5-1-2009

A graph-based factor screening method for synchronous data flow simulation models

Gregory Tauer

ROCHESTER INSTITUTE OF TECHNOLOGY

A GRAPH-BASED FACTOR SCREENING METHOD FOR SYNCHRONOUS DATA FLOW SIMULATION MODELS

A Thesis

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Industrial Engineering

in the

Department of Industrial and Systems Engineering

Kate Gleason College of Engineering

by

Gregory W. Tauer

B.S., Industrial Engineering, Rochester Institute of Technology, 2009

May, 2009

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Gregory W. Tauer
has been examined and approved by the
thesis committee as satisfactory for the
thesis requirement for the
Master of Science degree

Approved by:

Dr. Michael E. Kuhl, Thesis Advisor

Dr. Moises Sudit

Abstract

This thesis develops a method for identifying important input factors in large system dynamics models from an analysis based on those models' underlying structures. The identification of important input factors is commonly called factor screening and is a key step in the analysis of simulation models with many input parameters. Models under investigation are system dynamics models implemented as synchronous data flow programs, a model of computation that requires encoding the model components' dependencies in a graph format. The developed method views this graph as a stochastic process and attempts to rank the importance of inputs, or source nodes, with respect to an output, or non-source node. This ranking is accomplished primarily through the use of weighted random-walks through the graph. A comparison is made against other factor screening techniques, including fractional factorial experiments. The presented structure-based method is found to be comparably accurate to statistical factor screen experiments at magnitude order ranking. Run time of the developed method compared against a resolution III fractional factorial design is found to be similar for small models, and significantly faster for large models.

Contents

List of Figures

List of Tables

Acknowledgements

During my work on this thesis, I received advice and help from a number of individuals. I thank everyone who assisted me in this work. I especially thank the two members of my thesis committee: my advisor, Dr. Michael Kuhl, for his advice and Dr. Moises Sudit for his insight.

I also thank Kevin Costantini for his assistance in developing the software used for the representation and manipulation of graphs, and Pat Ciambrone for his help in developing the software used to generate 2^k factorial experiments.

Chapter 1

Introduction

Simulation is a widely used tool for the modeling and analysis of complex systems. Unfortunately, complex systems often require large and complex simulation models to represent their behavior. Since large simulation models can be time consuming to evaluate, there exists an incentive to decrease the number of simulation replications required by any experimentation or optimization to be performed on such models.

Simulation is often used as a tool to answer questions about the result of making changes to a system. A simulation model can be viewed as a function where a set of input factors, which may or may not be controllable in the real system, are controlled in the simulation to obtain a set of output values. Although the number of input variables in a simulation may be very large, not all input variables are likely to have equal effects on the output variables (Montgomery, 1979). Factor screening is the process of identifying which input variables have a meaningful effect on a set of output variables without necessarily determining the exact nature of those effects (Watson, 1961). This knowledge allows for the asking of more efficient "what-if" questions by allowing analysts to narrow their focus.

Although interesting alone, the results of factor screening are especially valuable when used to improve the efficiency of optimization or further experimentation on a model. As the number of considered input variables is reduced, the execution time of many simulation optimization algorithms quickly increases, and the number of required replicates in a factorial experiment decreases exponentially.

Factor screening is often carried out as a designed experiment on a simulation model. When carried out in this way, factor screening requires a number of dedicated simulation replications. Although tasked with decreasing the cost of additional study, a designed statistical screening experiment may itself be impractically expensive if the simulation model under investigation is large. It would be advantageous to either replace, or improve the efficiency of, a screening experiment with a factor screening method that does not require as many lengthy and expensive simulation executions.

A specific class of system dynamic simulation models, implemented as synchronous data flow programs, will be examined. These programs are used in a variety of applications. For one, they can conveniently represent models described by a system of difference equations. Difference equations are widely used tools to describe discrete dynamic systems, or the manner in which certain systems develop over time (Elaydi, 1996). Systems of such equations are widely used in both the social sciences and engineering, with some more specific examples being their application to control theory problems, econometric models, queuing problems, and behavior learning (Goldberg, 1958). This thesis will focus on synchronous data flow programs that are simulation models, including systems of difference equations.

Besides their ability to concisely represent a system of difference equations, data flow programming languages themselves are popular as general purpose programming tools. One of the more notable examples of data flow programming's popularity is G, a data flow programming language at the core of National Instrument's LabVIEW program (Bishop, 2007). Ptolemy II, produced by the Ptolemy Project at University of California, Berkeley, is another widely available program that supports data flow programming, as well as multiple other models of computation (Brooks, Lee, Liu, Neuendorffer, Zhao, and Zheng, 2008).

The act of modeling a system requires representing theoretical knowledge about the system under study in the structure of its model (Banks, 1998). This thesis proposes a method of utilizing the structural information encoded in a synchronous data flow simulation program to improve the efficiency of factor screening on that program.

Specifically, the developed method operates on the natural graph structures of data flow programs. Factor screening is accomplished through the application of an algorithm to rank

relative node importance on a data flow graph in an attempt to make conclusions about which nodes representing inputs are most important to a node representing an output. This algorithm, named the "Weighted Random Walks" method, makes heavy use of random walks on a data flow program's underlying graph with arcs weighted by studying sub-components of the model.

The utility of data flow programming, as well as their popularity, make this class of program an important topic for research. As with all simulation models, identifying the set of important input factors thereby reducing the number of considered input combinations in a data flow simulation is an important step in analysis. The ability to quickly estimate the importance of factors in data flow models has the potential to greatly speed up the analysis of this class of model, to the benefit of any discipline where such models are employed.

Chapter 2

Problem Statement

A common challenge of simulation modeling is that analysis of a given model can be very time consuming. Many frequently studied real world systems can be modeled to an arbitrary level of detail and the number of possible input combinations to these complex systems can be overwhelming from the standpoint of analysis (Banks, 1998). Formally, the problem of factor screening in simulation is the search through all of a model's potentially important input factors, k, for the most important subset of those factors, g, where $g << k$ (Montgomery, 1979, Bettonvil and Kleijnen, 1996).

The goal of this thesis is to develop an efficient method for identifying important input factors in large synchronous data flow simulation models. To be of practical usefulness, the developed method should have a run time smaller than that of alternative statistical factor screening experiments for models with many inputs.

The inherently graph based structure of data flow programs makes them well-suited to analysis based on their structure. The methods developed in this thesis explore using the graph structures of these data flow programs to suggest which input factors are more important than other input factors. The main hypotheses which will be examined are:

1. A graph-based measure of an input node's relative importance with respect to an output node on a synchronous data flow program's actor graph can be used to help identify which input factors have the greatest effect in the modeled system, and that

2. Information on input factor importance can be obtained more quickly through structure

based analysis compared to strictly statistical factor screening techniques of comparable accuracy.

These hypotheses will be explored through the main objectives of this thesis, which are to:

1. Modify as appropriate, and then apply tools from the domain of graph theory to the estimation of an input factor's importance to a given output in a simulation model implemented as a synchronous data flow simulation,

2. Assess the accuracy of the resulting methods, with comparisons to other factor screening techniques, and finally to

3. Assess the practical usefulness of the resulting methods with respect to run time and accuracy versus other factor screening techniques.

Reasonable inaccuracy, as proposed by the first hypothesis, is acceptable to the goals of this thesis since even less than perfect identification of important factors in a simulation are valuable. For example, many comprehensive statistical factor screening algorithms benefit from prior knowledge about the factors' expected importance (Kleijnen, 1998). The stated goal of the developed algorithms is to guide, and improve the efficiency of, further experimentation and optimization - not replace it. Successful completion this thesis' goal of efficient factor screening would be beneficial, as synchronous data flow programs are useful tools in a variety of fields.

Chapter 3

Literature Review

3.1 Review of Statistical Factor Screening Techniques

To help demonstrate the benefits of a model-structure based factor screening technique, statistical factor screening techniques are reviewed. The general mechanics of these factor screening techniques are also important, as they provide insight into how the structure, or substructures, of a simulation model may provide clues about important factors in a given model. Furthermore, an understanding of statistical factor screening reveals how prior knowledge about important factors, even if not exact or complete, can be useful for increasing the efficiency of the designed factor screening experiments reviewed in this section.

Factor screening has a long history in the literature of statistics and designed experiments. Recently, many historically standard factor screening techniques have been adapted to account for the unique characteristics of simulation. Although it is easy to control an arbitrarily large number of inputs in a simulation model, the same is not true when experimenting on a real system. Due to the difficulty of controlling many factors in real world systems, factor screening techniques developed outside the domain of simulation tend to focus on relatively few factors (Kleijnen, 1998). The high cost of experimentation on physical systems has also shaped the focus of factor screening to estimate as many effects from as few experimental runs as possible; a constraint which may not be as important in simulation experiments (Shen and Wan, 2005). Another important consideration when applying factor screening experiments to simulation

models is the special care that must be exercised when certain techniques are used, an example being the use of analysis of variance techniques to simulation output data if variance reduction techniques have been used (Montgomery, 1979).

Research into simulation factor screening has tended to focus on effects to a single response of a model, contrary to the tendency of an analyst to be interested in multiple outputs from the simulation (Cook, 1992). At least two ways of considering multiple outputs are proposed by Cook. The outputs from a model can each be considered independently, or they can be combined into a response function. Combining outputs has the advantage of a single response to analyze at the cost of less detailed information about individual outputs.

Typically, statistical factor screening experiments fall into categories including random balance sampling, full or fractional factorial experiments, and group screening. Some of these general factor screening techniques have been adapted into more specialized factor screening techniques, such as sequential bifurcation (Bettonvil and Kleijnen, 1996), controlled sequential factorial designs (Shen and Wan, 2005), and a hybrid controlled sequential bifurcation and controlled sequential factorial design (Shen and Hong, 2006).

Montgomery (1979) points out that full factorial designs, when appropriate, have the advantage of providing information about all input factor's main and interaction effects. Of full factorial designs for simulation, 2^k designs tend to be most efficient, although care must be taken to select appropriate factor levels. Although highly informative, due to the quick rate at which required simulation replications increase with an increase in considered factors, full factorial designs work poorly when a large number of factors must be considered. This makes these designs impractical for factor screening with thousands of potential input factors; a 2^k full factorial experiment with 1000 factors would require testing 1.07×10^{301} input combinations.

Factorial designs more practically applied to factor screening are 2^{k-p} partial factorial experiments. If an assumption is made that higher order interaction effects can be considered negligible, some of these effects can be aliased with each other or the main effects (Montgomery, 2005). Partial factorial designs result in experiments that require fewer input combinations than their full factorial relatives. For instance, a fully saturated, commonly referred to as resolution III, fractional factorial design can estimate main effects and requires as few as one

more run than the number of factors to investigate. Unfortunately, since resolution III designs require aliasing main effects with two way interaction effects, they may not be appropriate for factor screening when meaningful two-way (or higher order) interaction effects exist. Another popular resolution of two-level fractional factorial designs for factor screening are resolution IV designs, capable of estimating all main effects in a minimum number of runs equal to two times the number of factors. The benefit of a resolution IV design over a resolution III design is that in the resolution IV design, factor main effects are not aliased with two way interaction effects (although two way interaction effects are aliased with each other and higher order interaction effects). This characteristic makes resolution IV designs very popular factor screening experiments.

Random balance sampling, like two level factorial designs, requires varying input factors between two levels; high and low. The largest difference from factorial designs is, in random balance sampling, which inputs are set high and low in any given replication is determined randomly. The "balance" in the name of this method comes from the constraint that each factor must be set high in exactly half of the experimental replications. More formally, given k factors, initially set to their low value, and N total replicates, for each factor in k, randomly select a set of replications of size $N/2$ from N for which the current factor should be set to its high value (Mauro and Smith, 1984).

The main advantage of this design is the ability of an analyst to set the magnitude of N independently of the size of k. This is to say, regardless of how many factors one wishes to examine, the number of replicates may be set to any even number greater than zero (Mauro and Smith, 1984). Analysis in this way will lead to aliasing of effects, as also occurs in fractional factorial designs. The main disadvantage of random balance sampling compared to partial factorial designs is that in random balance sampling, aliasing occurs to a random degree (Mauro and Smith, 1984), while in fractional factorial designs the alias structure is specified (Montgomery, 2005). Random balance sampling is best suited for detecting a few large main effects with a small number of replications, which are qualities that make it well suited to use in factor screening.

In group screening methods, factors are assigned to groups and examined together. A group

of factors is considered a single "group factor" and the entire group's values are set high or low together (Kleijnen and Van Groenendaal, 1992). After grouping, experimentation can continue as it would without groups, by assigning group factors as an experiment would treat any individual factor. If a group factor is found not to be significant, it can be assumed that none of that group's members are important. On the other hand, if a group factor is found to be significant, at least one of that group's individual members is likely significant. All of a significant group's members could be marked for inclusion in further study, or additional factor screening could be carried out to determine which specific factors in a group are important.

Group based factor screening requires assumptions about the possible effects in a system. Of particular concern is the possibility of multiple main or interaction effects within a group to cancel the total effect to zero (Watson, 1961). Although potentially disastrous to a group screening method's ability to detect significance, this problem does not usually show up in practice, especially if an effort is made to code factor directions to correspond to a common direction of change in the output (Kleijnen and Van Groenendaal, 1992).

Sequential bifurcation is a specialized form of group screening that relies heavily on the aggregation of inputs. More specifically, sequential bifurcation is a sequential search technique that involves testing groups of factors simultaneously then splitting the initial group factors found to have a significant effect into smaller groups (Bettonvil and Kleijnen, 1996). When a group of factors is found to collectively have an effect, that group is split into two equally sized sub-groups which are tested in the same way. Search by splitting groups in two makes sequential bifurcation a type of binary search.

For sequential bifurcation,

$$maxn = 1 + g\left[\log_2\left(\frac{2K}{g}\right)\right] \tag{3.1}$$

Shows the relationship between the worst case number of simulation runs, $maxn$, given the total number of factors to search through k, and g, the number of all factors actually important (Bettonvil and Kleijnen, 1996). As expected, this relationship is very similar to the number of iterations required for any binary search algorithm. At its worst, sequential bifurcation may be less efficient compared to a fully saturated factorial design at searching a large number of

factors for the few important ones. Sequential bifurcation is more efficient at screening for a small number of important factors in a large number of total factors. Given 256 input factors of which 2 are actually important, sequential bifurcation will take less than or equal to 17 runs to find those two factors, while a completely saturated fractional factorial design would take 257 runs. This efficiency over a saturated fractional factorial design can be attributed in part to sequential bifurcation's specialization in factor screening. A factorial design aspires to determine detailed information about effects, sequential bifurcation does not.

An important disadvantage of sequential bifurcation is its ability to work on only one output at a time. Unless multiple outputs could be grouped into a single response function, as described above, sequential bifurcation may not provide any computational benefit to factor screening in models with many important outputs.

Prior knowledge about the importance of a simulation model's factors can greatly improve the efficiency of statistical group screening experiments. With group screening in general, if unimportant factors are grouped together, they can be eliminated together in efficiently large groups. One specific example is the sequential bifurcation method described by Bettonvil and Kleijnen (1996), which is most efficient if the input factors can be ordered by expected importance. Ordering the input factors by expected likelihood of importance increases the chance that unimportant factors will be grouped, and thereby eliminated, together during implementation of this algorithm. While better ordering increases the efficiency of sequential bifurcation, poor ordering will not decrease the procedure's effectiveness; it will only cause the number of required replications to rise closer to the maximum described by Equation (3.1).

