CINET GDS-Calculator: Graph Dynamical Systems Visualization

Project Report of

CS 5604: Information Storage and Retrieval

By
Sichao Wu and Yao Zhang

Instructor：Prof. Edward A. Fox

Department of Computer Science

Virginia Tech

December 6, 2012

Abstract

This report summarizes the project of Graph Dynamical Systems Visualization, which is a subproject under the umbrella of project CINET. Base on some input information, we extract the character of system dynamics and output corresponding diagrams and charts that reflect the dynamical properties of the system, so that it can provide an intuitive and easy way for the researchers to analyze the GDSs.

In the introduction section, some background information about the graph dynamical systems and their applications are given in the introduction. Then, we present the requirement analysis, including the task of the project, as well as the challenge we met. Next, this report records the system developing process, including the workflow, user's manual, developer' manual, etc. Finally, we summarize the future work.

This report can serve as a combination of user's manual, developer's manual, and system manual.

1. Introduction

1.1 Background

Graph dynamical systems (GDSs) generalize concepts such as cellular automata and Boolean networks and can describe a wide range of distributed, nonlinear phenomena. It provides a good mathematic model to solve some practical problems.

The graph structure is a natural way to represent interacting entities, agents, brokers, biological cells, molecules, and so on. A vertex v represents and entity, and an edge $\{v, v'\}$ encodes the fact that the entities corresponding to v and v' can interact in some way. An example of such a GDS graph is the social contact network for the people living in some city or geographical region. In this network the individuals of the population are the vertices. There are various ways to connect people by edges. One way that is relevant for epidemiology is to connect any pair of individual that were in contact or were at the same location for a minimal duration on some given day. Clearly, this is a natural structure to consider for the disease dynamics.

Thus, it is of both theoretic and practical interest to investigate the dynamical properties of GDS. In the above example, through analyzing the dynamical properties of GDS, one can figure out how to minimize the spread of epidemics.

This project focuses on the visualization part of GDS calculator, that is, given an existing GDS, we need to visualize the some important dynamical properties as well as certain statistical results so that the researcher can analyze the system dynamics in a easier way.

1.2 Formal description of GDS

A GDS basically contains following core features:

- A finite graph Y,
- A state for each vertex v,
- A function F_v for each vertex v,

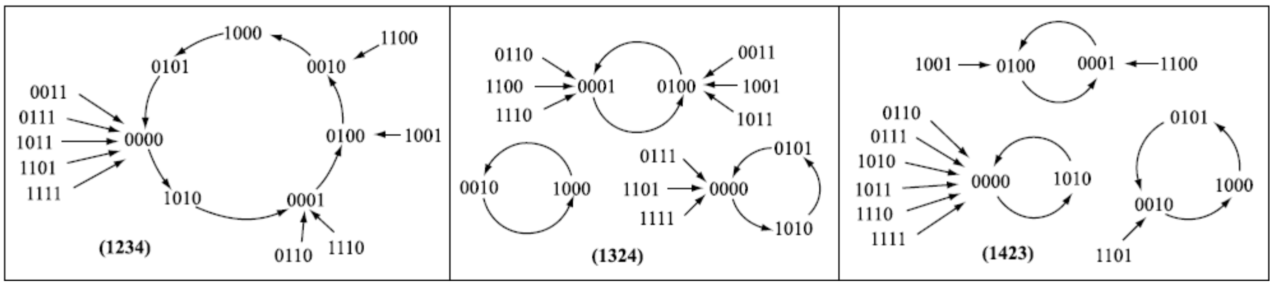
- An update order of the vertices.

In general, an GDS is constructed from a graph Y of order *n*, say, with vertex states in a finite set or field *K*, a vertex-indexed family of functions F_v, and a word update order $w = (w_1, \ldots, w_k)$ where $w_k \in v[Y]$. The GDS is the triple (Y, F_v, *w*), which is a time- and space- discrete dynamical system.

The phase space of a GDS denotes the system state transitions. Since the number of states is finite, it is clear that the phase space is a finite union of finite, unicyclic, directed, graphs. The goal in the study of GDS is to derive as much information about the structure of the phase space as possible based on the properties of the graph Y, the functions F_v and the update order *w*.

1.3 Important dynamical properties of GDS

In most applications, we are interested in how would the system will evolve, and what is the final state of the system. Thus, it is very important to investigate the long term dynamics of the system. As mentioned in the section 1.2, phase space provides us a good way to reflect the long term dynamics of the GDS, and we need to extract some important character of GDS from it. Figure 1.1 is an example of phase space of the graph $Circ_4$ with update function Nor_3



(a) (b) (c)

Figure 1.1. Phase space of Circ_4 with update function Nor_3. (a) The phase space with the update sequence (1234). (b) The phase space with the update sequence (1324). (c) The phase space with the update sequence (1423).

