Systems, Irvine, California 92714, 1989. Note that although the Unique SMILES implementation has been changed by the Daylight Chemical Information System, this appears to be the most recent publication describing the algorithm.

Haoliang Jiang Haixun Wang Yu, P.S. Shuigeng Zhou. (2007) GString: A Novel Approach for Efficient Search in Graph Databases. In proceedings of the Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference, Istanbul, 2007, pp. 566-575.

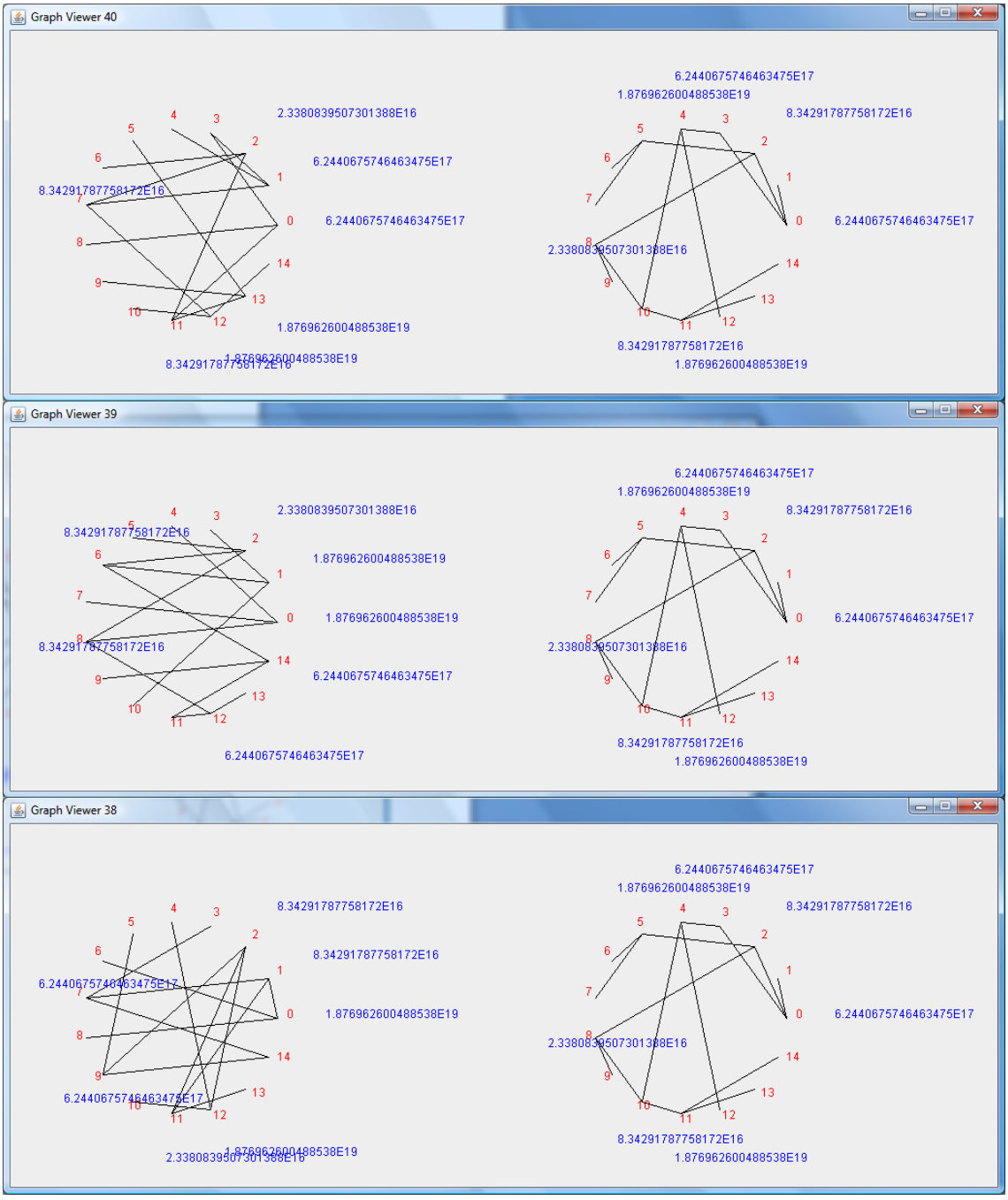
Messmer and Bunke. (1995) Subgraph isomorphism in polynomial time.

