A FRAMEWORK FOR PROBABILISTIC ANALYSIS OF PROGRAMS

by

Paul Douglas Hein

A Thesis Submitted to the Faculty of the

DEPARTMENT OF COMPUTER SCIENCE

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2019

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Master's Committee, we certify that we have read the thesis prepared by Paul Douglas Hein, titled A Framework for Probabilistic Analysis of Programs and recommend that it be accepted as fulfilling the thesis requirement for the Master's Degree.

Date: 3 May 2019

Mihai Surdeanu

Date: 3 May 2019

Clayton Morrison

Date: 3 May 2019

Richard Snodgrass

Final approval and acceptance of this thesis is contingent upon the candidate's submission of the final copies of the thesis to the Graduate College.

I hereby certify that I have read this thesis prepared under my direction and recommend that it be accepted as fulfilling the Master's requirement.

Date: 3 May 2019

Mihai Surdeanu
Associate Professor
Department of Computer Science

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The pervasiveness of the use of scientific models in all fields of research has led to model selection being a critical and challenging component of the scientific process for modern day research. Given a phenomena of interest, the model selection problem includes the following challenges: identifying which models exist, accessing the models in a form that allows for observation and experimentation, and comparing the models by some metric in order to determine which model is best suited for a given set of experimental criterion. Modern day researchers are required to overcome all of these challenges with little computational aides if they wish to be certain that the models they are using to observe phenomena do represent the best that the state-of-the-art research in their field of study has to offer. In this thesis I will present a computational tool that automates the process of extracting scientific models that are present in scientific source code, grounding the extracted models to ancillary documents, and efficiently analyzing the models both individually and comparatively to facilitate automatic model selection.

CHAPTER 1

INTRODUCTION

When studying a given natural phenomena, many researchers turn to modeling as a method to formalize their reasoning about the phenomena. Models have many advantages for researchers, including:

1.) They allow researchers to clearly communicate the relations between variables observed to be associated with a phenomena. This gives the researchers the ability to reason about uncertainty within a model while incorporating information from past experiments as background knowledge.

2.) They are exportable, comparable, and updatable. One researcher can use the model of another, competing models can be compared, and under-performing components of a model can be updated upon discovering new information about the model.

These advantages have been the driving force that has pushed the scientific community towards models as a method of communication of scientific research. However, modeling is not without its challenges. One of the new issues facing scientific researchers is the sheer prevalence of models. Model selection is now a task facing modelers in any field of research. Not only do scientists have to explore many competing models when deciding which to use to model a particular phenomena, this exploration is also expensive. In the information age, most of these models exist as source code with associated grounding documents. However, with the extreme prevalence of programming languages many competing models for the same phenomena are likely written in different programming languages. Asking scientist to learn a single programming language is already a large drain on research time, but the prospect of needing to learn multiple languages represents a large barrier to entering the realm of model selection. Given the enormous amount of items competing for the limited time of scientists the task of model selection commonly is side-lined.

In this thesis I will present an automated system that is able to extract scientific models from source code, ground the models using information gained from associated texts, and finally automate the task of model selection. Accomplishing this task will further unlock the potential of models to revolutionize the objective study of naturally occurring phenomena.

Below we see an example of a model being used to study the yield of crops by observing changes in multiple input variables.

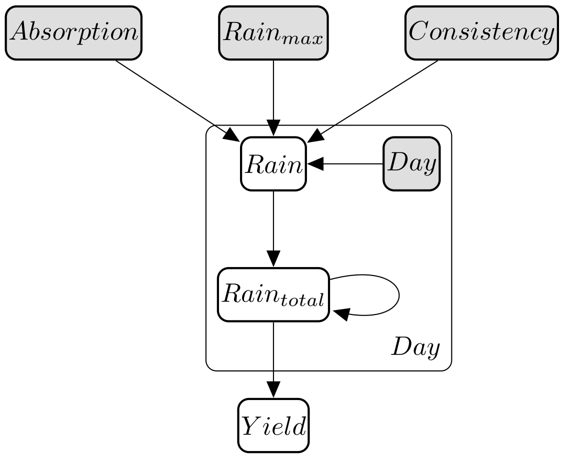

Figure 1.1: A model depicting the affects of rain over a span of days given observed values for absorption and consistency constants.