Function equivalence equivalent class (FEEC): Two GDSs belong to the same

FEEC if their phase spaces are identical.

Dynamical equivalence equivalent class (DEEC): Two GDSs belong to the same DEEC if their phase spaces are isomorphic, that is the topological structure of the phase spaces are identical.

Cycle equivalence equivalent class (CEEC): Two GDSs belong to the same CEEC if their phase spaces have identical periodic orbits (have the same cycle structure).

It is straightforward to see that if two GDSs belong to the same FEEC, then they must belong to the same DEEC and CEEC. And if two GDSs belong to the same DEEC, they must belong to the same CEEC. In Figure 1.1, one can observe that none of the GDS phase spaces are functionally or dynamically equivalent, but (b) and (c) are cycle equivalent.

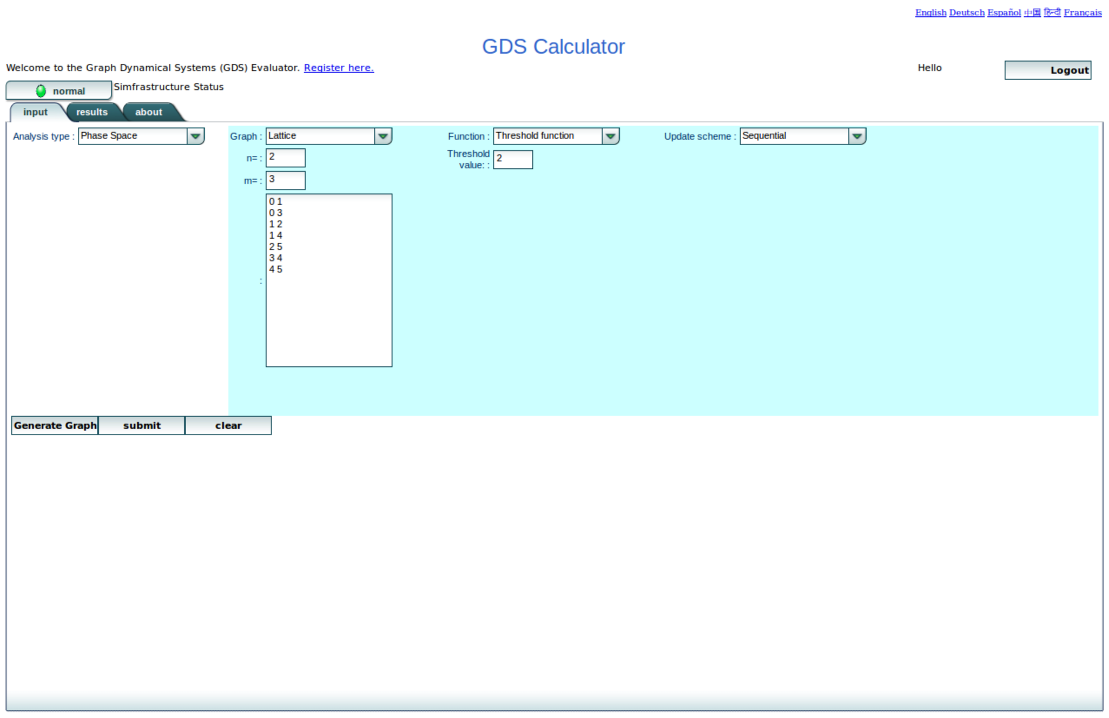
1.4 Project overview

This project is a subproject under the umbrella of CINET, which is developing a cyberinfrastructure middleware to support Network Science. The National Research Council defines Network Science as "the study of network representations of physical, biological, and social phenomena leading to predictive models of these phenomena." This middleware will give Network Scientists access to an unparalleled computational and analytic environment for research, education and training.

Integrated with GDS Calculator, this project focuses on the visualization part. Base on some input information, including the graph, vertex function, update scheme, we extract the character of system dynamics and output corresponding diagrams and charts that reflect the dynamical properties of the system, so that it can provide an intuitive and easy way for the researchers to analyze the GDSs.

2. Requirement Analysis

Figure 2.1 is the user's interface of GDS Calculator. As is shown, the user could choose and set some basic features of the GDS, such as the graph type, the vertex function, the update scheme, etc. Once these features are set, the calculator will generate a XML file containing the dynamical information of the GDS. The mainly task of this project is to parse the XML file and visualize the information extracted from the XML file and draw corresponding charts and diagrams, which will also be integrated with the GDS Calculator.


English Deutsch Español 中国 हिन्दी Français

GDS Calculator

Welcome to the Graph Dynamical Systems (GDS) Evaluator. Register here.