One of the disadvantages of the cited statistical factor screening techniques is their reliance on model evaluation. These designed experiments all require manipulating a simulation model's input factors, then evaluating the model to obtain an exact output value. Although clearly valuable tools, it may prove too costly to perform a full factor screening experiment on a model with many input factors and a long run time. It would be beneficial if some input factors could be identified as unlikely to be important and discarded prior to the execution of any designed experiments.

3.2 Data Flow Programming

Data flow programming is the model of computation used by the simulation models that will be examined. What follows is a brief, high level overview of data flow programming with an emphasis on the specific subtype of data flow that will be analyzed; synchronous data flow programs. Special attention is given to analysis of the graph structure of such programs, in addition to concepts such as time and recurrence which are useful for the implementation of simulation models in data flow languages.

3.2.1 General Data Flow

In the data flow model of computation, modules react only to the presence of data at their inputs. This is different from other models of computation such as "imperative", where modules are executed sequentially, or "discrete event", where modules react to events that occur at a specific time (Chang, Ha, and Lee, 1997, Lee and Sangiovanni-Vincentelli, 1998).

A data flow program consists of a directed graph containing a set of processing nodes called "actors" connected by message passing arcs called "relations" (Dennis, 1980). Data flow program execution is controlled by the flow of value holding tokens between nodes along relations on this "data flow graph". In the context of data flow programming, a token should be thought of as a vehicle for the transportation of values around the data flow graph. In this context, the values carried by tokens are usually the results of intermediate calculations.

A node on a data flow graph may be categorized by its degree as one of three main types. In this work, a node with no input arcs is referred to as a source node, a node with no output arcs referred to as a sink node, and a node with at least one input arc and at least one output arc is called an intermediate node. A source node represents an actor that has no incoming arcs and must, therefore, operate without input from any of the other nodes on the actor graph. Source actors create tokens following some predefined rule. For example, the source actor pictured in Figure 3.1 may be designated to produce one token of value 3 at the beginning of the program execution. This token would then be passed to the successor of this source node on the actor graph.

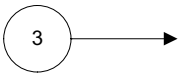

Figure 3.1: A source node producing tokens containing a value of 3.

Intermediate nodes receive tokens from either source nodes or other intermediate nodes and produce new tokens as a function of any received tokens. The addition actor shown in Figure 3.2, for example, will take a token from each input arcs *A* and *B*, and produce a new token with a value equal to the sum of the received tokens' values. The produced token will then be passed along the addition actor's output arc, *C*. The arcs in this example have been labeled only so they may be referred to in the narrative; arcs on a data flow graph are not typically labeled, nor do they perform any function besides describing paths that tokens may flow across. Although often representative of simple arithmetic, as is the case with the addition actor shown in Figure 3.2, intermediate nodes may perform operations of any complexity.

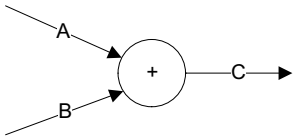

Figure 3.2: A simple addition actor.

Figure 3.3 illustrates a simple, but complete, data flow program consisting of two source nodes, two intermediate nodes, and one sink node. In this example, assume the sink node simply records the value of any tokens it receives for later review. Further assume the two source nodes in this example are set to produce and send one token on each of their output arcs. The produced tokens will contain a value equal to the label of their producing node. For this example, arcs *A* and *B* will each carry one token of value 3 from the source actor labeled "3", and arc *C* will carry one token of value 5. The two intermediate nodes "$+$" and "$\times$" will perform the operation described by their label, addition and multiplication respectively. During execution, arc *D* will carry one token of value 8, produced by applying the addition operator to tokens with values 3 from arc *B* plus 5 from arc *C*. Finally, the output of this program, carried by *E* to the output, is equivalent to $A \times D = 3 \times 8 = 24$. The output from the entire program can be described by the system of equations $A = 3, B = 3, C = 5, D = B + C, E = D \times A$,

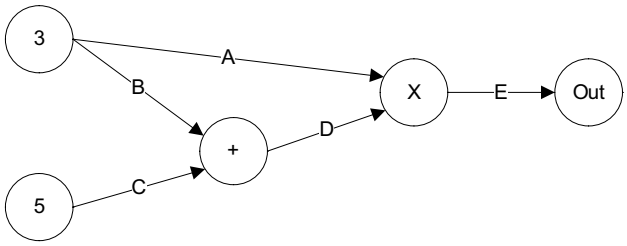

Figure 3.3: A simple data flow program.

and $Out = E$.

The order of node execution in Figure 3.3 is easy and straightforward to determine. After an actor's data dependencies are met, that actor performs its assigned computation and the result is propagated forward through the program. In data flow programs, a node's execution is completely controlled by the availability of data, as carried by tokens, at its inputs. Therefore such programs are referred to as "data-driven".

Any token received at a node is stored by that node in an internal queue representing which arc the token is received on. Once an intermediate node has received the required number of tokens on each of its required input arcs, that node is considered "enabled". Figure 3.4 (a) shows a partially enabled node that has received a token on one of its two input arcs. Figure 3.4 (b) shows a node in its fully enabled state. After becoming fully enabled, an intermediate node will autonomously "fire". When an actor fires, it removes some tokens from its input queues, performs some operation using the values of tokens it has received, and then produces one or more tokens that are sent along the data flow graph to the intermediate node's successors, as shown by 3.4 (c). Note that the token placed on the output arc is not the same as any of the tokens consumed to produce it, although its value may be related (in this example a relation by addition exists). The generalized data flow model does not constrain the number of tokens consumed or produced by an actor each time it fires to be one, or even constant; a generalized data flow actor may produce or consume a varying number of tokens.

The above described process of actor firing and token passing continues until some termination condition is met. During program execution an actor may receive and fire many tokens. A description of this model of computation has been formalized by Lee and Sangiovanni-Vincentelli (1998). In this formalization, a token is defined as an event containing a value and

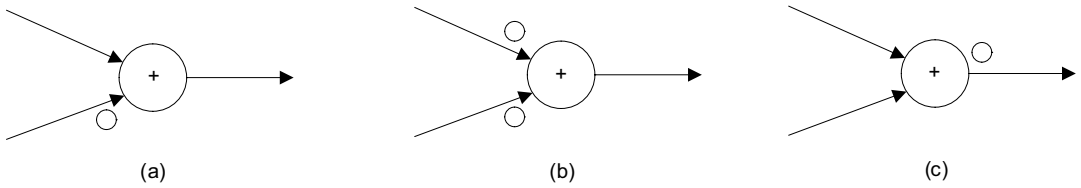

Figure 3.4: A partially enabled node (a), fully enabled node (b), and firing node (c). A node is considered fully enabled if there is at least one token at each input arcs it requires to fire.

tag pair, and a signal is defined as set of such events. Using this definition, the output of a data flow program can be described from part, or all, of one or many of the signals generated during the program execution.

Because data flow programs allow an actor to fire any time that input data is available, it is possible for more than one actor to be ready for firing at the same time. This offers the potential for a high level of concurrency, often cited as a main advantage of data flow programming (Dennis, 1980, Lee and Messerschmitt, 1987). Unfortunately, because the data flow model lacks the concept of control flow, program overhead must be spent to determine which actor to fire next when data flow programs are run on computers lacking the appropriate hardware architecture (Dennis, 1980).

Also important to the efficiency of a data flow program is the granularity of that program's nodes. Complex operations can be built from a large number of simple units, or one complex unit. The granularity of the model is considered course if the representative operations are implemented as single, highly complex, actors or fine granularity if the same operations are represented as a collection of many, individually less complex actors. In a data flow program, larger granularity result in less overhead (Greening, 1988, Lee and Messerschmitt, 1987) however, due to their more abstract representation, models with courser granularity also contain less structural information in their actor graphs.

Although not ideal from the standpoint of structural analysis, the reality that data flow nodes operate as independent processing units of any complexity greatly adds to the expressive power of data flow languages from the perspective of simulation modeling. Actors are not constrained to implement only simple arithmetic operators, or even single expressions of many arithmetic operators. One example of this power is demonstrated by Chang, Ha, and Lee (1997), where

the authors show how it is possible to mix dataflow models with discrete-event models. One example given by Chang et al. (1997) shows how a data flow actor may itself implement a discrete event simulation as its operation; instead of evaluating an expression of some parameters, a data flow actor could call a discrete event simulation and pass the result to other actors.

It should also be noted that actors may have internal state. An actor is allowed to remember the values of any tokens it received, or any states derived from these tokens, since the start of program execution. If the exact mechanics of a historically sensitive actor is known, it is possible to convert it to an actor without state through adding a self-loop to communicate prior state to itself between program iterations (Buck, 1993).

3.2.2 Petri Nets

Petri nets are a type of data flow program commonly used for a variety of applications, especially the modeling of concurrent systems (Tadao, 1988). One specific example is the application of Petri nets to manufacturing systems (Proth and Xie, 1996).

A Petri net is a directed bipartite graph containing nodes called places connected to nodes called transitions (Proth and Xie, 1996). Transition nodes in a Petri net behave like actors, as described above. Place nodes in a Petri net behave like relations as described above or, more specifically, as a queue of tokens at an actor's input.

Besides the differences mentioned above, Petri nets behave exactly as a simple form of data flow. Transitions are considered enabled when their dependencies are met (Proth and Xie, 1996). They then fire or remove some tokens from their input places and put some tokens at their output places. In basic Petri nets, these tokens are unlabeled. An important extension to the Petri net model are colored Petri nets. Colored Petri nets use tokens having different colors, or labels. They generally allow a more compact representation of a system, but are disadvantageous in that many of the more useful analytical properties of elementary Petri nets are not easily applied to colored Petri nets.

It is in the study of Petri Nets that many techniques for the study of synchronous data flow program structures originate. A review of such techniques, including examples which investigate the reachability, boundedness, and liveness of a system, as well as a complete introductory

tutorial of Petri nets' usage in modeling, can be found in Tadao (1988).

3.2.3 Synchronous Data Flow

Synchronous data flow (SDF) programming is a special case of data flow programming where, at least in its pure state, the number of tokens consumed and produced by an actor each time it fires is independent of the token values (Lee and Messerschmitt, 1987). This allows a static order of execution for actors in a SDF program to be computed during the program's initialization phase, greatly reducing some of the overhead associated with data flow program execution (Lee and Messerschmitt, 1987). Furthermore, this assumption makes synchronous data flow a subclass of Petri Nets (Buck, 1993).

In reality, the synchronous data flow language described by Lee and Messerschmitt (1987), and which will be used to implement simulation models, differs from the above stated ideal. The most notable difference is the inclusion of dynamic actors, two popular ones being the switch and select actors shown in Figure 3.5, capable of producing or consuming a varying number of tokens on each of their arcs (Buck, 1993). This extension of SDF can be made with little impact to the scheduling and efficiency of the resulting programs, although some scheduling decisions will need to be made at run time (Lee and Messerschmitt, 1987).

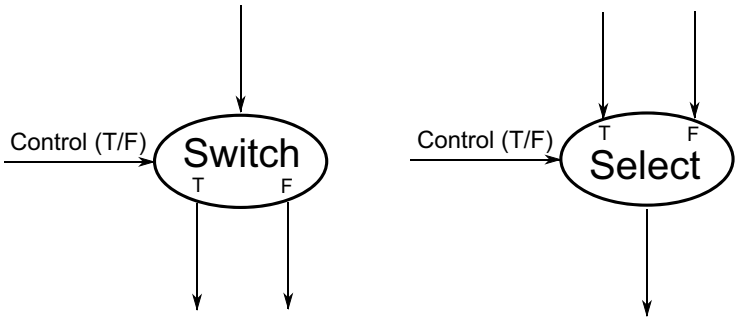

Figure 3.5: Two example dynamic data flow actors.

The addition of dynamic actors to extend synchronous data flow greatly improves the power of such models. While basic Petri net models are not Turing-equivalent, dynamic data flow programs usually are (Buck, 1993).

An important property of the synchronous data flow model of computation is that it contains a representation of time defined by a complete firing cycle (Chang et al., 1997). Time in an

SDF model progresses globally to the model in discrete quantities referred to as "ticks" or "iterations", unlike discrete event and continuous time programs where time is modeled by the application of a time-stamp to events.

Time is an important concept if a SDF program is to be used for certain simulation purposes. One consequence of time dependent models is that an input factor may have an effect on a model's output instantly and to a constant extent over time, or that input's effect may vary over time- such an effect may be meaningless in the first iterations but grow to be a dominate effect after many. Such accumulations of effect may happen when state data is stored in an SDF model between iterations, as happens when such a model contains a directed cycle.

Figure 3.6 shows two actor graphs with a directed cycle between nodes *A* and *B*. The model shown in Figure 3.6 (a) is impossible to start. Node *A* depends on both the value of *X* and the value of *B* to fire. Node *B* depends on node *Y* and node *A*. Since node *A* depends on node *B* and node *B* depends on node *A*, a cyclic dependency exists which effectively prevents the model from starting up. One way this cyclic dependency can be handled in SDF is to introduce a delay on one of the arcs in the cycle (Lee and Messerschmitt, 1987). A delay holds any token it encounters for some fixed number of iterations before releasing it. Figure 3.6 (b) shows the same model as (a) but with a single iteration delay on the directed arc from *A* to *B*. Since, by the definition of synchronous data flow, it must pass a token in every iteration, a delay must hold an initial token at the beginning of execution for passage in the first iteration. Curly brackets are used here to show the initial token values contained by the delay, in this example one token of value 0. In the first iteration of the SDF model from 3.6 (b), node *B* will take data from *Y* and the *delay*, initially 0, to produce its output. In the N^{th} iteration, node *B* will take data from *Y* and the delayed output of *A*, or the value sent by *A* in iteration $(N-1)$ to perform its operation. In a SDF model, it is required that all cycles be broken by a delay; failure to do so is a structural fault that results in an unusable model (Lee and Messerschmitt, 1987). Depending on the implementation language, delays may exist as attributes of the arc as shown in this example or as specialized actor nodes on the network.

Cycles and delays in SDF can result in varying importance of an input over time, such as is common in system dynamics models. A self loop can also be used by an actor to remember

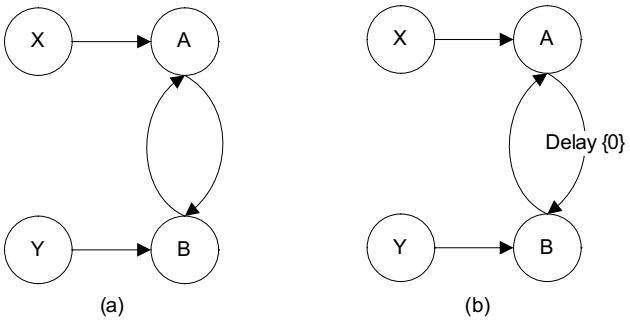

Figure 3.6: An example SDF model with a directed cycle.

a state between iterations (although this can also be done internally by the actor). In the new example of Figure 3.7, for example, assume the initial output of node f is $f(X)$ and node g's output is $g(Y, 0)$. Note that node g has two input arcs, one of which is a self loop. Assuming X and Y remain constant, the values output in the second iteration for f and g are $f(X)$ and $g(Y, g(Y))$, respectively. If Y were not constant, call Y_0 and Y_1 the initial and second iteration values of Y respectively, the value of g at the second iteration could be more thoroughly described as $g(Y_2, g(Y_1))$. In general, the cycle around g in this example will cause g's output on any given iteration to be partially dependent upon its output from the previous iteration, which is itself partially dependent on node g's output from the iteration before that. This dependency continues recursively back to the first iteration, with the result being that node g's output is dependent upon all historical values of Y. This recursive dependency is one definition of a difference equation; $x(n+1) = f(x(n))$ (Elaydi, 1996).