1.1 Problem Definition

Some text.

1.2 This Work

Some text.

CHAPTER 2

MODEL EXTRACTION FROM SOURCE CODE

In order to perform analysis on scientific models found in software we must first build a system that is able to detect source code that corresponds to scientific models, extract that necessary associated source code, and then represent the model in a form that allows analysis to take place. In the following sections I will outline the AutoMATES systems approach for handling this problem.

2.1 Grounded Function Network Extraction

Some text about extracting lambda functions and a GrFN spec from source code.

2.2 Computation Graph Generation

Some text about creating GrFN CGs.

2.2.1 Assignment Statements

Assignment statements can be handled by loading the assignment statement found in the source code into the function node so that values can propagate from the input variable nodes to the output variable node during computation graph execution.

2.2.2 Conditional Statements

Conditional statements are handled via a set of two lambda evaluations. The first evaluation is known as a `conditional` function node, that will actually evaluate the conditional property. The second is a `decision` function node that takes as input the evaluation from the `condition` function as well as the two possible assignments for an output variable. The `decision` function will be responsible for assigning the appropriate value to the output variable node based upon the conditional input.

Both the `decision` function node and the `conditional` function nodes output variable node are artifacts that did not exist in the original source program that have been added to the computation graph. Thus these will not be displayed when rendering the function node or variable node views of the computation graph.

2.2.3 Function Calls

Function calls can be thought of as containers, notated as plates when drawing a DBN. The only intricacy here deals with the correct wiring of variable inputs into the containers plate, and the correct wiring of outputs from the container to the current position in the computation graph.

2.2.4 Indexed Loops

Indexed loops require a loop plate and have a specific index variable as well as a number of iterations through the loop. They can easily be handled like containers as mentioned above, but require additional storage to handle information about the number of executions needed to satisfy the plate during compuation.

2.2.5 Open-ended Loops

Open ended loops are hard. Here we can have conditional exit cases defined at a start or end point of a loop, which presents a much larger challenge than loops with an index and pre-defined amount of iterations.

An extra challenge is added when dealing with open-ended loops that can include multiple exit points (introduced either by `break` statements or `goto`s) as well conditional skip points where parts of the loop are skipped on an iteration (introduced either by `continue` statements or `goto`s).

2.2.6 Unstructured Branching

The usage of the `goto` statement has been hotly debated by computer scientists for nearly half a century. In most modern programming languages the usage of `goto` or other such statements that allow for unstructured branching is prohibited. However, the AutoMATES system seeks to handle source code inputs from languages that do allow for unstructured branching, and thus this paradigm must be handled during the wiring phase of a GrFN computation graph.

2.2.7 Recursive Functions

Recursion is a commonly used software practice that must be handled for our computation graphs. Most importantly, recursion must be identified and recursive edges that would create loops in the computation graph must be pruned.

2.2.8 Indirect Recursive Functions

Possibly the most difficult challenge for our graph wiring is the identification and handling of indirect recursion.

2.3 Call Stack Creation

Creating a computation graph from a GrFN specification allows us to formally represent an extracted scientific model as a graph data structure. However, if we wish to analyze the extracted model, then we will need the ability to compute information over this data structure. To accomplish this we introduce the idea of execution over a computation graph. The computation graph contains a set of function nodes. Computing the lambda function stored at each function node is analogous to executing the computation graph from the set of inputs to the output. However, the function nodes rely upon having values populated at each of their input variable nodes in order to perform their computation. Therefore the task of executing a GrFN CG can be rephrased as determining how to order and execute the functions nodes contained in the computation graph.

A Naïve first-pass solution to accomplish this goal would be to use a graph traversal from the output to the inputs where at each function node, the node will determine whether values for each input variable node have been populated. For any input variable nodes that have not been populated, the function node will call the parent function node responsible for computing the value of the input variable node. Once all such calls have returned, the function itself will evaluate. This recursive calling procedure is very similar to message-passing, a method for inference on factor graphs. While this will ensure correct model execution, this method of handling execution is not as efficient as possible. To start the recursive call structure adds additional function setup and calls to the execution, on the order of the number of functions included in the computation graph.