Hello

Simfrastructure Status: normal

Analysis type : Phase Space

Graph : Lattice

n= : 2

m= : 3

```
0 1
0 3
1 2
1 4
2 5
3 4
4 5
```

Function : Threshold function

Threshold value: : 2

Update scheme : Sequential

Figure 2.1 User’s interface of GDS Calculator

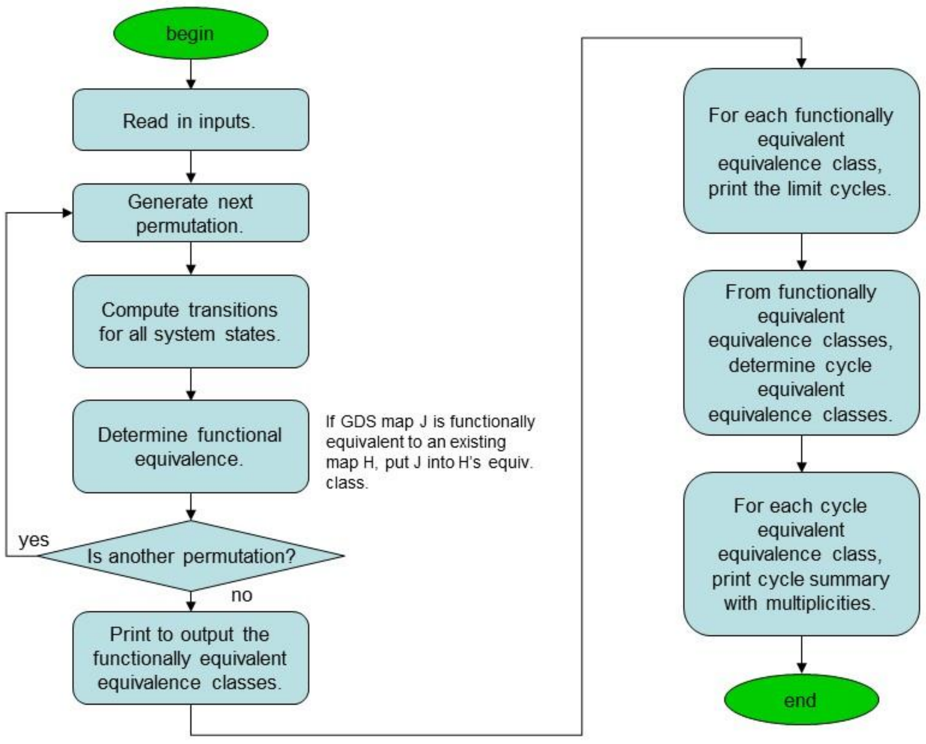
The required diagrams are as following:

a) Number of permutations for each FEEC.
b) Maximum cycle length for each FEEC.
c) Total number of cycles for each FEEC.
d) Draw the phase space for each FEEC
e) The number of permutations for a certain maximum cycle length
f) Draw the cycle structure for each CEEC.
g) Some other diagrams that reflect the long term dynamical properties of the GDS

When these diagrams and charts are obtain, store them as pdf, png and eps. Design a friendly user’s interface to show them and it is also necessary to provide output XML file for download.

3. Workflow

Figure 3.1 shows the workflow of the phase space analysis.



begin
Read in inputs.
Generate next permutation.
Compute transitions for all system states.
Determine functional equivalence.
If GDS map J is functionally equivalent to an existing map H, put J into H's equiv. class.
Is another permutation?
yes
no
Print to output the functionally equivalent equivalence classes.
For each functionally equivalent equivalence class, print the limit cycles.
From functionally equivalent equivalence classes, determine cycle equivalent equivalence classes.
For each cycle equivalent equivalence class, print cycle summary with multiplicities.
end

Figure 3.1 Workflow of phase space analysis

4. User's Manual

This project is part of CINET GDS-Calculator. The potential users should be researches who are interested in graph dynamical systems. The goal of this project is to give researchers a straightforward way of how a graph dynamical system looks like, and help them better understanding the mechanism of phase spaces transit. GDS calculator visualization system has the following tasks: Visualizing graphs like bar chars; exporting graphs to files.

"GDSCalViz.jar" is an excitable file for users. Users only need to click this file to execute our project. "GDS.xml" is the configuration file for users to set up the GDS calculator system. Codes are all stored in “src” folder, and the “output” folder is used to store the graphs and charts that are exported by our project.