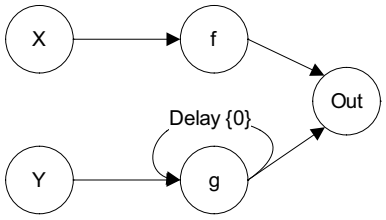

Figure 3.7: A SDF model with one self loop.

In this way, SDF programs may be used to simulate systems of difference equations; the behavior of which can lead to highly dynamic behavior. There exist methods for finding exact solutions to certain difference equations (Goldberg, 1958). Unfortunately, the application of

these solution methods to SDF programs would be extremely difficult in practice, due in part to the varying levels of granularity and transparency in SDF models. Although an actor may contain a process to implement some function f to map a set of inputs X to a set of responses Y, no guarantee is made that f itself is retrievable from the program. Any or all actors in a data flow program may be implemented as a "black-box".

As with Petri Nets, since their structures are strongly graph based, synchronous data flow programs lend themselves well to graph theoretic based analysis. One simple example of this is presented by Lee and Messerschmitt (1987) to verify the consistency of token production and consumption.

Feng (2008) develops a model transformation approach to aid in the design of large structurally configurable models. This is accomplished through the application of graph transformation algorithms. Principally concerned with the transformation of a model to speed up construction and configuration, Feng's work is an example of how the graph structure of a data flow program can be examined and manipulated for productive purposes.

Of specific interest to analysis of a SDF simulation model is the ability to determine which source actors, or inputs, have a causal impact on other actors in the model. A further question is, given a set of source actors that have an effect on a given node, which of those effects are of most importance.

The first question, which inputs have an effect on which outputs, can be addressed using "causality interfaces" (Zhou and Lee, 2008). A causality interface, as defined by Zhou and Lee (2008) states the dependency that an actor's output signal has on input signals. It is also shown that if the causality interfaces are known for all actors in an actor graph, it is possible to determine the causality interface for the entire network, or composition of actors. Causality interfaces are described by Zhou and Lee (2008) primarily as a means for deadlock analysis in data flow, or as a tool for discovering structural faults in a model.

Just having a causal effect does not necessarily indicate that effect has a meaningful magnitude. Given a set of inputs that are known to contribute to an output's value, the individual effect of some inputs may be overwhelmed by the effect of others. The solution to this problem is the defined goal of factor screening. The problem of determining important effects becomes

even more difficult with the existence of cycles, as was shown by example in 3.7. To date, there does not appear to be any literature on ranking or identifying important effects using an actor graph's structure alone.

3.3 Review of Literature on Graph Structure Analysis

Considerable work has been done in developing methods to determine useful information from structured data. Where graph based information exists there can often be found research into how best to analyze that information. Some specifically interesting questions to the scope of this thesis are:

- What makes a path of flow through the graph important?
- What makes an individual node important, both globally and relative to another node?

This section contains a review of literature relevant to answering these questions, framed by application to analysis of a data flow program's actor graph. Background information, particularly on Markov chains, will be included as required.

3.3.1 Linear Signal Flow Graphs

Linear signal flow graphs are a well established tool for studying and reducing complex feedback systems. They consist of a graph with nodes representing linear functions and arcs representing the dependency of data between nodes. In this way, linear signal flow graphs are closely related to the data flow programs described in Section 3.2.

The behavior of a linear signal flow graph can be described concisely by two rules. The first rule, shown by Figure 3.8 and described by $X_i = \sum_{j=1}^{n} A_{ij} X_j$, shows how values for individual nodes are computed (DeStefano, 1990).

The second rule, shown by Figure 3.9 and $X_i = A_{ik} X_k$, constraints values transmitted from a common source to be related by the value of the node which is their source (DeStefano, 1990).

Linear signal flow graphs can provide a convenient graphical representation of linear systems. They can also help with analysis, and can be structurally reduced. One example of such

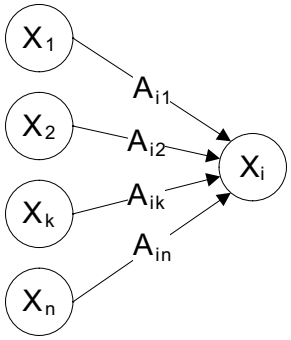

Figure 3.8: Addition rule for signal flow graphs.

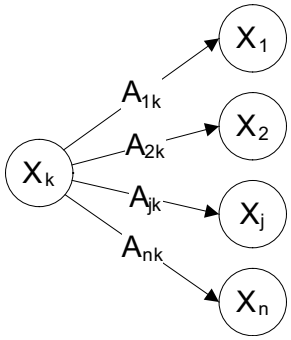

Figure 3.9: Transmission rule for signal flow graphs.

a structural reduction is given by Figure 3.10 and its reduction, Figure 3.11.

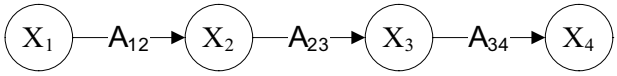

Figure 3.10: Example linear signal flow graph.

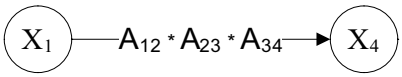

Figure 3.11: Reduction of example linear signal flow graph.

This example shows how entire variables, and the nodes representing them, can be removed by applying properties of multiplication. More complex examples of reduction, including reduction techniques that can simplify feedback loops, are given in DeStefano (1990).

Linear signal flow graphs can be implemented as special types of data flow programs, however the restriction that they contain only linear equations makes them, and methods for their reduction, inadequate for analysis of all but the most specific synchronous data flow graphs.

3.3.2 Path Importance

Paths on an actor graph represent channels over which information can flow. In the context of a data flow actor graph, an important path may be considered one that is likely to have great influence on an output of interest. This is to say, if information enters the head of an important path, that information could be expected to prove very significant to the information resulting at the tail of that path.

A collection of paths connecting two nodes is a subgraph. In general, a subgraph is useful to describe the relationship between two nodes when relationships between those nodes are multifaceted; often times representing a relationship using a single path is ineffective (Faloutsos, Mccurley, and Tompkins, 2004). For an example in a social context, modified from an example given by Faloutsos et al. (2004), imagine that your brother is married to your favorite author. The shortest path between you and your sister in law on a social network would be that she is your favorite author. It is, however, likely that your most important relation to her is through her marriage to your brother. Multiple relationships like this exist in data flow graphs. The most important path between actors may not be the shortest, and many less important paths may contribute to one most important effect. It is important to consider all paths between an input and output of interest.

The concept of important paths are used heavily by the work of Mojtahedzadeh, Anderson, and Richardson (2004), which discusses a method of analyzing a system dynamics model's structure called the "Pathway Participation Method" (PPM). The PPM starts with a single variable (output) of interest, then works backward to determine the most important structure in the model. This is accomplished by studying individual components to identify which path into that component most influences the output. The PPM then recursively examines the component upstream along the just identified path until the method converges on the most important structure of the model.

Faloutsos et al. (2004) propose a method for discovery of important subgraphs connecting two nodes. Their procedure treats the search for a subgraph as a maximum flow problem. The result is a collection of the important components comprising the potentially complex relation between two nodes on a graph.

3.3.3 Node Importance

The definition of node importance, as with the importance of a path, is ambiguous and domain dependent. In literature, the interpretation of a node's importance usually depends on the means used to compute it.

It is easy to see how a measure for a node's importance is useful in a variety of graph structures. Considering the World Wide Web, one may be interested in the most important website on the Internet. In social networks, the most important person in an organization may be of interest. Analysis of a transportation system may be interested in the most important bridge or intersection. In the actor graph of a data flow program, it may be useful to determine which actor is most important, the answer to which is synonymous with the most important calculation. Although the meaning of an important node is interpreted differently in these four examples (the Internet, social networks, transportation networks, and actor graphs), in each case a node is considered important because it is somehow related to other nodes in a meaningful way.

The importance of a node also depends on perspective. In a social network of a large organization, the most important person on the whole graph is arguably the company's leader. Less globally, the most important person from the perspective of an individual in a company is likely that individual's immediate boss. A node's absolute importance is defined as the importance of that node considering all other nodes in the graph. The relative importance of a node can only be found specific to a subset of nodes.

Much work has been done on developing methods for discovering important nodes. Of these methods there appear to be at least two main classes:

1. Methods that rank nodes based on graph theoretic notions (distance, node degree, etc.).

2. Methods that rank nodes by considering the graph to be a stochastic process.

3.3.3.1 Methods of Graph Theoretic Notions

Many methods from the first class rely on measures of a node's centrality in a graph. Possibly the simplest measure of a node's importance is the degree of that node. In the case of a

transportation network, the highest degree node would represent the intersection with the most roads entering it. In a data flow actor graph, the highest degree node will represent the function with the most input factors.

Global closeness is a method developed by Freeman (1979) to rank the centrality of a node on a graph. Freeman defines a given node's closeness centrality to be the sum of the distance from that node to all other nodes on the graph. While closeness centrality is a useful measure when distances on a graph hold important meaning, its power is limited when distances themselves are not necessarily meaningful (Borgatti, 2005). Freeman (1979) also proposed another measure of centrality called "betweenness". Call g_{ij} the total number of geodesic paths from node *i* to *j* and $g_{ij}(k)$ the number of those geodesic (shortest) paths passing through node *k*. The computation for betweenness centrality is then given by Freeman (1979) to be

$$\sum_i \sum_j \frac{g_{ij}(k)}{g_{ij}}, \forall i \neq j \neq k. \tag{3.2}$$

Essentially, betweenness centrality is a measure of how many geodesic paths pass through node *k* (Borgatti, 2005). As with global closeness, betweenness centrality is most relevant when distance has meaning in a graph. Even so, measures of centrality in an actor graph may be helpful in determining which nodes have the most widespread effect. If an actor on a data flow actor graph has a large influence on information passing through it, the larger that influential node's centrality measure, the more other nodes its influence is likely to effect.

3.3.3.2 Stochastic Process Methods

A second main class of methods for importance ranking is a class of methods which view a given graph as a stochastic process. More specifically, assume a graph to be a representation of a first order Markov chain. A Markov chain is a discrete-time probabilistic model that is useful for studying a system of states and transitions between those states. The study of a Markov chain is facilitated by an application of the Markov assumption that the probability of transitioning to any given state depends only on the current state of the system (Beichelt, 2006). Under this assumption, a Markov chain can be completely described by a matrix of one-step transition probabilities $\mathbf{P}$ and a corresponding initial probability vector $\mathbf{p}^0$. If the Markov chain

is in state *i* at time *n*, it will transition to state *j* at time $n + 1$ with a probability of p_{ij}. The system does not necessarily have to transition to a different state; it will remain in its current state with probability p_{ii}. Being composed of transition probabilities, certain key properties must hold for the elements of **P**:

$$0 \leq p_{ij} \leq 1, \forall i, \forall j \text{ and } \sum_{j} p_{ij} = 1, \forall i. \tag{3.3}$$

For representation as a Markov chain, the individual nodes of a graph should be considered different states and arcs indicate transition probabilities between those states that may be non-zero. Under this assumption, imagine a single token on the graph. If the token is at node *i* at time *n*, **P** for this system is made up of some fixed probabilities p_{ij} that the token will transition to node *j*. The token will only be allowed to transition between nodes if they are connected in the graph; $p_{ij} = 0$ if node *i* is not connected to node *j*, $1 \geq p_{ij} \geq 0$ if *i* is connected to *j*. If this single token is allowed to randomly transverse the graph (take a random walk), the long-term fraction of time that this token is expected to be at any given node can then be thought of as an estimate for that node's importance (White and Smyth, 2003).

Further background in discrete-time Markov chains is required to fully understand the mechanics behind computing the long-term fraction of time a token is expected to reside at a given state. Take for example the Markov chain described by Figure 3.12 which was built as described above from the graph of Figure 3.13, assuming an equal probability of transition along any outgoing arcs from a node. Because *Node B* has two out arcs, each arc has a 0.5 probability of transition. If *Node B* had three out arcs, each arc would be associated with a 0.33 probability. The first column and row of **P** in Equation 3.4 refers to *State A*, the second column and row to *State B*, and the third to *State C*. The 1 in the first row and third column of **P** in Equation 3.4 refers to the 100% probability of transition from *State A* to *State C*. In this way **P** is closely related to the incident matrix of the graph from Figure 3.13.

If **P** denotes the single-step probabilities in a Markov chain, $\mathbf{P}^2$ gives the two-step transition probabilities, and $\mathbf{P}^m$ is a matrix of the *m-step* transition probabilities (Beichelt, 2006). Examples of the two-step, three-step, and hundred-step transition probabilities for the model of Figures 3.12 and 3.13 are given in Figure 3.14.

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{pmatrix}; \mathbf{p}^0 = (1, 0, 0)$$

Figure 3.12: Example Markov chain expressed as one-step transition probability matrix and initial probability vector.

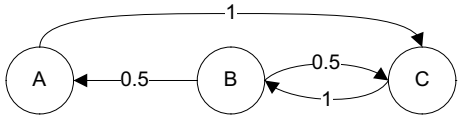

Figure 3.13: Example Markov chain expressed as a graph.

What the individual matrices in Figure 3.14 describe are the probability of transition from state *i* to state *j* in the number of steps indicated by the degree of **P**. The first row of $\mathbf{P}^3$, for example, shows that if starting in *State 1*, given three steps there is a 0.5 probability the system will end up back in *State 1*, and a 0.5 probability the system will end up in *State 3*.

The matrix $\mathbf{P}^{100}$ seems to indicate that, regardless of the starting state, after 100 steps there is a 0.2 probability of being in *State 1*, a 0.4 probability of being in *State 2*, and a 0.4 probability of being in *State 3*. In reality, the apparent indifference to starting state observed in $\mathbf{P}^{100}$ of Figure 3.14 is due to rounding. If enough decimal places were displayed, it would be shown that, given a start from *State 2*, there is actually a slightly less than 0.4 probability of ending back in *State 2* after 100 steps. Even so, $\mathbf{P}^{100}$ does suggest that the elements of the rows from these m-step transition probability matrices are converging to some values as m approaches infinity, specifically $\pi = (0.2, 0.4, 0.4)$. If a probability distribution vector that describes the long-term behavior regardless of the starting distribution vector $\mathbf{p}^0$ for the system under study exists, it is called the stationary state probability vector and is referred to as the vector π (Beichelt, 2006). More formally, to be considered a system's stationary state probability vector, the elements of π must satisfy a system of linear equations described by

$$\pi_j = \sum_i \pi_i p_{ij}; \forall j. \tag{3.4}$$

From this, assuming some π exists for a model its exact values can be computed using Equation

$$\mathbf{P}^2 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0.5 \end{pmatrix}$$

$$\mathbf{P}^3 = \begin{pmatrix} 0.5 & 0 & 0.5 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{pmatrix}$$

$$\mathbf{P}^{100} = \begin{pmatrix} 0.2 & 0.4 & 0.4 \\ 0.2 & 0.4 & 0.4 \\ 0.2 & 0.4 & 0.4 \end{pmatrix}$$

Figure 3.14: Various m-step transition probability matrices.

3.4 and $\mathbf{P}^0$.

From this definition, the interpretation of π is synonymous with the long-term fraction of time that a randomly-walking token is expected to reside at any given node, one of the importance metrics presented by White and Smyth (2003).

Variations on random walks for importance evaluation have been successfully applied to a variety of applications. One successful example is the PageRank algorithm of Page, Brin, Rajeev, and Terry (1999) used by the Google Internet search engine. The PageRank algorithm has a strong foundation in random walks on graphs. A simplistic explanation of PageRank is that a website is ranked highly if an individual randomly following links on a graph of the Web is likely to spend a large percentage of his time at that given website.