The computation graph has the form of a factor graph with variable and function nodes, such that no variable node is adjacent to another variable node and vice-versa for a function node. Therefore representing a computation graph in terms of just the contained function nodes

2.4 Variable Domain and Range Detection

2.5 Vectorized Computation Graph Execution

2.6 Discussion of Time and Space Complexity

CHAPTER 3

GRAPH-BASED METHODS OF MODEL COMPARISON

While analytical methods can provide a large amount of useful information to modelers about a single model, the real benefit of the AutoMATES system comes from the ability to automate comparisons among competing models. Competing models can be identified from the output variables of their computation graphs. After identifying a selection of competing models the comparison phase can begin.

For any two competing models of the same phenomena the comparison phase consists of the following:

1.) Identify the overlap between the variables in each models computation graphs. This corresponds to overlap in observable real-world phenomena.

2.) Extract the sub computation graphs for each model based on the variable node overlap.

3.) Perform analysis on the overlapping computation graphs and compare the results with the analysis results from the models whole computation graphs.

In order to accomplish the tasks outlined above, I will introduce a new construct, known as a Forward Influence Blanket (FIB). A FIB is a specific instance of a Markov blanket, derived from a GrFN computation graph, that can be used for forward analysis. After the completion of these tasks the information gained from the comparative study of these models can be added to the final model report, or used for automatic model selection. In this chapter I discuss model comparison in terms of two models compared directly with one another. At the end of the chapter I will elucidate on the necessary steps to generalize this form of binary model comparison to a set of N models.

3.1 Forward Influence Blanket Creation

Imagine the structure of two computational models of the same phenomena in the most general sense. We can say with certainty that both models will have the same output variable, namely the variable that represents the phenomena of interest. From this there are three options for how the set of input variables between the two computational graphs can overlap. The least interesting option is that the two models could share no inputs variables. This would mean that the computations involved in each model are wholly independent and could be combined if necessary in a trivial manner, at least at the input level. The more interesting option is that a subset of the inputs are shared between the two models. This entails that the models will make use of the same data, though the computations used to transform that data into a model output will almost certainly differ. It is possible that the set of input variables will overlap exactly between the two models; however, it is much more likely that there will be some input variables that are not contained in both. In the following subsections I will discuss how to build a computation graph that represents the computation present in a GrFN that corresponds to utilization of shared variable nodes with another model.

3.1.1 Shared Structure Identification

Some text.

3.1.2 Cover Set Variables

The key aspect of a FIB that distinguishes it from a GrFN computation graph is that the portions of the original computation graph that are not shared between the two models under comparison are pruned. In order to ensure that the resulting models are still executable, variable nodes representing new inputs to the FIB computation graph must be retained. We have identified this set of variables as the cover variable set.

Identifying a variable as being a member of the cover set stems from the initial shared graph structure extracted from the original GrFN computation graphs. From the

3.1.3 Forward Influence Blanket Execution

In order to execute a FIB the user must provide values for the input variable nodes and values for the cover variable nodes. At execution time, both sets of variable nodes will be populated before beginning to compute the function nodes in the partial order of functions provided by the GrFN computation graph. This is the only difference between computing on a FIB computation graph and computing on a GrFN computation graph.

Execution of a FIB computation graph can be done either with singular preset values for all of the cover variables, or with ranges for each cover variable. FIB computation graph supports Torch-aided vectorized computation similar to the GrFN computation graph structure, and no additional memory constraints are imposed on execution by the FIB class.

3.2 Analysis Methods for a Forward Influence Blanket

Some text.

3.2.1 Comparative Sensitivity Index Assessment

Some text.

3.2.2 Cross-model Sensitivity Surface Examination

Some text.