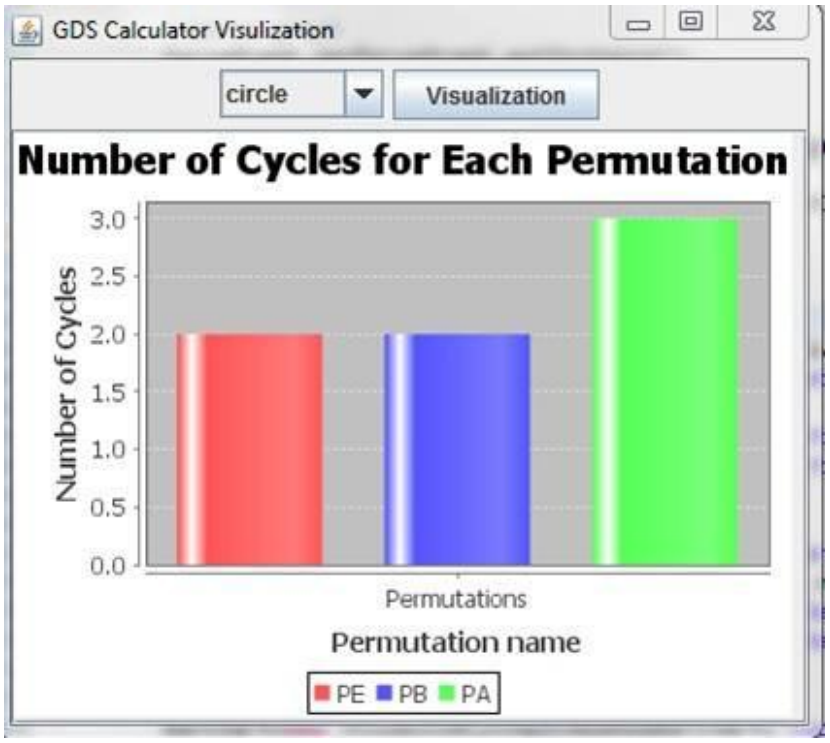


Figure 4.1 User interface for this project

The GUI for users is very straightforward: we provide options for users to visualize some charts. Figure 4.1 shows the GUI for users. Once a user selects which graph they want to visualize, and then he clicks the “visualization” button, the graph will be showed in our user interface.

Currently we have implemented the following bar charts and graphs:

1. Number of permutations for each FEEC.
2. Maximum cycle length for each FEEC.
3. Total number of cycles for each FEEC.
4. Number of permutations for a certain cycle length
5. Number of cycles for each cycle equivalent set.
6. The functional transition state graph.

Furthermore, users can download these graphs and save them in different formats. Currently we provide two formats: png format and pdf format.

Note that our project is part of CINET GDS calculator project which means finally our project should be integrated to the whole CINET GDS calculator system. So this user manual is a manual for temporary use, and the final version will be accompanied with the CINET GDS calculator's user manual.

5. Developer's Manual

The goal of this developer's manual is to help developers to understand how the project is developed including the design of this project, the system implementation and the provided API.

5.1 MVC architecture

The GDS calculator visualization system follows a classical MVC architecture. Figure 5.1 shows the overview of this architecture.

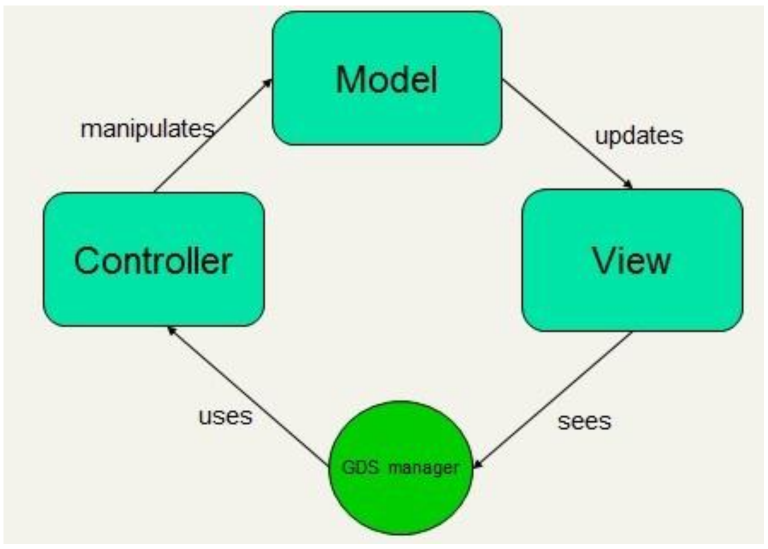


Figure 5.1 Architecture for this project

In MVC architecture, our project has three modules: model module, controller module and view module. The GDS manager uses controller module to manipulate data in the model module. And when the data changes or stores in the model module, the view module will visualize the data in UI. Here are the detailed introductions of three modules.

Model. The model module is responsible for store the data. We mainly have six types of data in the model: permutation, node state, functional transition, cycle transition, functional equivalence and cycle equivalence. Permutation records the node order to execute node functions. For example, if we have four nodes, we should have 16 permutations. Node state records the state when a node applies its node function. It is also a permutation of the node number. Functional transition and cycle transition record how node state transit from one to another. Finally the cycle equivalence and functional equivalence record the equivalent transitions for the GDS system.