Similar to the long-run proportion of time a random walker is expected to be at any given node, measures of the token's expected first-passage time and commute time between nodes are often cited measure of relative importance between nodes. The average first-passage time of a randomly walking token between two nodes is the expected number of steps that a randomly-walking token, leaving the first node, will take to reach the second (Fouss, Pirotte, Renders, and Saerens, 2007). Expected commute time for two nodes is the expected number of steps required for a round trip from the first node to the second node and back to the first. These metrics can all be computed for a Markov chain, in a similar fashion to how the long-term residence probabilities were obtained for the above example. Fouss et al. (2007) showed these reviewed

methods based on considering a graph to be a Markov chain work well when compared to other standard scoring algorithms for a case study of a movie database; although they point out these techniques are not likely to scale efficiently to very large systems. Even so, these methods that view a graph as a stochastic process have the potential to be highly useful to analysis of a data flow graph, as they rely on the connection structure of a graph - not distances between nodes.

3.4 Discussion

Two hypotheses were proposed in the problem definition of Chapter 2. The first stated hypothesis, about the existence of information in a data flow graph, has roots in prior work as detailed in the review section on data flow programming in Section 3.2 and the review of literature on graph structure analysis in Section 3.3. Three specific examples on the analysis of a data flow program from that program's structure were noted; one related to the use of model transformation to the construction of structurally configurable models, another related to consistency of token production and consumption rates, and the third to deadlock, or liveliness, analysis.

More applicable to the goals of this thesis is the work of Mojtahedzadeh et al. (2004). In this work, an analysis of a graph-based model is performed, called Pathway Participation Method (PPM), to identify the most important paths through a simulation with respect to an output. The PPM method, however, only considers the most important effect on individual components; it does not account for instances where many paths in a model combine to result in large effects. Furthermore, the PPM method focuses on identifying important model structures to aid in model understanding, with less emphasis placed on the search for important input factors.

Taylor, Ford, and Ford (2007) proposes a method to utilizing the results from statistical factor screening for the identification of important model structures. After factors of high importance have been identified, the model can be examined to see which model sub-structures the most influential parameters are connected to; essentially the opposite of this thesis's goal of using important structures to identify important input factors. While of benefit for model validation, since the method proposed by Taylor et al. (2007) requires computation of correlation coefficients through a statistical screening experiment, it is unlikely to yield a reduction in

factor screen time.

Many more examples exist, especially for the structural analysis of Petri nets, a type of model very similar to synchronous data flow. Many of these examples, such as reachability, boundness, and liveness, deal almost exclusively with the behavior and study of the program semantics, not necessarily the output of these programs. While potentially useful, the overhead involved with the most promising of these specialized tools is too large to be of practical value to a simulation analyst.

Similarly, the reachability between nodes of a graph based simulation model has been cited as an analysis technique (Oliva, 2004). Although simple reachability is fine in most cases, the causality interfaces of Zhou and Lee (2008) greatly extends the ability to determine which nodes may effect other nodes in models where causality information is known for the individual nodes.

In this work, a focus will be placed on the analysis of data flow actor graphs through representation as Markov chains, as demonstrated in Section 3.3. Similar work was performed on data flow graphs by Greening (1988). In his work, Greening (1988) expressed a probabilistic data flow program by converting it into a Markov chain for analysis. Greening's work focused on probabilities relating to the flow of tokens between actors in a non-synchronous data flow program and the application of Markov analysis to partition, or schedule, the actors of that program's actors into threads for efficient execution.

In addition to algorithms for the study of specific graph based models, a number of generalized examples of algorithms to determine node importance on graphs exist and a selection of these algorithms was reviewed in Section 3.2. Although the analysis of data flow graphs has been previously explored, none of the reviewed literature attempted to utilize these graphs to identify the importance of an input node with respect to an output node in data flow simulation models. Likewise, although evaluating the importance of nodes on a graph has been successfully documented in a number of application areas, notable social networks, transportation, and the Internet, no examples were found of these algorithm's application to data flow graphs or, more specifically, system dynamic models.

The second hypothesis proposed in the problem statement, predicting time savings com-

pared to statistical factor screening techniques, is based off the many simulation runs required by statistical screening methods; the exact nature of which are reviewed in Section 3.1. Compared to the relative ease of structural analysis referenced in the data flow review, statistical experiments are expected to take significantly longer.

One consequence of the first hypothesis, and related to the first goal, is that the resulting methods from this investigation will likely be heuristic in nature. The first hypothesis is a conjecture that information in the actor graph structure of a SDF program can provide help in identifying important input factors, not that enough data exists in this structure to support an exact conclusion. This is supported by the review of data flow programming in Section 3.2, where it is noted that the graph structure of a data flow program is only part of that program's description. For instance, no claim is made that the operations performed by the individual processing elements of a data flow program are visible; many such nodes may be opaque with respect to their internal processing.

Chapter 4

Methodology

As stated in Chapter 2, the first objective of this thesis is to modify as appropriate, and then apply tools from the domain of graph theory to the estimation of an input factor's importance to a given output in a synchronous data flow (SDF) simulation model. The main hypothesis of this thesis, that a graph-based measure of relative importance between nodes on the actor graph of a SDF simulation model can be examined to help identify which input factors have the largest effects in the target model, would allow realization of this goal.

This chapter contains the methodology and development of an algorithm, based on random walks through a data flow program's actor graph, that effectively meets this goal, organized as follows. Section 4.1 discusses the scope of this work, including the scope of the information the developed factor screen methods will return and information relating to the simulation modeling environment that will be used. This section on scope also includes the definition of what is meant by "importance". The use of shortest paths through a data flow graph for use in identifying potentially important inputs is discussed in Section 4.2. Section 4.3 contains a discussion on the motivation for the use of random walks for the identification of important inputs, including the featured Weighted Random Walks method. Finally, Section 4.4 contains the methodology and a description of the implementation used to run the Weighted Random Walks method.

4.1 Scope of Work

The processes developed by this thesis will result in a list of a synchronous data flow simulation's input factors ordered by their estimated importance with respect to one of the target system's performance measures after a given, predetermined, quantity of simulated time. This list will be based on structural analysis of the model's underlying actor graph.

Methods developed by this work operate as heuristic factor screening techniques. They apply graph theoretic algorithms resulting in a measure of one node's relative importance with respect to another node. This measure is an estimate of relative importance which differs from the actual magnitude of effects among actors in a data flow program. Ideally, these two importance quantities would be highly correlated. In practice, a method will be considered good if the order of magnitudes agree between these two measures; if one node has a larger actual effect on an output than a second node, the estimated first node's importance should also be higher than the estimated importance of the second node.

The final output of the methods developed are represented as a list of input nodes sorted by their estimated importance to an output node. The order of this list will imply the likeliness of being important with respect to the output under analysis. The claim made by such a list is that the input listed first is more likely to be important to the target output than the input listed second, and so on. In general, the input listed in position n is more likely to be important to the target output than the input listed in position $n+1$. Outputs will be examined one node at a time, instead of being combined into a single response function as described in the review of statistical factor screening. Due to the nature of data flow programs, multiple outputs could be combined at a single data flow actor if an analyst wishes to examine them together.

Quantifying the relationship between estimated importance and actual magnitude of effect is outside the scope of this work. This is why output from the developed methods should be considered an ordered list, instead of a collection of input-importance pairings; although the estimated importance is meaningful, its exact meaning is unknown outside its relative position to the estimated importance of other inputs.

4.1.1 Simulation Models

Experimental simulation models are required for testing and evaluating the methods developed by this work. A variety of different models are required to adequately characterize the developed methods. These experimental models must be implemented as synchronous data flow programs, with inputs modeled as user-controllable parameters. Additionally, all values passed by the experimental models must be typed such that they can be used in a 2-level statistical experiment; specifically, as either decimal, integer, or boolean. Details of the models built for testing purposes are presented in Section 5.3.

4.1.2 Data Flow Program Implementation Language

Ptolemy II, produced by the Ptolemy Project at University of California, Berkeley, is used as the environment for implementation of the simulation models under study. Ptolemy II supports a variety of computation modes, including a robust synchronous data flow domain. Members of the Ptolemy team are highly active in the field of concurrent modeling. Ptolemy II is a full featured software package constantly updated to include the most recent advancements in its field.

Ptolemy II was specifically designed to be a "laboratory for experimenting with design techniques" (Brooks et al., 2008). The open architecture and source of Ptolemy II, along with its detailed documentation, make it a good platform on which to build methods.

An example Ptolemy II model is shown in Figure 4.1. This simple example contains four atomic actors, two *parameters* and one director. This model evaluates, then plots, the function $9 \times (9 + 1) + 100 \times 3$. Note how the two *parameters*, *A* and *B*, are not directly connected to the blocks which reference them. This important connection information will be incorporated during the graph elicitation process described in Section 4.1.4.

4.1.3 Model Inputs and Outputs

In general, the inputs to a data flow program are the values produced by source actors. Outputs can be values recorded by sink actors or values generated by any intermediate actors in the

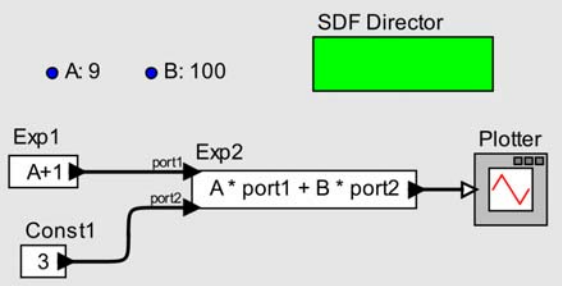

Figure 4.1: Example Ptolemy II model.

graph. In practice, at least two types of inputs exist:

1. Inputs that determine the value of source token production. Source actors themselves will not be called inputs, but any parameters controlling the value of the tokens produced by a source actor will be.

2. Inputs that are attributes of an intermediate node and help determine the behavior of such nodes. In data flow simulation models, such inputs often take the form of coefficients used in regression functions.

In the scope of this work, input factors are defined as parameters that are user-controlled attributes of the model as well as source node parameters on that program's elicited data flow graph. In Ptolemy II, this includes any model components belonging to the class *parameter*. This definition of an input differs from simply considering source actors, and is required to represent the two types of inputs listed above. A consequence of this definition is that only source nodes that draw their values from model parameters are considered inputs. Any nodes that read a value from an attribute that is only in scope for that specific node are considered constants, and are not considered an input to the model. An assumption is made that such source nodes are modeled in this explicitly inflexible manner specifically to exclude them from consideration as inputs.

An output is defined in this work as the value of an actor's output stream on a given iteration. According to this definition, factor screening will be carried out for a given, predetermined span of time. Further according to this definition, any actor that produces values may be considered an output. It is not enough to only consider sink actor nodes, as a simulation analyst may be

interested in intermediate outputs. By this definition, every non-source actor node in the model may be considered an output. In practice, there is usually a very small subset of total outputs that are meaningful and worth examining. It is these meaningful outputs that are the target of interest to this thesis. Which individual outputs are meaningful is a question that is dependent on the domain of the simulation under evaluation as determined by a domain expert.

4.1.4 Elicitation of Graph Structure

As discussed extensively in the review of data flow programming in Section 3.2, synchronous data flow programs are naturally represented as directed graphs. Even so, some framework is needed to interact with these structures. The framework selected for this task is the Java Universal Network / Graph Framework (JUNG), a software library written in Java that provides an architecture for the creation, analysis, and visualization of graphs (Madahain, Fisher, Smyth, White, and Boey, 2005).

In general, an elicited graph is constructed by querying and copying the actor graph's structure, and is identical to the actor graph of the data flow program it is based on. Nodes on the elicited graph represent actors, and edges represent relations.

One difference between the data flow graph and the elicited graph structure is required in certain environments where inputs are expressed as *parameters*, such as in Ptolemy II. While not technically actors themselves, these *parameters* act as actors in the way they interact with the program. Because of this, parameters are modeled as nodes on the elicited graph and connected by directed arcs to any other nodes referencing them. Many such parameter representative nodes are source nodes on the elicited graph, and thereby program inputs, as defined by Section 4.1.3.

A further elicitation step is performed for any actors with multiple outputs. For reasons described later, in Section 4.3.2.2, these actors will need to be expressed as multiple nodes. Specifically one node per actor output is required in the elicited graph. After this elicitation step, nodes on the elicited graph represent individual actor outputs in the data flow program.

Graph structure resulting from this elicitation process applied to the example Ptolemy II model of Figure 4.1 is shown by Figure 4.2.

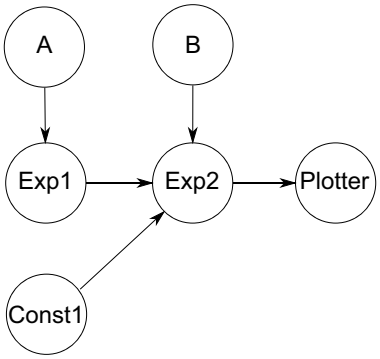

Figure 4.2: Graph structure elicited from example Ptolemy II model.

4.1.5 Defining Importance

In the remainder of this document, a node's importance is defined to be the magnitude of that node's main effect on an output. This magnitude of main effect will be referred to as *actual importance*. The employed factor screen methods will return an estimate for the relative magnitudes of those actual importance. This estimate will be referred to as *estimated importance*. The provided definition of importance as main effect does not intentionally ignore interaction effects, but does rely on the assumption that if a node participates as part of an important interaction effect, that node will also have an important main effect. This is a commonly used assumption in factor screening (Montgomery, 2005).

Two level experiments are used to compute main effects in the experimental models, as they are convenient methods for inspecting the structure of effects between inputs and an output over a range of values. For specific information about the applications of two level experiments to simulation analysis, see Section 3.1.

4.2 Shortest Path for Importance Estimation

It was noted in the review of literature on graph structure analysis that algorithms for estimation of importance in graphs can be categorized into a number of classes. Included in these classes was a class of algorithms designed to operate on graph theoretic notions of distance. Such algorithms propose that nodes that are close to each other are also likely important to each other.

One of the simplest such methods would be to consider the shortest paths between nodes. The shortest path distance is defined as "the minimum total path length between two given nodes" (Dijkstra, 1959). There exist a variety of algorithms to find shortest path distances. The exact mechanics of methods for finding shortest path distances are outside the scope of this work and unnecessary for understanding the examples presented in this chapter.

One possible application of shortest path lengths to the identification of important inputs in a data flow program would be to consider one input node as more likely than a second input node to have a meaningful effect on a given output node, if the first node's shortest path length to that output is less than the shortest path length to that output from the second node. If $\delta(x, y)$ represents the shortest path distance from x to y, given two input nodes a and b, and one output node c, the input estimated to have the greatest effect on c is given by $min(\delta(a, c), \delta(b, c))$.

Remembering that in the scope of this work, relative importance is defined as the main effect on the output from one node given a change in a different node, the distance between nodes on a data flow program's actor graph does not necessarily have a relationship to the magnitude of importance. The lack of a clear relationship between node distance and importance can be shown through example, such as by the models in Figure 4.3. In Figure 4.3 (a) the two input nodes *Node A* and *Node B* may produce tokens of value $\{0, 1\}$, while the other source nodes produce tokens with the value of their label. The remaining nodes perform the operation described by their label (multiplication or addition). The model in Figure 4.3 (a) evaluates the function $Out = A \times 2 \times 2 + B$; the model in Figure 4.3 (b), $Out = A \times 0.1 \times 1 + B$. Because this model has no nonlinear interactions between non-constant signals, only main effects exist. The contribution of *Node A* on the output can be obtained from the product of individual multiplication operation coefficients between *A* and the output, which is 2×2, or 4. Similarly, the contribution of *Node B* on the output is 1. *Node A* having a greater contribution than *Node B* is opposite what would be expected if importance were determined by the shortest path between nodes, as the closest input to the output in this example is *Node B* with a distance of 1, against the input of *Node A* with a distance of 3.