CHAPTER 4

VARIATIONAL ANALYSIS OF MODEL UNCERTAINTY

The greatest benefit researchers receive from modeling is being able to reason about the uncertainty involved in observing a phenomena of choice. From the modeling perspective, explicit statements about the uncertainty of a phenomena can be made by adding inputs to the model of the phenomena that represent a source of variance upon the phenomena. A powerful tool used by modelers to quantify the uncertainty present in models is sensitivity analysis. Broadly speaking, sensitivity analysis is the study of how variance in the inputs of a model affect the variance, or uncertainty in the output of the model.

4.1 Sampling Techniques

Some text about the basic method for sampling from the inputs to a GrFN given little information.

4.1.1 Saltelli Sampling

This section should introduce Saltelli sampling and go into great detail on the process.

4.1.2 Data Informed Sampling Techniques

This section should introduce the methods that we can use to sample from pre-existing data files associated with the codebase that we are working with.

4.1.3 Grounded Sampling Techniques

This section should go into detail on how we can use the grounded nature of our variables to get information about how they exists in the real world that we can use to constrain the domains for our inputs.

4.2 Sensitivity Analysis

Some text that discusses our methods for conducting Sensitivity analysis of our extracted GrFNs.

4.2.1 Variance Based Analysis

This section will go in depth on the Sobol method of Sensitivity analysis.

4.3 Output Surface Analysis

This section will introduce the idea of generating output surfaces that scientists can view and interact with in order to better understand the sensitivity of their models based upon key inputs. The choice in which surfaces to show will be directed by sensitivity analysis.

CHAPTER 5

METHODS FOR INFORMED MODEL SELECTION

Some intro about the task of selecting a model given the information from model analysis

5.1 Model report generation

Some text

5.2 Input space partitioning

Some text

CHAPTER 6

RELATED WORK

Some intro.

6.1 Overview

Some text.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Some intro.

7.1 Conclusions

Some text.

7.2 Future Work

As discussed above, AutoMATES project has created an excellent framework for the extraction and comparison of scientific models found in source code. While many methods for analysis already exist in the AutoMATES system, as well as information necessary to facilitate automated model selection, there are still plenty of directions for future work. Many of the opportunities for extending the AutoMATES system build upon one another, and all are focused on expanding the scope of modeler questions that AutoMATES is able to handle without additional input from the modeler. Below I catalog some of the immediately visible extensions to the AutoMATES program improve the power of the AutoMATES model selection capabilities.

7.2.1 Model Selection via Uncertainty Analysis

The current analysis methods employed by AutoMATES allow for automated model selection based upon behavior of model inputs or sets of model inputs. While this advanced capability is likely desired by modelers in many situations, it only allows for indirect comparison between models. A method for direct comparison such as error propagation that includes an estimate of metrics such as variance in model output allows for stronger comparison statements that will likely be more acceptable metrics for automated model selection.

7.2.2 Iterative Model Improvement

After gathering enough information about various competing models as well as error information of model improvements,AutoMATES should be able to begin learning how to update models to lower uncertainty in model outputs.

A key component to this enhancement would be to identify similar function nodes or series of function nodes in a computation graph that correspond to the same overall computation. This, along with the grounding of variable nodes, which has been assumed, will enable modular computation components for variables to be added, removed, or mutated in order to improve model accuracy, efficiency, or other metrics of choice to modelers.

7.2.3 Input Space Division

Modelers would likely benefit from the AutoMATES being able to answer more general questions about what models to use to study a certain phenomena. For instance, modelers may not be able to provide bound information for the variables of interest to them when gathering data to study a particular phenomena. AutoMATES could assist modelers in this regard by discovering the furthest possible extent of all possible variables for each competing model of a phenomena and then partition the input variable space based upon peak model performance, such that each separate partition has an identified ideal model for studying the phenomena given inputs contained in that range. As previously stated the partition criterion would be some aspect of model fitness.

7.2.4 Data Space Discovery

An extension of the idea of automatically discovering the input bounds of a set of input variables for a model is the idea of discovering the total possible data space for a phenomena of interest. Modelers would benefit from future versions of AutoMATES being able to identify potential additional variables for a given phenomena, other than just those identified by the modeler. A potential example would be the identification of a combination of variables that can be used to model a given model input with higher precision.

APPENDIX A

Sample Appendix

Stuff.....

REFERENCES