Controller. The controller module is used to manage date in the model module. In this project, the controller module has three main tasks: parsing configuration file; analyzing GDS data; exporting graphs to files. First the controller should parse the configuration file to generate data in the models. The configuration file is stored as the XML format. It contains all data we need such as permutations, node states, functional transitions and so forth. Secord since we would like to visualize the charts and graphs to demonstrate some statistical properties for GDS calculator system, the controller will collect and summarize data for visualizing graphs. Finally the controller module is responsible for save graphs as different types of format to files.

View. The view module is used for visualizing graphs. Here we want to visualize five graphs: number of permutations for each FEEC; maximum cycle length for each FEEC; total number of cycles for each FEEC; number of permutations for a certain cycle length; number of cycles for each cycle equivalent set. Also the view module is responsible for drawing user interface. Right now we only need to draw an off-line UI. But the concern for designing view module is to make the system flexible for multiple UI. Once date in the model updates, the view module will show these modifications.

5.2 Use-cases diagram

GDS visualization system has four use-cases: parsing configuration files; managing model data; visualizing graphs; exporting graphs. Figure 5.2 shows the use-case diagram.

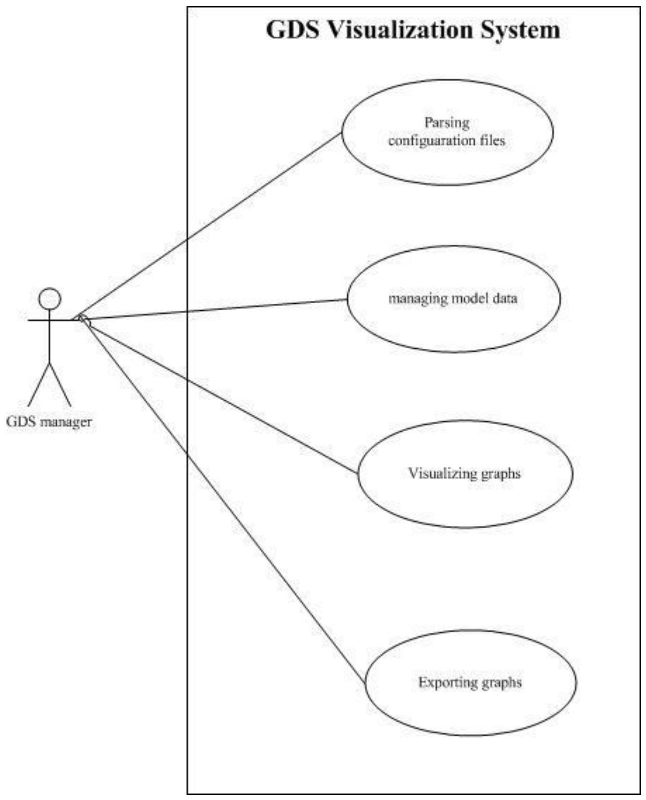


Figure 5.2 Use-case diagram for this project

The controller module manipulates these use-cases. These use-cases follow the order from top to down in figure 5.2: GDS manager parses the configuration file to generate the data in the model; it summarizes the date for statistical analysis; then it visualizes graphs; finally it exports graphs to files.

5.3 Environment and tools

Environment. GDS calculator visualization system is implemented by Java (JDK 1.7). The IDE for implementation is Eclipse Juno.

Tools. Except for JDK 1.7, we mainly use four java package tools: jdom, jfreechart, gephi-toolkit and itext.

Jdom [5] is a Java-based solution for accessing, manipulating, and outputting XML data from Java code, which provides a light-weight means of parsing XML data without the complex options that current solutions like DOM and SAX provide. In our project we use Jdom to parse the configuration date to generate data in the model module.

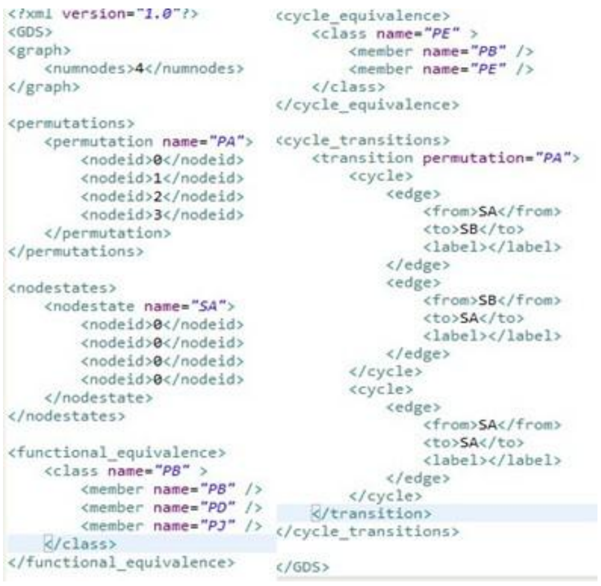
Jfreechart [6] is a free Java chart library that makes it easy for developers to display professional quality charts in their applications. In our project we use it to generate different kinds of bar charts to analyze GDS calculators.