Robert Grossman, Donald Hamelberg, Pavan Kasturi and Bing Liu. (2005) Experimental Studies of the Universal Chemical Key Algorithm on the NCI Database Of Chemical Compounds, Proceedings of the 2003 IEEE Computer Society Bioinformatics Conference.

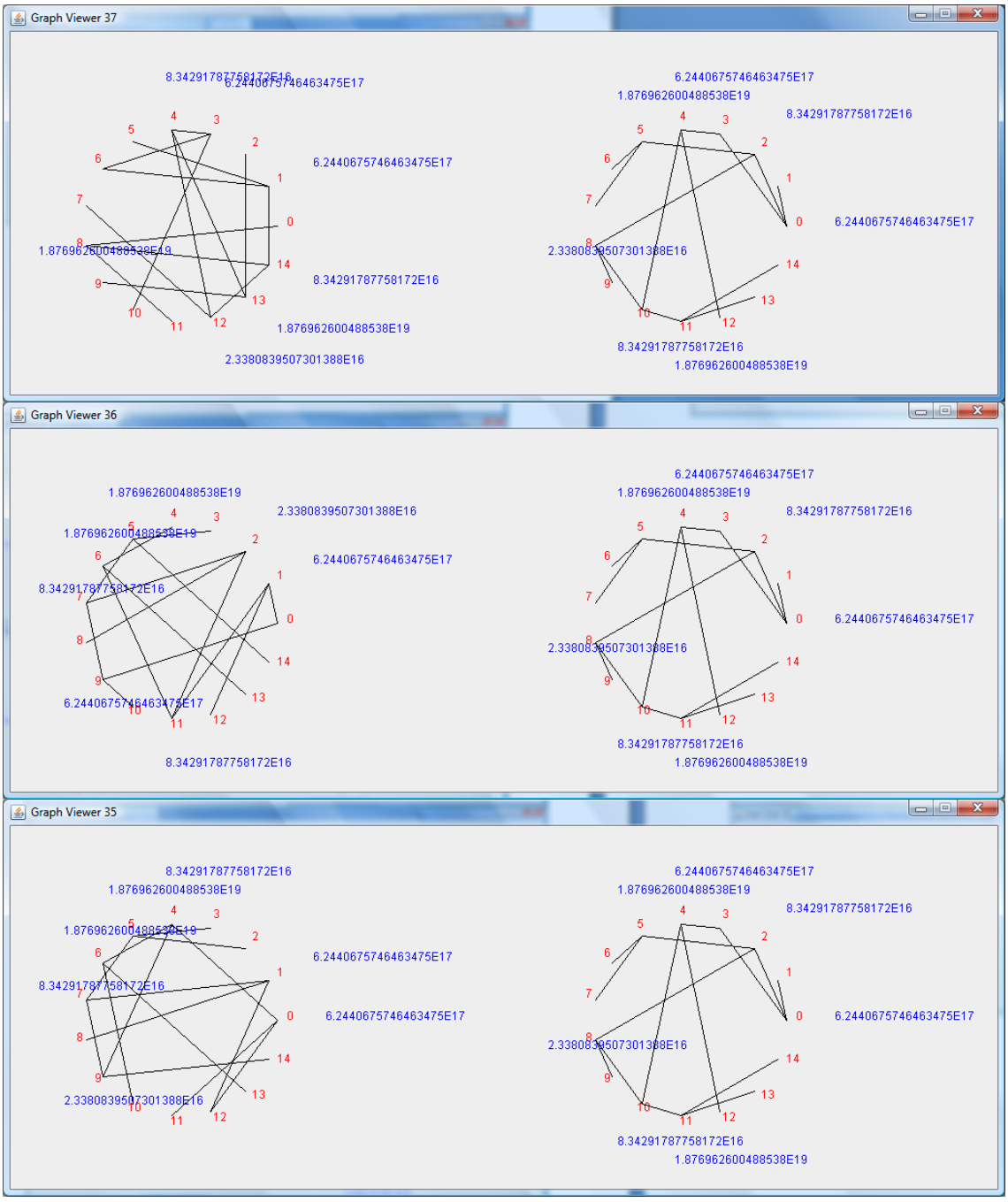
Robert L.Grossman, Greeshma Neglur and Bing Liu. (2005) Assigning Unique Keys to Chemical Compounds for Data Integration: Some Interesting Counter Examples.