The magnitudes of contribution are reversed for *A* and *B* in the model depicted by Figure 4.3 (b). In this counter example, the contribution of input *A* on the output is 0.1 and the contribution

of B on the output is 1. Since the underlying graph structure was not changed, the shortest path between A and the output is still 3 nodes; compared to a path length of 1 node for B. This example shows that distance to an output alone is not necessarily a good, or consistent, predictor of an input's importance.

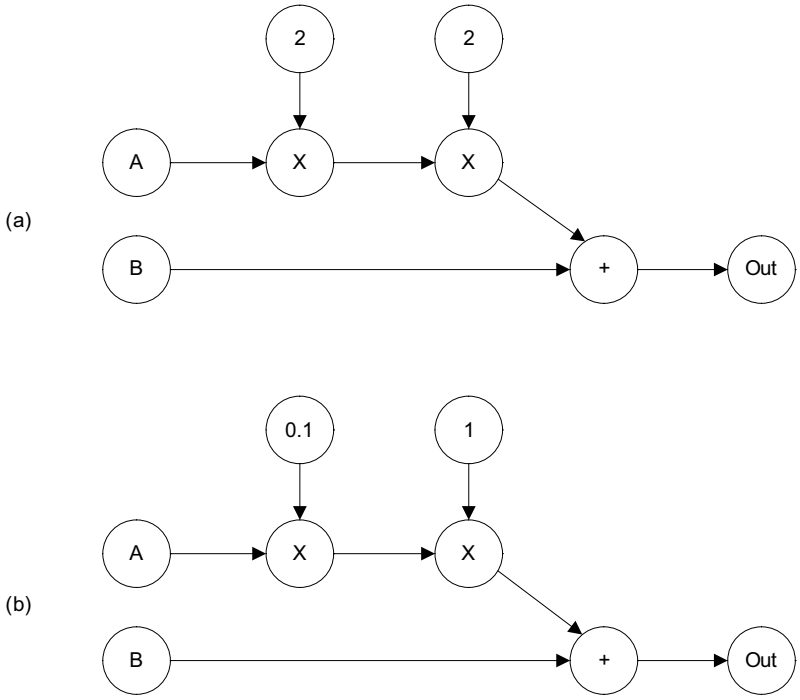

Figure 4.3: Example data flow models demonstrating influence of distance.

This isn't to say that information about distance can never be useful. Providing that certain assumptions can be made, finding the path length between inputs and an output may be sufficient to identify important factors. If a model were to be constructed with structures and values similar to either of those found in the models of Figure 4.3, path distance between nodes would be an effective way of estimating an input factor's relative importance to an output. However, unless of a very specific kind, identifying models where path distance could be appropriately applied to importance identification may be difficult, as the exact behaviors of actors in the program are not always known. Even if detail about these behaviors could be approximated through experimentation, there are still a number of difficulties in using path distance for importance identification.

Perhaps the most obvious of these difficulties is how to handle inputs and outputs that are connected to each other by more than one path. One possible approach around this difficulty

would be to measure importance by the shortest path length in the subgraph connecting a node to an output. Considering only one path in this way is problematic because, as was noted in the literature review of graph structure analysis, it is often inadequate to describe the relationship between two nodes by only one of their connections. Faloutsos, Mccurley, and Tompkins (2004) additionally propose that "any automated mechanism to pick the most important path will make mistakes", an error especially likely in this context given the uncertainty of actor behaviors in data flow programs. Even more complicating is the potential for cycles to exist in the full connection subgraph. It seems clear that to obtain a useful measure of relative importance, a method for simultaneously considering the entire connection subgraph is required. By viewing the graph as a stochastic process, the second class of algorithm to be explored concisely addresses these issues.

4.3 Random Walks for Importance Estimation

Random walks for importance identification are discussed in the literature review of graph structure analysis in Section 3.3.3. What follows is the motivation behind the application of such methods for the identification of importance in data flow graphs, followed the modification of such an algorithm for application to the ranking of relative importance between an output and a model's inputs.

These random walk methods are based off the concept of representing a graph as a Markov chain. A Markov chain is a discrete-time probabilistic model easily applied to graphs, that is useful for studying a system of states and transitions between those states. The behavior of a Markov chain model can be completely described by a matrix of one-step transition probabilities **P** and a corresponding initial probability vector $\mathbf{p}^0$.

For representation as a Markov chain, nodes on a graph are considered different states and arcs indicate transition probabilities between those states that may be non-zero. Given a single token on the graph at node *i* at time *n*, **P** for this system is made up of some fixed probabilities p_{ij} that the token will transition to node *j* at time *(*n+1). The token will only be allowed to transition between nodes if they are connected in the graph; $p_{ij} = 0$ if node *i* is not connected

to node *j*, $1 \geq p_{ij} \geq 0$ if *i* is connected to *j*. $\mathbf{P}^m$ then gives a matrix of m-step transition probabilities for the system. This m-step transition probability matrix can be multiplied with $\mathbf{p}^0$ to determine the m-step probability of being in a state given an initial probability distribution, an importance metric described in Section 3.3.3.

A more detailed explanation of how a graph may be represented as a Markov chain was given in Section 3.3.3, specifically Equation (3.4), Figure 3.13, and their supporting narrative. Following the methods described both above and by the examples of Section 3.3.3, any actor graph of a data flow program may be transformed into the basic structure of a Markov chain model. Although lacking specific arc weights, and therefore concrete transition probabilities, such a structure will indicate which transition probabilities may be greater than zero, as indicated by an arc on the data flow graph.

Take for example the data flow graph in Figure 4.4 and corresponding Markov chain structure in Figure 4.5. To satisfy the requirements that the one-step transition probability matrix be a stochastic matrix, as described by

$$0 \leq p_{ij} \leq 1, \forall i, \forall j \text{ and } \sum_j p_{ij} = 1, \forall i \tag{4.1}$$

A self loop is added to *Node G* in the Markov chain structure. In the matrix representation of the Markov chain single-step transition probability structure, the first column and row refer to *Node A*, the second to *Node B*, etc, and p_{ij} is an element's value to indicate a probability that, while unknown, may be greater than zero due to the existence of an arc on the data flow graph. If a node on the data flow graph has only one output arc, the corresponding probability must be 1 to satisfy the requirement that the sum of transition probabilities out of a state must equal 1; **P** must be a stochastic matrix.

Regardless of the unknown probabilities, since *State G* is the only absorbing state in this system, and since *State G* is reachable from all other states, any initial probability distribution will tend towards $\mathbf{p}^{(t)} = (0, 0, 0, 0, 0, 0, 1)$ as *t* approaches infinity. Or, using the fictional token example introduced in Section 3.3.3, regardless of where a randomly walking token is placed on this example graph, it will eventually end up trapped in *State G*. Another interpretation

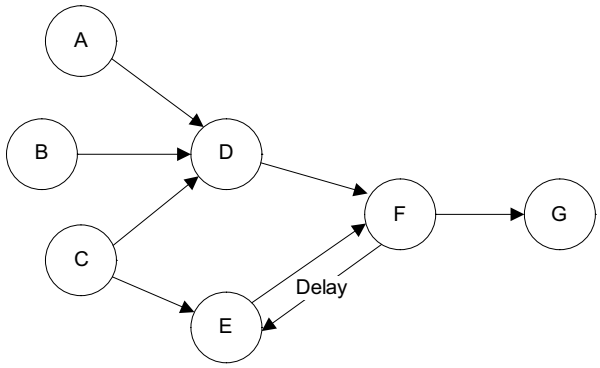

Figure 4.4: Example data flow graph.

$$\mathbf{P} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_{3,4} & p_{3,5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p_{4,6} & 0 \\ 0 & 0 & 0 & 0 & 0 & p_{5,6} & 0 \\ 0 & 0 & 0 & 0 & p_{6,5} & 0 & p_{6,7} \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{7,7} \end{pmatrix}$$

Figure 4.5: Example one-step transition probability matrix for data flow graph.

might be that *Actor G* has an importance of 1 to each of the input actors *A*, *B*, and *C*.

Now reverse the direction this fictional token walks across arcs. Transitions will now take place from the head of a directed arc to that arc's tail; a token starting in *State G* will transition to *State F* with a probability of 1. Instead of the sink node, the source nodes must now be made into absorbing states through the addition of a self loop. The structure of the Markov chain built under this assumption, shown in Figure 4.6, is also an absorbing system because an absorbing state is reachable from all non-absorbing state.

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ p_{4,1} & p_{4,2} & p_{4,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{5,3} & 0 & 0 & p_{5,6} & 0 \\ 0 & 0 & 0 & p_{6,4} & p_{6,5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4.6: Example one-step transition probability matrix for data flow graph.

If the fictional token were now released from *State G* it would end up in one of the source node representing states; either *State A*, *State B*, or *State C*. This method of reverse random walks considers the entire connection subgraph between the source and sink nodes and results in a measure of relative importance between them. This measure of relative importance is expressed in the form of which absorbing state, or the inputs in this model, a randomly walking token is most likely to end up trapped at given this token's release from an output.

From a different perspective, variations of this method will attempt to recursively account for importance, and therefore magnitudes of causality. *Node G* in Figure 4.4, for example, has only one input arc. Following the constraints of data flow programming, *Node G* must make decisions about what value to output influenced externally only by information it has received along this arc. Because of this, any changes to the output of *Node G* must be attributed to changes in data it has received along its input arc, or any random variables determined internally by *Node G*. Similarly, any change to the output of *Node F*, which has two input arcs, must be attributed to a change in either, or both, of those two inputs. In general, 100% of a change in a deterministic actor's output between replications must be attributed to a change in its inputs. If it were known exactly which inputs contributed exactly how much, transition probabilities could be assigned proportional to these amounts. All that this section's presented method does then, is propagate these importance's backward through the actor graph to the inputs.

The requirement that actors operate independently, receiving input only through their input arcs, is one reason why data flow programs lend themselves so well to this type of analysis. The added constraint of synchronous data flow that all inputs must receive a constant amount of data each iteration makes accounting for these changes a more manageable task. If the amount, not just specific value, of data received at an input was variable, it would be more difficult to identify which of an actor's inputs are most important.

With an appropriate methodology for selecting the exact values for the currently unknown members of **P**, the resulting stationary state probability vector could be used to identify which input factors are most important to determining the value of the output under study. Two such methodologies will be presented, one with uniform probabilities in Section 4.3.1 and one requiring arc weights in Section 4.3.2.

4.3.1 Uniform Random Walks for Importance Estimation

One possible method for choosing values for the missing transition probabilities of the reverse Markov chain structure of Figure 4.4 would be to assign them in a uniform way, assigning each unknown in a row to have the same probability.

Consider the actor in Figure 4.7. When fired, this actor evaluates some function, *F*, of its inputs *X*, *Y*, and *Z*. This actor could be part of any data flow program, perhaps as *Node D* in the model of Figure 4.4. From the actor's perspective it has three inputs which will be supplied with data. The actor is indifferent about the root source of this data, all it does is receive the data, perform some operation on the data, then pass the results of this operation along the data flow graph.

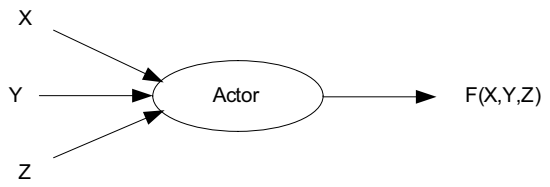

Figure 4.7: General representation of an unknown data flow actor.

It would be of great benefit to know which of the given actor's inputs are most important to determining the value of its unknown function, $F(X, Y, Z)$. Since this question is unanswerable without further information, it might be assumed that *X*, *Y*, and *Z* contribute equally. Under this assumption, the probability of transition out of the state representing this actor in the Markov chain model should be uniform; all three arcs should indicate a one third probability of transition. In general, this can be accomplished by normalizing each row in the one-step transition probability matrix so all elements of a row are equal and their sum is 1. Applying this assumption to the model depicted in Figure 4.4 results in the extension of Figure 4.6 into the complete one-step transition probability matrix shown by Figure 4.8.

Now, the only missing information necessary to compute the relative importance of an input with respect to an output using random walks is an initial probability distribution vector for the output under study. For the only sink node in this program, *Node G*, this vector is $\mathbf{p}^0 = (0, 0, 0, 0, 0, 0, 1)$; or a fictional randomly-walking token will start at *Node G* with a probability or 1.00. The vector representing the solution to the long term behavior of this system can be

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0.33 & 0.33 & 0.33 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 4.8: One-step transition probability matrix for example data flow graph, generated using uniform assumption.

approximated numerically by recursively computing $\mathbf{p}^t$ using the equation $\mathbf{p}^{(t+1)} = \mathbf{p}^t \times \mathbf{P}$ for a sufficient number of iterations. Numerical approximation is favored over an exact solution since calculating an exact solution is computationally difficult on large models. For this example, $\mathbf{p}^t$ will approach $\mathbf{p} = (0.22, 0.22, 0.56, 0, 0, 0, 0)$ as t becomes very large. The interpretation of this outcome is that *Node C* is the most important input to *Node G*, followed by a tie between *Node A* and *Node B*.

From a top-down perspective, it makes sense to rank *Node A* and *Node B* as having equal importance. All three of this example's source nodes have a single directed input arc to *Node D*. Since, for both *Node A* and *Node B*, their input to *Node D* is the only channel through which these inputs may influence this model, and since without further information it is impossible to distinguish between their individually specific influences on *Node D*, it seems reasonable to consider them equally important.

Furthermore, it also makes sense to rank *Node C* as the most important input with respect to *Node G* in this example. Following the same logic used to justify the equivalent scoring of *Node A* and *Node B*, it is clear that *Node C*, which also contributes one input arc to *Node D*, must have at least the same importance score as the first two inputs. Through *Node D* is not the only way *Node C* influences this model, however, as it is also connected to *Node G* recursively through *Node E*. It is this additional connection that causes this method to rank *Node C* as more important than *Node A* and *Node B*, and therefore the single most important input to this model with respect to *Node G*.

Even though referred to as a model, the graph of Figure 4.4 does not provide any information about the behavior of its actors and is therefore merely the structure of a model. What

Figure 4.4 does show are the aspects of a model easily retrieved from a data flow program, or the model's components and their interconnections. As previously stated, no guarantee is made that the function mapping an actor's inputs to output is easily retrieved, or even expressible- even though this detailed information must exist for program execution to be possible.

Without this detailed information on implementation, it is impossible to determine if the proposed importance measures for the inputs of this example are good. Even without testing, it is still easy to see one of the main limitations of this uniform method, which is the case in which the assumption of equal input importance for an actor is invalid. This assumption of uniform importance, although very powerful when appropriate, is very likely invalid for many actors in a data flow program. Consider again *Node D* in Figure 4.7. While we don't know the behavior of this node from the program's structure, a more detailed examination might reveal the output of *Node D* to be $D(A, B, C) = 2A+B+C$. Although the weightless method assumes *A*, *B*, and *C* to have equal importance to $D(A, B, C)$, in reality *A* appears more important than *B* or *C*. If this had been known beforehand, the importance estimate for *Node A* could have been adjusted to be twice that of *Node B*. Unfortunately, specifics about $D(A, B, C)$ were not known ahead of time, forcing the previously mentioned assumption of uniform importance. Real world simulation applications often involve complex functions where this uniform importance assumption would be expected to hold up poorly.