Gephi-toolkit [7] is a standard Java library, which developers can use in new Java applications and achieve tasks that can be done in Gephi automatically. It is a useful tool for graph analysis. We use it to generate graphs for GDS calculators.

Itext [8] is an open source pdf library for Java. It allows developers to create and manipulate pdf documents. In our project we use it to generate pdf files from graphs we generate.

5.4 Configuration file

Figure 5.3 is an example of the configuration file, which should be generated by GDS calculator system. Basically it is an entry for our system, and all data we need has been stored in such XML configuration file. In our project, it is called “GDS.xml”.



```xml
<?xml version="1.0"?>
<GDS>
<graph>
    <numnodes>4</numnodes>
</graph>

<permutations>
    <permutation name="PA">
        <nodeid>0</nodeid>
        <nodeid>1</nodeid>
        <nodeid>2</nodeid>
        <nodeid>3</nodeid>
    </permutation>
</permutations>

<nodestates>
    <nodestate name="SA">
        <nodeid>0</nodeid>
        <nodeid>0</nodeid>
        <nodeid>0</nodeid>
        <nodeid>0</nodeid>
    </nodestate>
</nodestates>

<functional_equivalence>
    <class name="PB" >
        <member name="PB" />
        <member name="PD" />
        <member name="PJ" />
    </class>
</functional_equivalence>
```

```xml
<cycle_equivalence>
    <class name="PE" >
        <member name="PB" />
        <member name="PE" />
    </class>
</cycle_equivalence>

<cycle_transitions>
    <transition permutation="PA">
        <cycle>
            <edge>
                <from>SA</from>
                <to>SB</to>
                <label></label>
            </edge>
            <edge>
                <from>SB</from>
                <to>SA</to>
                <label></label>
            </edge>
        </cycle>
        <cycle>
            <edge>
                <from>SA</from>
                <to>SA</to>
                <label></label>
            </edge>
        </cycle>
    </transition>
</cycle_transitions>

</GDS>
```

Figure 5.3 configuration file for this project

We have the following labels in our configuration file:

<permutations> stores the order to execute the node functions.

<nodestates> stores the states for each node. Figure 5.3 is an example with {0, 1} states.

<functional_equivalence> stores all equivalent permutations for functions.

<cycle_equivalence> stores all equivalent permutations for cycles.

<functional_transitions> stores all state transitions.

<cycle_transitions> stores all state transitions that have cycles.

This configuration file should be strictly generated by CINET GDS calculator system. Since CINET GDS calculator system is developing, currently we use a four-node graph dynamical system as our test case.

5.5 Class diagram

Figure 5.4 shows the class diagram for the model module. It implements all types of data discussed in the model module.