X. Yan, P.S. Yu, J. Han. (2004), Graph Indexing: a Frequent Structure-Based Approach, International Conference of Management of Data.

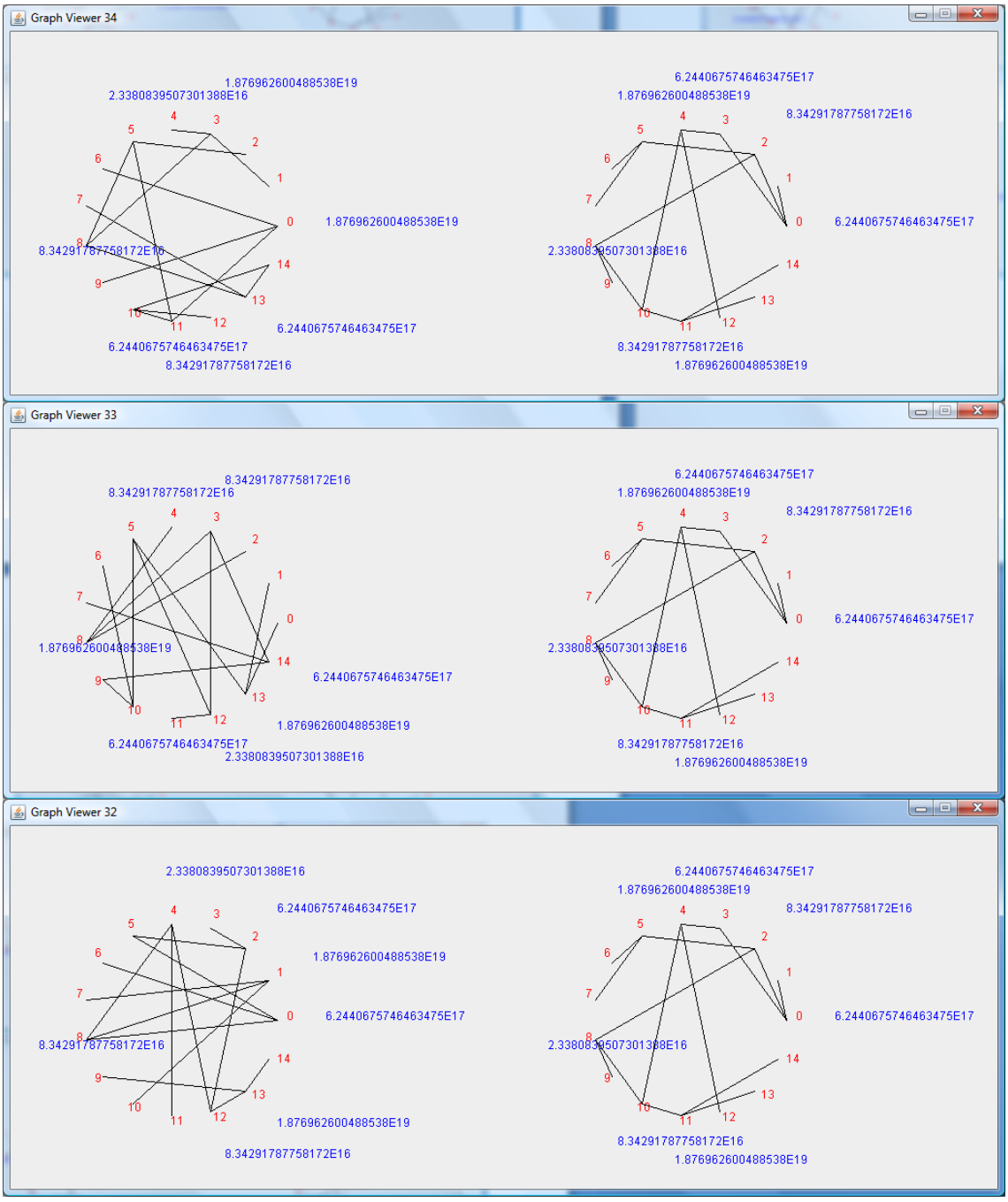
Appendix A



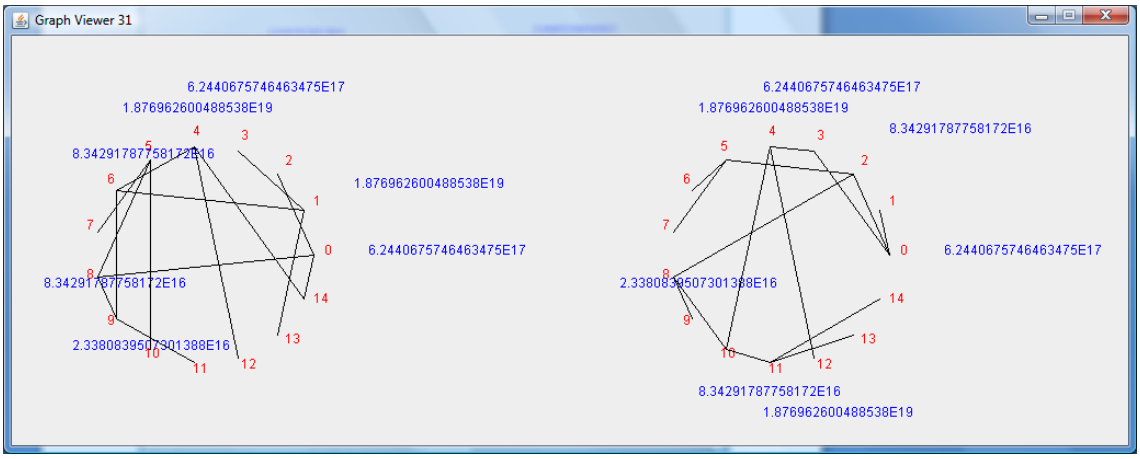
Appendix A



Appendix A


Graph Viewer 34
1.876962600488538E19
2.3380839507301388E16
1.876962600488538E19
8.34291787758172E16
6.2440675746463475E17
6.2440675746463475E17
8.34291787758172E16
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
Graph Viewer 33
8.34291787758172E16
8.34291787758172E16
1.876962600488538E19
6.2440675746463475E17
6.2440675746463475E17
2.3380839507301388E16
1.876962600488538E19
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
2.3380839507301388E16
8.34291787758172E16
1.876962600488538E19
Graph Viewer 32
2.3380839507301388E16
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
2.3380839507301388E16
8.34291787758172E16
1.876962600488538E19

Appendix A


Graph Viewer 31
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
1.876962600488538E19
6.2440675746463475E17
8.34291787758172E16
2.338083950730138E16
6.2440675746463475E17
1.876962600488538E19
8.34291787758172E16
2.338083950730138E16
8.34291787758172E16
1.876962600488538E19

Appendix B

The GLR algorithm can be run as follows. Open Netbeans IDE, File > Open project > Select the GLR folder.

Compile project.

Execute the main file.

Press 1 for a simulation to view graphs.

Press 2 for an automated simulation of 10,000 random structures from nation cancer institute database.

Results can be viewed in the results folder.