4.3.2 Weighted Random Walks for Importance Estimation

In the development of a uniform reverse random walks method in Section 4.3.1, it was shown that for reverse random walks to be useful for factor screening in a large variety of models, some methodology for weighting the one-step transition probabilities in a Markov chain representation of a data flow graph must be developed. This section contains the development of such a method, called here "Weighted Random Walks for Importance Estimation" and abbreviated "WRW", as well as a full example of the resulting method's application to a simple model with a tree structure. The result is a method similar to the Pathway Participation Method (PPM) developed by Mojtahedzadeh et al. (2004), in that it studies individual model components to determine which inputs to that component are most important. The WRW method differs greatly

from the PPM method in the way it utilizes this importance information; specifically WRW's consideration of the entire subgraph connecting inputs to an output of interest.

4.3.2.1 Weight Selection

The largest challenge to the development of a weighted random walks method for importance identification is the generation of weights for use by such an algorithm. To be useful, the resulting weights must be related in some way to importance. As defined by the scope of work in Section 4.1, importance is synonymous with absolute value of main effect. Thus, the arc weights used for importance identification should be heavily related to main effects in the model under study.

To compute arc weights based on main effects, it is no longer sufficient to consider only a given model's structure. Individual nodes must be studied and certain properties of the data flow program need to be determined through full execution of that program. Specifically, if main effects are to be assigned as weights, some experiment needs to be performed to determine these main effects.

The actor shown in Figure 4.9 takes three inputs, computes the specified function, and produces the result of the specified function on its output arc. This exact actor was discussed in Section 4.3.1 as an example where the weightless version of the reverse random walks method would not work well. Assuming high and low factor values of 0 and 1 for X, Y, and Z, it is easy to compute main effects for this function. Through examination of the function F in this example, these main effects are 1 for inputs Y and Z, and 2 for X.

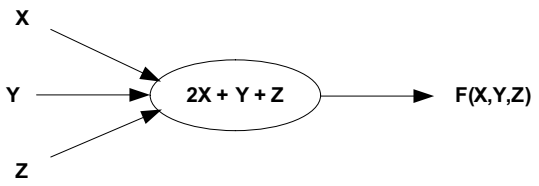

Figure 4.9: Example linear expression actor.

Although it is not always possible to examine F, it is still possible to determine the main effects of the input factors through the use of a designed experiment on the actor node. Even though F can not be examined, it can still be evaluated by executing its implementing actor. For

this example with no interaction effects, it would be possible to compute the main effects for the three inputs through a fully saturated factorial experiment. This would require 4 experimental runs on the actor under examination.

However, it is not always possible to assume no interaction between input factors. As stated in the review of factor screening in Section 3.1, one limitation of a fully saturated experiment is the aliasing of main effects with potentially important interaction effects that occurs. This limitation can be avoided through the use of a more complete fractional factorial experiment, or even a full factorial experiment. Unfortunately, while the fully saturated experiment only requires 4 runs for this example, a full factorial experiment would take 9 runs to compute all of this actor's main effects. The completeness of this experimentation needs to be balanced with run time.

The main effect of an input on the output of this example actor also depends on the magnitude of the input factor levels. The main effects values of 1 for inputs *X* and *Y*, and 2 for *Z* were based on high and low factor values of *0* and *1* for *X*, *Y*, and *Z*. While appropriate high and low factors will be known with certainty for the global model inputs (this is part of the problem definition as detailed in Section 4.1) how these global inputs interact to determine the ranges which will be seen by the individual intermediate actors is not straight forward. In other words, if inputs *X*, *Y*, and *Z* in Figure 4.9 are global inputs to the model, it will be known what values these inputs may take. The determination of permissible values for global model inputs is a question to be answered by a simulation analyst. On the other hand, if inputs *X*, *Y*, and *Z* in Figure 4.9 take their values from other actors, their ranges will not be known ahead of time.

Although not known, the ranges for these intermediate inputs can be estimated through experimentation. Three things may cause the inputs to an intermediate actor's output to change. The first such cause would be if the model's structure itself were altered, which is not a case of interest to this work. Second, inputs to an intermediate actor may change between iterations of a replication if the model contains recursion, as shown in Section 3.2.3. Finally, these intermediate values may change if the global model inputs are changed. It is the second two causes which must be studied and accounted for in the determination of appropriate arc weights. Two factor levels are required to use a two level experiment, as suggested for this method of arc

weighting. These two levels should be based off the possible range of values that may occur at any given node.

A further complication to assigning arc weights exists if the function evaluated by an actor is not linear. The actor pictured in Figure 4.10 behaves similarly to the actor in Figure 4.9, in that it takes three inputs and produces an output by evaluating the function shown in its label. Figure 4.10 is different in that its function is not linear. Additionally, the information provided in Figure 4.10 is also different in that it includes information on the range of values which may be received at the inputs during program execution. Because there are no interaction terms, the magnitude of main effect for inputs *X* and *Y* depend only on their coefficients in the actor's function and the difference between their high and low factor values. In this simple example, the main effect of *X* and *Y* can be computed by multiplying the range of the factor values by the coefficient of the input in the expression. For *X*, this main effect is 2, and for *Y* it is 10. Since it is squared in the given expression, input *Z* depends not only on the range of its inputs, but on their exact values. In this example, *Z*'s main effect will be $100^2 - 99^2 = 199$.

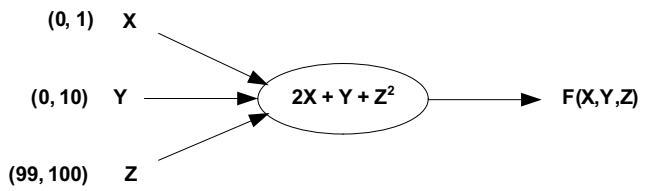

Figure 4.10: Example non-linear expression actor including (min, max) input ranges.

The most complete way to determine the range of values an intermediate input may encounter would be a full factorial experiment on the data flow program under analysis. This is not practical though, as full experimentation would eliminate the need for this analysis method in the first place. Luckily, as is the case with many phenomenon, it is possible to find a decent approximation of the intermediate input distributions through a small random sample of input configurations to the full model. An input configuration is defined as a unique setting of global model inputs; essentially one row from a tabular view of a full factorial 2^k experiment.

The steps for approximating the intermediate distributions through sampling are as follows.

1. Identify global model inputs and their permissible ranges.

2. Decide on an appropriate input configuration sample size and structure.

3. Generate the specified number of random input configurations.

4. Execute the simulation model for one or more replications under each of the generated input configurations, recording intermediate values passed along the data flow arcs.

5. For each arc, sort by ascending order the values recorded as passed along that arc. Choose a high and low factor value based on this list.

Step 1 above needs to be done before any factor screen techniques can be carried out, and the availability of this data is assumed to be known ahead of time. Step 2 involves setting the adjustable sample size parameter of this method and, by performing the actual generation of input configurations, Step 3 follows through on the work generated by Step 2. Step 4 is where data is collected by observing the values passed along intermediate arcs. Actual selection of the high and low factor values happens in Step 5, which introduces another adjustable parameter to this process. Depending on the model type, it may be appropriate to set the individual arc factor values based on the minimum and maximum values observed. Alternatively, it may be more appropriate to set these values as some upper and lower percentiles of observed values. Because Step 4 will result in such a large amount of data being generated, it may be necessary to collect only a sample of that data. For instance, if Step 5 is only interested in the minimum and maximum values, it would be sufficient to record and update only these two values.

It is only necessary to record arc weights in Step 4 for actors with more than one input. This is important, because it will speed up the overall run time for this method by reducing the amount of data that will need to be recorded and analyzed. Not studying actors with fewer than two input arcs is possible because these actors belong to a special type of state in the derived Markov chain. If an actor has no inputs, it is a source node and has no inputs that require weighting. If an actor has only one input arc, the transition probability out of the state which will represent that node must be 1 to conform to the requirement that the one-step transition probability matrix be a stochastic matrix.

Since it requires executing the entire simulation model, this described process of sampling intermediate values is very expensive. Because of this high cost, it is very important to use as

small a random input configuration sample as practical, and to make the most of each simulation replication. One way to improve the quality of the sample is to build that sample as a random balance design. Especially if that sample is small, a random balance design will help distribute the sample more evenly, by assuring each input is set high and low in an equal number of configurations.

As mentioned, this described process of arc weighting through experimentation and observation introduces three adjustable parameters into the reverse random walks process. One parameter in particular, the number of random input configurations used to sample the distribution of intermediate values, involves a large trade-off between speed and accuracy. Details of this trade-off will be examined through testing in the results section, specifically Chapter 6.

4.3.2.2 Special Considerations for Actors with Multiple Outputs

Until this point, only actors with one or zero outputs have been considered. Although the output from a node may be read by multiple other nodes, those multiple other nodes all read the same value. Not only is it possible for actors to produce more than one output as shown by Figure 4.11, it is quite common.

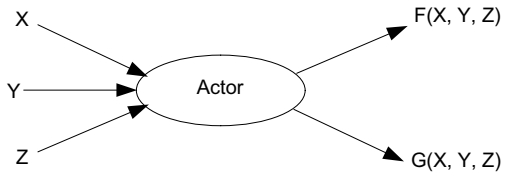

Figure 4.11: Actor with multiple outputs.

It is not required that every input to an actor be important to the determination of all output values produced by that actor. It is not even required that an input important to one output be important to all other outputs; Input *X* may be important to Output $F(X, Y, Z)$ but not important to Output $G(X, Y, Z)$. This has implications not only to factor screening but also the scheduling of actors in synchronous data flow, and is a focus of the causality interfaces developed by Zhou and Lee (2008).

To cope with the added complexity of multiple outputs from a single actor for methods using structural analysis, the actor graph is elicited into a form consisting of nodes on the

output level, or a graph with one node per output. The actor shown in Figure 4.11 would be represented by two nodes in the elicited graph, structurally arranged as shown by Figure 4.12.

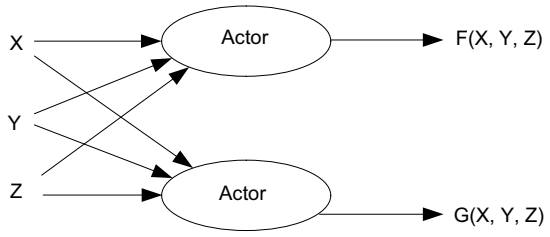

Figure 4.12: Actor with multiple outputs expressed as multiple nodes in graph elicitation.

This representation as multiple nodes is only made in the elicited graph that will be analyzed structurally. Although the resulting Markov chain used for analysis of this program will have a greater number of states, the data flow program will remain the same. A list is kept that facilitates look-ups between the two representations. Because both nodes in Figure 4.12 represent outputs from the same actor, the results from any weighting experiment performed on that actor can be used to populate edge weights for input arcs to both elicited nodes- the evaluation of an actor requires computing all of that actor's outputs, in this example both F and G.

4.3.2.3 Special Considerations for Actors with Internal State

As was noted in the review of data flow programming in Section 3.2, actors in a data flow program may contain internal state, or memory of past events. While the behavior of stateless actors may only be influenced by previous events or time through a directed cycle, an actor with internal state may exhibit time-varying behavior by itself. A common example of an actor with internal state is an accumulator actor. An accumulator produces a value equal to its current input value plus the sum of all past input values.

Because the behavior of their outputs may change over time, any experiments performed on an actor with internal state will be carried out here for the same time-frame as the full model under examination. Unfortunately, this is a simplification of the actual actor behavior, since no attempt is made by the weighting method to imitate changes to intermediate input values over time; the factor values will be set, and remain constant, for the entire time-frame of the

weighting experiment.

This is also unfortunate, as it means any experiments on actors with internal state will take longer than experiments on actors without internal state. While a single iteration for each experimental run is always adequate for determining the response of stateless actors, an actor with state will require more. This is a consequence of course granularity in a model; the price for modeling convenience is paid in the analytic information encoded in that model's graph.

4.3.2.4 Mechanics of Weighted Reverse Random Walk Algorithm

After selecting arc weights, the process of importance ranking continues as described in Section 4.3.1 for the uniform reverse random walks method. A one-step probability matrix can be built by normalizing the arc weights to conform to the requirements detailed in Equation (3.3) that the sum of each row equal one and each individual non-zero element be between zero and one. Because only magnitude of importance is of interest, this step includes taking the absolute value of any main effects computed. After creating a one-step probability matrix, computation of relative importance can continue exactly as described for the uniform variation of this method. The full process of applying this method for factor screening in a synchronous data flow program is:

Step 1: Perform any graph elicitation necessary to obtain a pure graph representation of the data flow program (details of why this is necessary are described in the Scope of Work, Section 3.4).

Step 2: Generate the reverse Markov chain model structure following methodology first described in Section 3.3.3. Set the transition probability out of any state representing an actor with only one input arc as 1 and add self-loops to states representing source nodes on the data flow graph.

Step 3: Decide on an appropriate sample size of random input configurations for use in arc weighting and generate the list of required random configurations following a random balance structure.

Step 4: Execute the simulation model at least once for each random input configuration, recording the values passed along arcs that need to be weighted.

Step 5: Select high and low factor values for arcs as some upper and lower percentile of the values recorded by Step 4.

Step 6: Weight any required arcs using two-level designed experiments with the factor values selected in Step 5 on the individual model actors.

Step 7: Normalize arc weights from Step 6 by destination node and use the results to populate the remainder of the one-step transition probability matrix from Step 2.

Step 8: Compute relative importance scores by selecting an appropriate initial probability vector for the output, or outputs, of interest then solving for long term behavior of resulting model. Repeat this step as needed for analysis of multiple model outputs.

These detailed steps can be grouped into the four main stages of the Weighted Random Walks algorithm, summarized and described through pseudocode below.

Stage 1: Pre-processing, Step 1 and 2:

Require: Graph representation G of model under study.
1: $A \leftarrow$ List of all actors in G.
2: $P \leftarrow |A|$ by $|A|$ single-step transition probability matrix with elements initialized to 0.
3: **for all** Actor a in A **do**
4: **if** A is a source node **then**
5: $P_{a,a} \leftarrow 1$ {Assign self loops.}
6: **else if** A has one predecessor. **then**
7: $q \leftarrow Predecessor(a)$
8: $P_{a,q} \leftarrow 1$ {Assign transition probabilities that must be 1.}
9: **else**
10: Add a to list $NeedsWeights$.
11: **end if**
12: **end for**

Stage 2: Intermediate range sampling, Step 3 and 4:

Require: Stage 1 variables.
Require: An integer $nReps > 0$ number of random configurations to run.
Require: An integer $TimeSpan > 0$ number of model iterations to perform analysis for.

1: $RB \leftarrow$ Generate Random Balance experiment with $nReps$ replications.
2: $RB_{Results} \leftarrow$ Execute all replications of RB for $TimeSpan$ iterations, recording all values passed across input arcs to actors in list $NeedsWeights$.

Stage 3: Assignment of arc weights through experimentation, Step 5, 6, and 7:

Require: Stage 1 and Stage 2 variables.
Require: A number $0.5 > factorPercentile \geq 0$ to select high / low factor values for individual arcs.