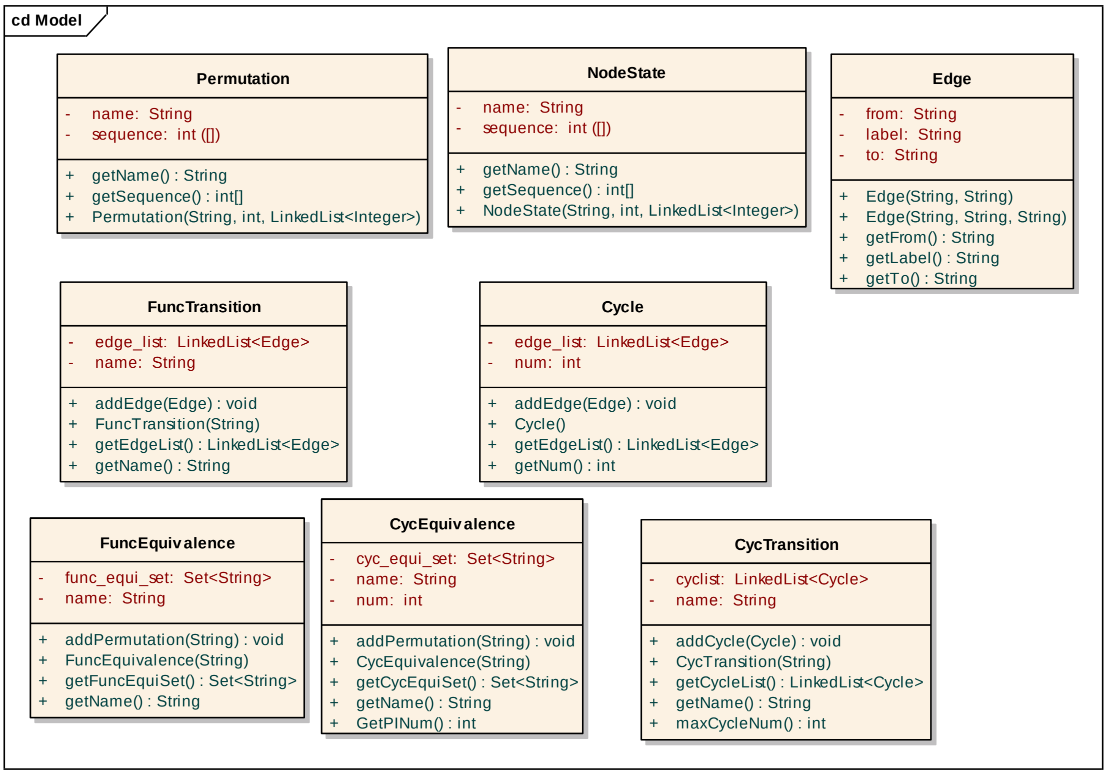


Figure 5.4 class diagram for model module

Figure 5.5 shows the class diagram for the controller module and view model. Three classes "ParseGraph", "ModelManager" and "VisualizeBarChart" implements use-cases in the controller model. It is obviously that “ParseGraph” is responsible for parsing XML graph. “VisualizeBarChart” is implemented for visualizing charts and exports charts to files. “ModelManager” is used to manage the controller model, which uses the Hashtable to store the data parsed by “ParseGraph”. “Visualzie BarChart” calls “ModelManager ” to use data to generate graphs.