1: **for all** Actor a in $NeedsWeights$ **do**
2: **for all** Actor p in $Predecessors(a)$ **do**
3: $Vals \leftarrow RB_{Results}(p, a)$ {Assign to $Vals$ the list of all values passed from p to a during execution of RB.}
4: $SortAscending(Vals)$
5: $LowFactor(p, a) \leftarrow Vals[\lfloor |Vals| \times factorPercentile \rfloor]$ {Low factor value for arc (p, a).}
6: $HighFactor(p, a) \leftarrow Vals[1 - \lceil |Vals| \times factorPercentile \rceil]$ {High factor value for arc (p, a).}
7: **end for**
8: **end for**
9: **for all** Actor a in $NeedsWeights$ **do**
10: $FF \leftarrow$ Generate full factorial experiment for node a using factor values for arcs as previously defined in $LowFactor$ and $HighFactor$.
11: $FF_{Results} \leftarrow$ List of main effects resulting from execution of FF.
12: $TransitionProb \leftarrow Normalize(FF_{Results})$ {Normalize main effects into single-step transition probabilities for arcs into a. Direction of transition will be wrong until transposed on line 16.}
13: **for all** Actor p in $Predecessors(a)$ **do**
14: $P_{a,p} \leftarrow TransitionProb(p, a)$
15: **end for**
16: **end for**

Stage 4: Algorithm to determine relative node importance on graph, Step 8:

Require: Stage 1, 2, and 3 variables.
1: $p^0 \leftarrow$ Vector of size $|A|$ with elements initialized to 0.
2: $p_y^0 = 1$ {Set initial probability vector to indicate starting at output of interest with probability of 1}.
3: **repeat**
4: $\quad p^{(t+1)} = p^t \times P$
5: **until** $StoppingCondition$
6: **return** p

4.3.2.5 Example Application of to Tree Model

The process of arc weighting and reverse random walks will now be demonstrated through a full example. The model shown in Figure 4.13 is a data flow model of a linear signal flow graph, as described in Section 3.3.1. The model contains nine inputs, labeled *X1* through *X9*. In this example these inputs will be constrained to a range of $[0, 1]$. Every non-input node in this model takes three inputs and outputs a sum of those inputs weighted by the arc each input arrived on. The *Y1* actor, for instance, will generate an output value equal to $Y1(X1, X2, X3) = 1.88X1 + 1.24X2 + 1.38X3$. The structure of this model was explicitly set to be a tree with 9 leaves and one intermediate layer, but the coefficients were randomly assigned as a number between 0 and 2.

This example was specifically designed to work well with the developed factor screening method and also has a couple of properties that make it very easy to analyze through inspection. Since it contains only linear functions, there are no interaction effects. Furthermore, the main effects for the various actors are very easy to compute without the need for a full factorial experiment. Finally, as it contains no cycles, the long term behavior of the derived Markov chain will be easily determined.

Also interesting about this example is that the data flow graph structure alone contains no information distinguishing different importance between nodes. All input nodes are exactly 2 nodes away from the output *Z1*, and each intermediate node has three inputs. Any process designed to rank the factors' importance based on the actor graph structure alone, such as the uniform reverse random walk method of Section 4.3.1, or any methods based on path lengths

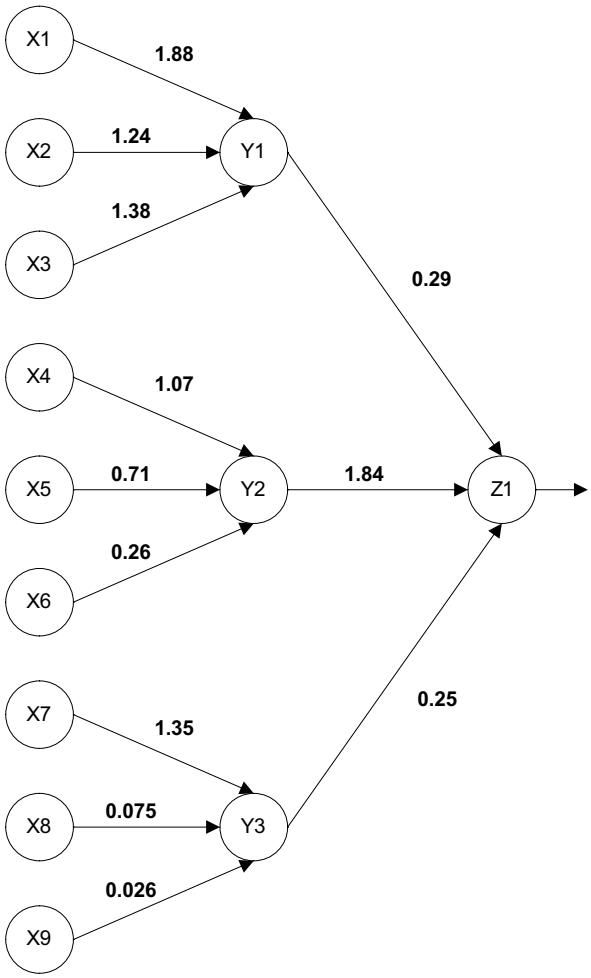

Figure 4.13: Example model with a tree structure.

as described in Section 4.2, will conclude that all nine inputs are of equivalent importance to the output. This is an example of when methods operating on structure alone may work poorly unless supported with information about the behavior of the model's individual actors.

The first step to generating such supporting information, as detailed in Section 3.3.3, is to obtain a usable representation of the data flow graph for the program under analysis. The results for this step are contained by Figure 4.13. The second step is to translate the graph into the structure of a Markov chain, the results of which are shown in Figure 4.14 for this example. As described in Section 4.3.1, $p_{i,j}$ in this matrix represents a transition probability that, while unknown, may be greater than zero due to the existence of an arc between nodes on the data flow graph. For additional clarity, and due to the size of this matrix, the name of the actor represented by a given row has been added in the left most column, separated from the

rest of the matrix by a vertical bar. This column should not be considered part of the one-step transition probability matrix when performing operations such as matrix multiplication.

$$\mathbf{P} = \left(\begin{array}{cccccccccccc|c}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X2 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X3 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X4 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X5 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X6 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & X7 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & X8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & X9 \\
p_{10,1} & p_{10,2} & p_{10,3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y1 \\
0 & 0 & 0 & p_{11,4} & p_{11,5} & p_{11,6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y2 \\
0 & 0 & 0 & 0 & 0 & 0 & p_{12,7} & p_{12,8} & p_{12,9} & 0 & 0 & 0 & 0 & Y3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{13,10} & p_{13,11} & p_{13,12} & 0 & Z1
\end{array}\right)$$

Figure 4.14: Markov chain structure for example model with a tree structure.

The next step is to choose an appropriate sample size of random input configurations. This is the first time a decision has needed to be made in the application of this method for factor screening. The goal of this step is to select a sample size large enough to obtain a good approximation of the distribution of values that will be passed as intermediate values along arcs on the data flow graph. Although potentially interested in more detail about these distributions, in an effort to simplify this example, only the minimum and maximum of values passed will be recorded.

In this example, the value passed along the arc connecting actor *X1* to *Y1* will be 0 if *X1* is set to its low value and 1 if *X1* is set to its high value. This is because (0, 1) is the range of *X1* (and all other inputs for this example) as stated by the problem definition. Although possible in small examples, it is unlikely that every possible intermediate value will be seen during this sampling of input configurations. Structuring the sample to be a random balance will improve the chance but it is not guaranteed that both the true minimum and true maximum values for every arc will be seen in this small sample. Luckily, this factor screening method is expected to be robust to less than perfect range estimates. Figure 4.15 shows the result of the range sampling through simulation execution procedure, or Step 4 from Section 4.3.2.4. In this

example, this was accomplished through the use of 10 randomly selected input configurations. Included in Figure 4.15 are both the range, as measured from experimentation, and best possible range, as determined analytically through inspection.

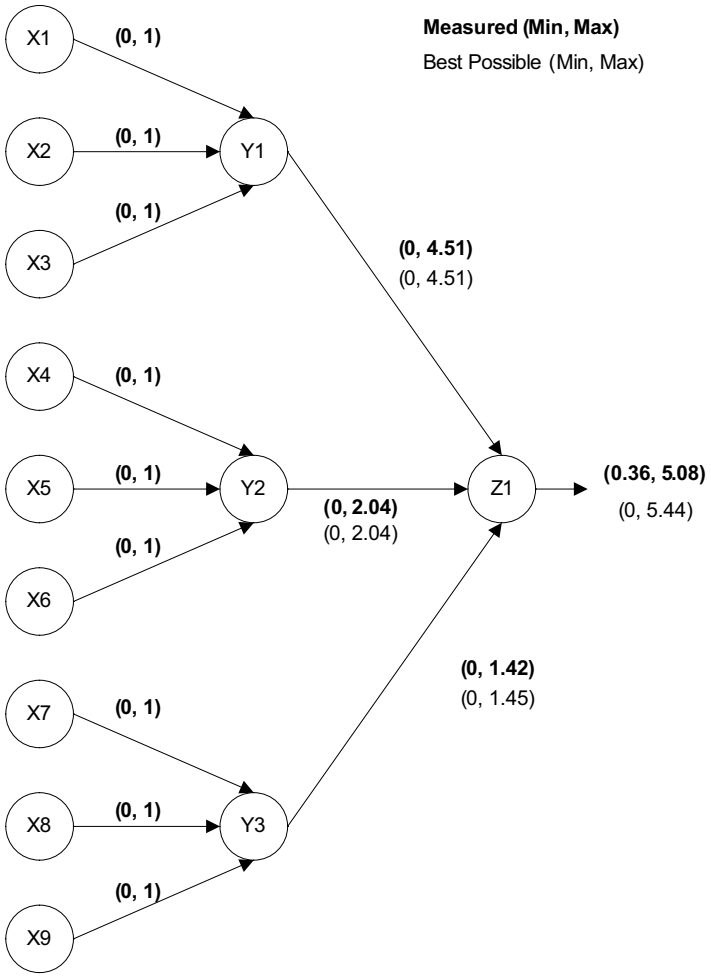

Figure 4.15: Example model with a tree structure including arc ranges.

As Figure 4.15 shows, the range of values possible on the *Y1* to *Z1* and *Y2* to *Z1* arcs were found exactly. For the range of possible values on the *Y3* to *Z1* arc, the lower bound was correctly found but the upper bound was missed slightly; the upper bound was estimated at 1.42, but is actually 1.45. The arc out of *Z1* also missed exactly finding the upper and lower bounds. This arc out of *Z1*, while not important for identifying importance in this model, is interesting as it demonstrates the degradation of estimated range accuracy, that occurs as distance from the inputs increases in this model.

From these estimated ranges, high and low factor values for the individual arcs may now be

selected, as required by Step 5 of Section 4.3.2.4. This is the second decision that has needed to be made in the application of this method. Since this model is both simple and deterministic, the high and low factor values will be chosen as the maximum and minimum values of the estimated arc ranges. If this model were highly dynamic or if this model contained any random elements, it may have been appropriate to choose some large percentiles of the range, instead of that range's bounds, to avoid including outliers.

The next step is to find the main effects of all intermediate arcs incident on the intermediate actors. As has been shown previously for the actors of this example, the main effect of an arc can be obtained by multiplying the range of its associated factor values by its coefficient in its destination's function. The resulting main effects are shown in Figure 4.16, along with the arcs normalized one-step transition probabilities. When these arc weights are normalized by-actor and placed back into the Markov chain structure originally shown in Figure 4.14, the result is the one-step transition probability matrix weighted by main effects shown in Figure 4.17; the culmination of Steps 1 through 7 as described in Section 4.3.2.4.

All that remains of the procedure detailed in Section 4.3.2.4 is Step 8, or the choosing of an appropriate initial probability distribution vector, $\mathbf{p}^0$, and the process of using this vector and the one-step transition probability matrix, $\mathbf{P}$, to compute the relative importance scores of inputs with respect to $\mathbf{p}^0$ as described in Section 4.3. For this example, an initial probability distribution vector to determine the inputs' relative importance to actor *Z1* (which is in the 13$^{\text{th}}$ row and column of the one-step transition probability matrix), would be $\mathbf{p}^0 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$. Performing the iterative computation to find the inputs' importance scores results in the importance vector

$$\mathbf{p} = (0.10, 0.065, 0.074, 0.36, 0.24, 0.090, 0.065, 0.0035, 0.0014, 0, 0, 0, 0).$$

Because this example model has a tree structure, these importance score values can also be obtained for each input by multiplying together all transition probabilities on the path between that input and the output. For *X1*, this would be $0.42 \times 0.24 = 0.10$.

The results of this factor screening method can be compared to the nine input's actual main effect on *Z0* to determine how well the presented structure based method performed. While

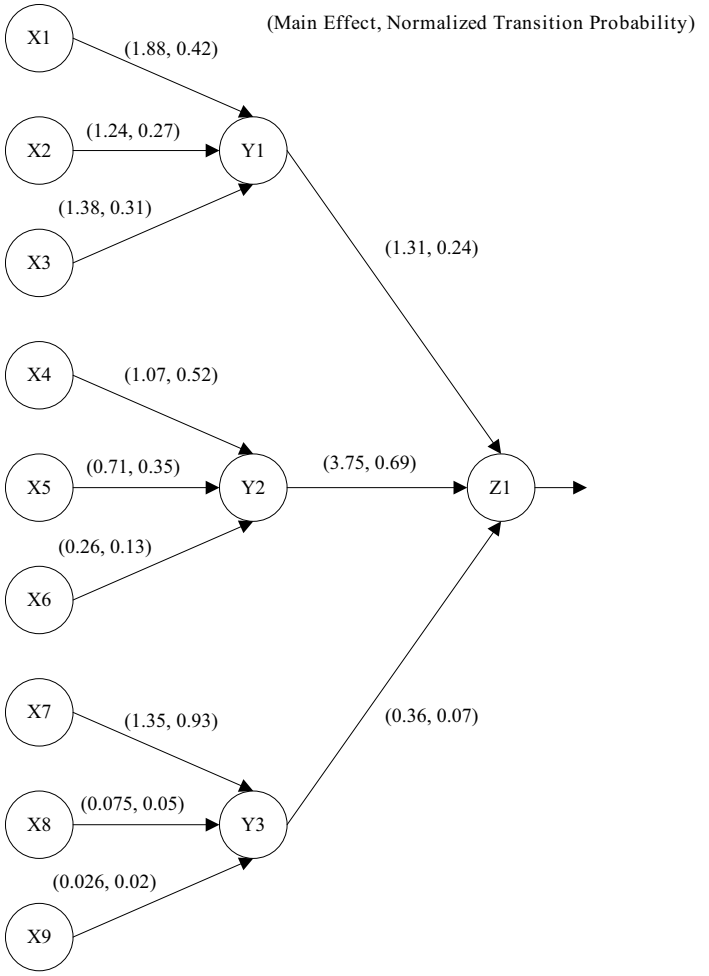

Figure 4.16: Example model with a tree structure including arc main effects and transition probabilities.

previously in this section the phrase "main effect" has been implied as local to a single given node, main effects will now be presented globally to the model. The statement "*X1* has a main effect of 0.55 on *Z0*" is used in reference to the main effect of *X1* on *Z0* given a full execution of the entire data flow program. This definition of global main effect is the same as importance as defined in the scope of work. Using the terminology given in the scope of work, the relative importance scores returned by the weighted reverse random walks method are an estimate of the importance of individual inputs, and are to be compared to the global main effect, or actual importance.

Because there are no interaction effects in this model, these global main effects were found through a full resolution III factorial experiment using the inputs' high and low factor values of

$$\mathbf{P} = \left(\begin{array}{cccccccccccc|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X5 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & X6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & X7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & X8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & X9 \\ 0.42 & 0.27 & 0.31 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y1 \\ 0 & 0 & 0 & 0.52 & 0.35 & 0.13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.93 & 0.05 & 0.02 & 0 & 0 & 0 & 0 & Y3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.24 & 0.69 & 0.07 & 0 & Z1 \end{array}\right)$$

Figure 4.17: One step transition probability matrix for example model with tree structure.

$\{0, 1\}$. The results for this example, along with the generated estimated importance scores are tabulated in Table 4.1 and presented graphically by Figure 4.18.