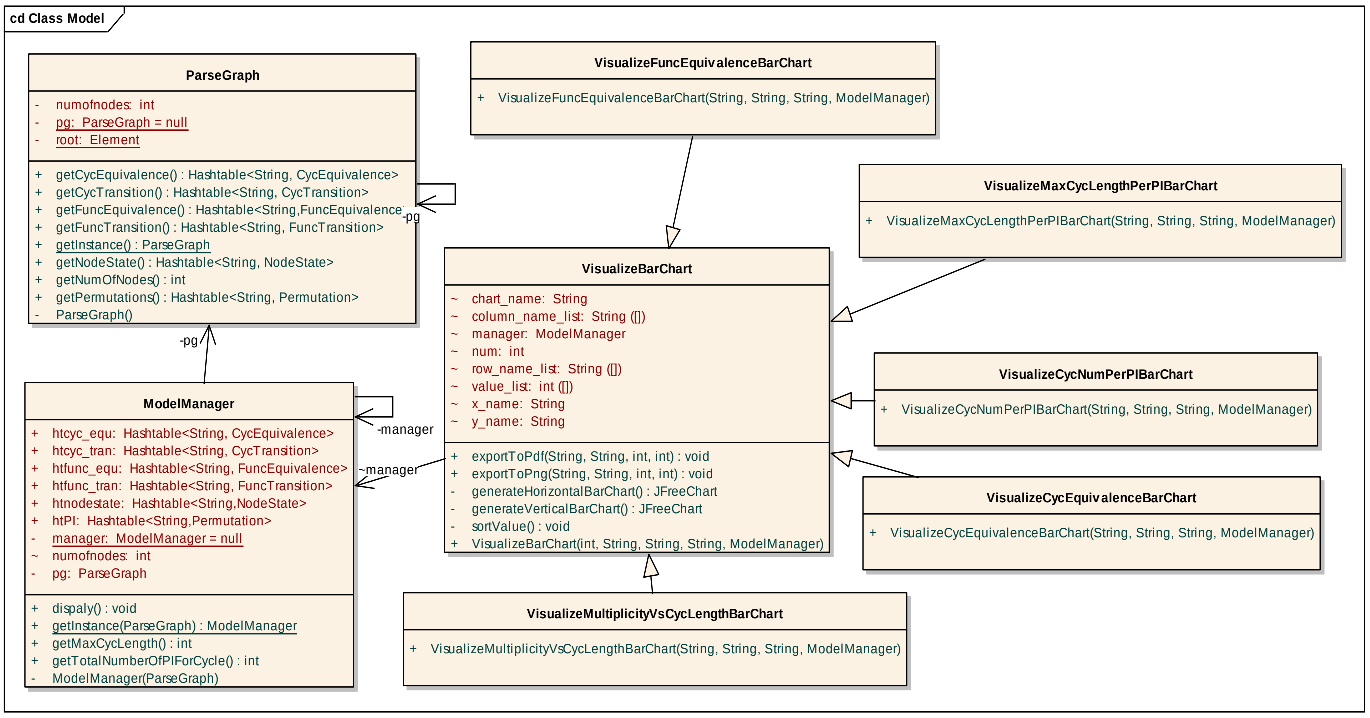

cd Class Model
ParseGraph
- numofnodes: int
- pg: ParseGraph = null
- root: Element
+ getCycEquivalence() : Hashtable<String, CycEquivalence>
+ getCycTransition() : Hashtable<String, CycTransition>
+ getFuncEquivalence() : Hashtable<String,FuncEquivalence>
+ getFuncTransition() : Hashtable<String, FuncTransition>
+ getInstance() : ParseGraph
+ getNodeState() : Hashtable<String, NodeState>
+ getNumOfNodes() : int
+ getPermutations() : Hashtable<String, Permutation>
+ ParseGraph()
-pg
-pg
ModelManager
+ htcyc_equ: Hashtable<String, CycEquivalence>
+ htcyc_tran: Hashtable<String, CycTransition>
+ htfunc_equ: Hashtable<String, FuncEquivalence>
+ htfunc_tran: Hashtable<String, FuncTransition>
+ htnodestate: Hashtable<String,NodeState>
+ htPI: Hashtable<String,Permutation>
- manager: ModelManager = null
~ numofnodes: int
- pg: ParseGraph
+ dispaly() : void
+ getInstance(ParseGraph) : ModelManager
+ getMaxCycLength() : int
+ getTotalNumberOfPIForCycle() : int
- ModelManager(ParseGraph)
-manager
~manager
VisualizeFuncEquivalenceBarChart
+ VisualizeFuncEquivalenceBarChart(String, String, String, ModelManager)
VisualizeMaxCycLengthPerPIBarChart
+ VisualizeMaxCycLengthPerPIBarChart(String, String, String, ModelManager)
VisualizeBarChart
~ chart_name: String
~ column_name_list: String ([])
~ manager: ModelManager
~ num: int
~ row_name_list: String ([])
~ value_list: int ([])
~ x_name: String
~ y_name: String
+ exportToPdf(String, String, int, int) : void
+ exportToPng(String, String, int, int) : void
- generateHorizontalBarChart() : JFreeChart
- generateVerticalBarChart() : JFreeChart
- sortValue() : void
+ VisualizeBarChart(int, String, String, String, ModelManager)
VisualizeCycNumPerPIBarChart
+ VisualizeCycNumPerPIBarChart(String, String, String, ModelManager)
VisualizeCycEquivalenceBarChart
+ VisualizeCycEquivalenceBarChart(String, String, String, ModelManager)
VisualizeMultiplicityVsCycLengthBarChart
+ VisualizeMultiplicityVsCycLengthBarChart(String, String, String, ModelManager)

Figure 5.5 class diagram for controller module and view model

For the view module, “VisualizeBarChart” is a superclass for the specific visualizations, which implements most part of visualization. All its subclass should only implement their construct method to generate the data for “VisualizeBarChart”. And for the state transition graph, we have two classes “Visualize Functional State Graph” and “Visualize Cycle State Graph”. This implementation is flexible for developer to visualize other graphs.

5.6 Installation and Extension

“GDSCalViz.jar” is an excitable file for users. Users only need to click this file to execute our project.

“GDS.xml” is the configuration file for users to set up the GDS calculator system.

"src" is the folder to store all codes.

"bin" is the folder to store excitable classes and packages we use.

"output" is the folder to store charts and graphs that are exported by our project.

There are four packages in our project:

"edu.vt.vbi.ndssl.gdscalvis": the entry for our project.

"edu.vt.vbi.ndssl.gdscalvis.controller": the controller module.

"edu.vt.vbi.ndssl.gdscalvis.model": the model module.

"edu.vt.vbi.ndssl.gdscalvis.view": the view module.

Developers can use model, view and controller module directly only if they build path for jdom, jfreechart, gephi-toolkit and itext packages.

6. Future Work

We have mainly three tasks for future work:

1. Improving GUI implementation. In future we plan to display more contents for user interface, such as how the configuration file looks like.
2. Visualizing more statistical charts. We believe that the more statistical charts, the better understanding of graph dynamical systems for users. So we plan to visualize more charts such as chart for betweeness trends and scale-free factor trends.
3. Integrating with the whole CINET GDS project. Our project is part of CINET GDS project. So after CINET GDS has been developed, we will integrate our project with CINET GDS project.

7. Acknowledgments

First authors would like to thank Professor Edward A. Fox and the teaching assistants of CS 5604 for providing this interesting course. We learn a lot from it. Both authors are also grateful to Chris Kuhlman and Sherif Abdelhamid in NDSSL for the helpful discussions.

References

[1] H. S. Mortveit and C. M. Reidys, An introduction to sequential dynamical systems, New York: Springer, 2007.

[2]M. Macauley and H. S. Mortveit, Cycle equivalence of graph dynamical systems, Nonlinearity, 22, 421-436, 2009.

[3] S. Eubank, H. Cuclu, V. S. Kumar, M. V. Marathe, et al., Modelling disease outbreaks in realistic urban social networks, Nature, 429, 180-184, 2004.

[4] H. S. Mortveit and C. M. Reidys, Discrete, sequential dynamical systems, Discrete Math., 226, 281-295, 2001.

[5] www. jdom.org.

[6] www.jfree.org/jfreechart/.

[7] www.gephi.org/toolkit/

[8] www.itextpdf.com.