Table 4.1: Results for example model with tree structure.

Factor Name	**Estimated Importance Score**	**Global Main Effect**
X1	0.10	0.55
X2	0.065	0.36
X3	0.074	0.41
X4	0.36	1.96
X5	0.24	1.31
X6	0.09	0.49
X7	0.065	0.34
X8	0.0035	0.019
X9	0.0014	0.0063

Figure 4.18 shows a very clear and strong correlation between the estimated importance scores returned by the weighted reverse random walks heuristic algorithm and the actual, empirically determined, global main effects for this model's inputs. Even better is that the relationship appears linear, far exceeding the criteria set up by the scope of work. A simulation analyst may use this information in a number of ways. They may use this information to prime a more in depth factor screening method, or use these results directly to aid in some optimization.

Although these results are extremely encouraging, it is important to remember that this

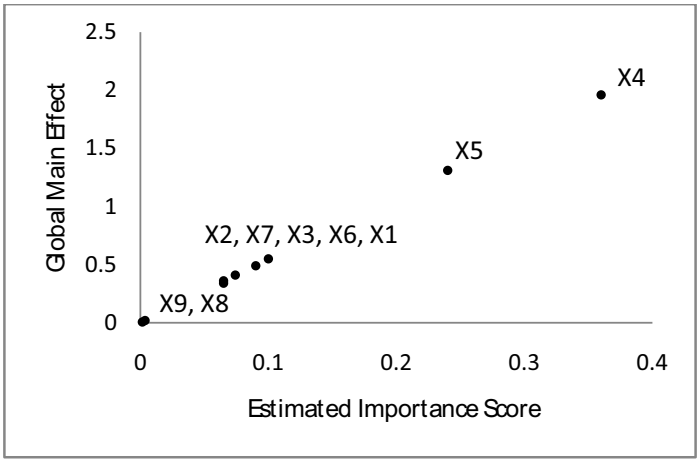

Figure 4.18: Plotted results for example model with tree structure.

model was specifically chosen as an example because it was expected to work very well with the presented factor screening method. If the WRW method presented in this section performed poorly for this example, it would not be expected to work well in more complex models. Results of application to more complex models will be presented in the experimental Chapters 6 through 7. Furthermore, due to the simplicity of the presented example, the WRW method did not run fast enough to justify its use over a more detailed statistical test. Section 4.3.2.6 contains a discussion of run time for the proposed method, with specific examples from the model used as an example in this section. Chapter 6 contains a detailed study of run times when the WRW method is applied to a variety of more complex models.

4.3.2.6 Discussion of Complexity and Run Time

For a heuristic method such as the WRW method to be valuable, it must produce a solution of acceptable quality much more quickly than it would be possible to obtain an exact solution. Section 4.3.2.5 showed that applying the weighted reverse random walks method described earlier in this chapter for relative importance identification can be reasonably accurate, in at least some models, as a factor screening method. This section will address the issue of run time.

Referring back to the example model presented in Figure 4.13, assume that the run time of this model is exactly proportional to the total number of intermediate actor firings. Since the example model has 4 actors (remembering that the inputs are parameters, not actors), running

the full model will be assumed to take exactly as long as the time required to fire any single actor 4 times. This is a gross simplification of the actual run time of a data flow program, not only because individual actors differ in their run times, but also due to the overhead of program execution associated with such tasks as setting input values and communication between actors. Even so, this simplification should perform reasonably for models with actors of similar complexity, such as the example under consideration, and permits a straightforward means of approximating run that is sufficient to demonstrate orders of magnitude in difference. The time required for the random walks portion of the weighted reverse random walks method is negligible when compared to the time required for actor firing in large models, as will be shown empirically in Chapter 6.

If not told otherwise, it may be expected that the model under evaluation will contain interaction effects between some of the 9 input factors. The model may even contain a 9 order interaction effect. To accurately measure main effects under these considerations requires a full factorial experiment with $2^9 = 512$ simulation runs, or $512 \times 4 = 2048$ individual actor firings.

Each individual actor has exactly 3 inputs, so the main effect of their inputs on their output can be calculated in $2^3 = 8$ runs. So it takes $8 \times 4 = 32$ actor firings to compute weights for each of the four intermediate actors, assuming appropriate input ranges are known. Since 10 random input configurations were used in Section 4.5.5 to sample the ranges used for arc weighting, the total number of actor firings required to use weighted reverse random walks for factor screening in this example is: $10 \times 4 = 40$ firings for range sampling, plus 32 actor firings for arc weighting makes a total of 72 node firings for the heuristic factor screen method. This means the presented heuristic will perform approximately 64 times faster on this example, compared to a full factorial experiment.

A lot of assumptions were made in Section 4.3.2.5 about the example. One assumption that greatly sped up analysis was the known lack of interaction effects. Because the model had no interaction effects, the main effects of *k* different inputs could be estimated in $k + 1$ runs using a resolution III factorial design, or 10 runs for the tree example in Figure 4.13. Since this model has 4 intermediate nodes that need to be fired in each model run, to evaluate the main effects of all 9 inputs for this example would only require 40 individual actor firings; a vast improvement

over the 2048 firings required for a full factorial experiment. Even if a lack of interaction effects could not be assumed, many of the statistical factor screening techniques presented in Section 3.2.1 rely on an assumption that the important factors in a model can be found regardless of considering any potential interaction effects. This means that any developed heuristic factor screen method should perform faster than a comparable statistical factor screening experiment, such as a resolution III or resolution IV fractional factorial experiment.

Returning to the tree structure example of Section 4.3.2.5, if true for the full model, an assumption of no interaction effects will also hold for the individual actors comprising the model. Each actor in this example has exactly 3 inputs, so the main effect of their inputs on their output can be calculated in 4 runs. Under this assumption, it only takes $4 \times 4 = 16$ actor firings to compute weights for all of the four intermediate actors, instead of the 32 required when full factorial experiments are used. This reduces the total number of actor firings required for this heuristic method to: $10 \times 4 = 40$ firings for range sampling, plus 16 actor firings for arc weighting, for a total of 56 node firings.

Compared to the 40 firings required for comprehensive statistical testing, 56 node firings is unacceptably high for a heuristic method. Not only will the heuristic provide less information about the exact nature of effects in the model, it will also take longer to run under these conditions. Even worse, more specialized factor screening techniques, such as Sequential Bifurcation, may require even fewer runs than the fully saturated experiment, making the tested heuristic even slower by comparison.

There are ways of making the heuristic perform faster. For one, the number of random input configurations used to sample the ranges of intermediate values can be reduced. In the example presented by Section 4.3.2.5, 10 random input configurations were used. If this number were cut in half to 5, the total number of actor firings required for the heuristic method would be reduced by 20, to a total of 36. The heuristic now runs slightly faster than a fully saturated experiment on this model. The difference, 4 actor firings, is probably not worth the compromised accuracy of a heuristic, however, especially since speed was gained in this case through a sacrifice in accuracy.

Run time for this heuristic could be decreased even further, as the minimum number of

input configurations that need to be sampled to estimate the intermediate values required for arc weighting is 2 (a single sample is inadequate since it provides no changes and therefore no range of values). Accounting only for time devoted to the range sampling stage, the absolute fastest run time of the heuristic should be $10 \div 2 = 5$ times faster than a fully saturated experiment.

The developed heuristic method simply does not work quickly enough to justify its use on small models. Although disappointing, this lack of usefulness is not surprising. The cost of sampling model behavior, as is done by the WRW method, is only acceptable if relatively low when compared to the cost of running a full statistical factor screen experiment. This is only possible in models with a large number of inputs, since in these models the number of replications needed to obtain a reasonable sample of intermediate values will be much smaller than the number of replications required for a saturated two-level experiment- similar to why a census is preferred to a sample on small populations.

As the number of inputs to a model increases, however, the WRW method provides significant time benefits over comparable fractional factorial experiments. For the tree model, each additional input will require an additional full simulation replication, which requires 4 additional actor firings. Assuming the new input is connected directly to one of the existing actors, the WRW method only requires 1 additional actor firing for each additional model input. The implications of these different complexity growth rates are shown in Figure 4.19.

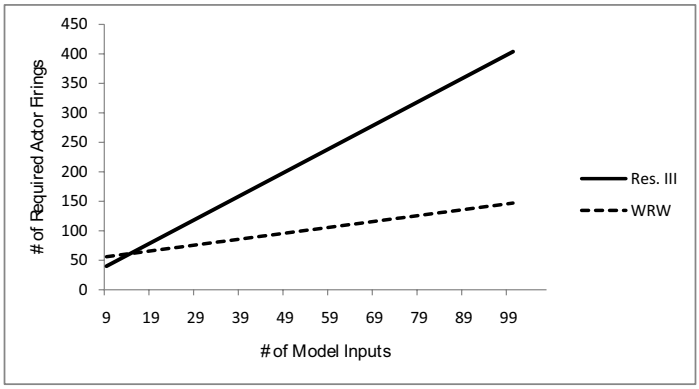

Figure 4.19: Plot of number of required actor firings by number of inputs for resolution III fractional factorial experiment and the WRW factor screening methods.

With greater than 16 inputs, the WRW method will require fewer actor firings than the

resolution III experiment. In models structured similarly to the tree model, as the number of inputs increases, so do the benefits of the WRW method over a fractional factorial experiment. This will be shown directly through experimentation in Chapter 7.

4.4 Implementation of Weighted Random Walks Algorithm

The described weighted random walks algorithm is implemented as a Java program, tightly coupled with the Ptolemy II and JUNG application environments as described in Section 4.1. Ptolemy II is the environment used for the construction and execution of synchronous data flow programs. JUNG is used to store and analyze the elicited graph structure of data flow programs.

The weighted random walks algorithm's pre-processing, intermediate arc range sampling, and weighting experiments stages are carried out through direct calls to the Ptolemy II application programming interface. Information about the resulting graph structure and computed arc weights are stored in a Sparse Multi-graph object of the JUNG project. Documentation of these application's API's can be found in their respective documentation packages.

The matrix multiplication required for the random walks portion of the weighted random walks algorithm is carried out directly on the elicited graph structure. The matrix structure is encoded in the elicited graph through a conversion process described in Section 4.1.4. The one-step transition probability values are stored as arc weights and the current probability distribution values are stored as attributes on the nodes. A list of current nodes with non-zero probability of token residence is kept. During multiplication, these probabilities are distributed, in a weighted manner, to that current node's predecessors on the data flow graph. The result is a method for matrix multiplication that is capable of skipping zero-valued elements of the current probability distribution vector, essentially an application of naïve matrix multiplication, $\mathbf{C} = \mathbf{AB}$ where $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$ for $1 \leq i, j \leq n$ that ignores elements where the product of $a_{ik} = 0$ or $b_{jk} = 0$, as described by (Gustavson, 1978). A further benefit of this method is that it allows arc weights to be normalized into transition probabilities on an *as needed* basis. No time is wasted normalizing arc weights that are not required for the current analysis.

The entire algorithm is executed in a single thread. This was done to preserve comparability

of run times against other tested methods that could take advantage of concurrency, but do not currently have concurrent implementations. If this algorithm were to be deployed in a production environment, a substantial run time reduction could be realized through exploiting concurrency. First, the individual simulation replications of the intermediate range sampling stage do not depend on each other and could all be executed simultaneously. Likewise, the individual arc weighting experiments do not depend on each other and could be executed at the same time. Finally, the matrix multiplication of the random walks portion of the algorithm can also take advantage of concurrency.

Chapter 5

Experimental Methods

The purpose of this chapter is to establish experimental methods that are common to the ensuing experimental chapters. The methods introduced here include tools used to evaluate the accuracy and run time of the examined factor screen methods, described in Section 5.1, a description of the experimental equipment in Section 5.2, and detailed information about the experimental simulation models in Section 5.3. The specific questions to be investigated experimentally are presented in detail by the chapters in which they are addressed and are organized as follows.

Chapter 6 presents the results of experiments relating to the adjustable parameters of the WRW method. The trade-off between run time and accuracy is explored in detail, and recommendations on WRW method setup are established.

Chapter 7 explores a comparison of the WRW method to other factor screen techniques. Specifically, the WRW method is compared to factor screen applications of full and fractional 2^k factorial experiments, random balance experiments, shortest path for importance identification, and a weightless version of the WRW algorithm.

Since Chapter 6 is not vital to understanding the procedures of Chapter 7, readers may skip Chapter 6 if they are primarily interested in the performance of the WRW method compared to other factor screen techniques, and less interested in the detailed inner-workings of the WRW method itself.

5.1 Evaluating Factor Screen Methods

5.1.1 Accuracy Metric

If the relation between estimated importance and actual importance were always linear, as was the case in Figure 4.18, a factor screen heuristic could be judged by some measure of linear fit, such as the coefficient of correlation. Unfortunately, not only will the relation between estimated and actual importance not always be linear, there is no way to determine what the relation between these two quantities will be. Even worse, being a heuristic method, any underlying relation will likely be very noisy and imprecise. Because of this, the factor screening methods will be evaluated based on how well they can order the inputs by their actual importance. The rationale for this evaluation methodology is described in detail by the scope of work in Section 4.1.

What is needed is some means of measuring how accurately a set of values are sorted in descending order. If the sorted list of inputs output by the factor screen heuristics are perfectly ordered, they should be assigned an accuracy of 1.00 and if the order is completely wrong, an accuracy of 0.00. Similarly, placing an input that is very important out of order should be penalized more than incorrectly placing a comparatively unimportant input.

This is accomplished through the formulation of an *Accuracy Score* for measuring the correctness of a method to place a set of actual factor importance into their correct order. For each input to a simulation model there is a pair of importance $I = (a, e)$ such that a is the actual importance for an input as measured by running the simulation, and e is the estimated importance of that input returned by a factor screening method. Let $\mathbf{I}$ be the set of all (a, e) in a model; $|\mathbf{I}| = k$. There is then an set, $\mathbf{\Theta} = \{(a_1, e_1), (a_2, e_2), \ldots, (a_i, e_i), \ldots, (a_k, e_k)\}$ ordered by $e_1 \geq e_2 \geq \ldots \geq e_k$ where $\mathbf{I} - \mathbf{\Theta} = \emptyset$ that describes the output from a factor screen method. If $e_i = e_{i+1}$ the tie in order is broken by the rule $a_i < a_{i+1}$. In general, this tie breaking rule for $\mathbf{\Theta}$ favors lower accuracy scores in case of a tie and assures that cases where $e_1 = e_2 = \ldots = e_i = \ldots = e_k$ will recieve an *Accuracy Score* of zero. The ordered set $\mathbf{\Theta}$ fundamentally describes what a factor screen method thinks is the correct descending order of actual importance for all inputs to a model.

In addition to the estimated order, $\mathbf{\Theta}$, let $\mathbf{\Theta}^*$ be the set $\mathbf{I}$ ordered such that $a_1 \geq a_2 \geq \ldots \geq a_k$ and $\mathbf{\Theta}^0$ to be the set $\mathbf{I}$ ordered such that $a_1 \leq a_2 \leq \ldots \leq a_k$.

If $\Theta(i)$, $\Theta^*(i)$, and $\Theta^0(i)$ return element a_i of $\mathbf{\Theta}$, $\mathbf{\Theta}^*$, and $\mathbf{\Theta}^0$ respectively then $AccuracyScore$ is given by

$$AccuracyScore = \frac{\sum_{i=0}^{k}\sum_{j=0}^{i}\left(\Theta(j) - \Theta^0(j)\right)}{\sum_{i=0}^{k}\sum_{j=0}^{i}\left(\Theta^*(j) - \Theta^0(j)\right)}. \tag{5.1}$$

For example, take a list of 100 random numbers normalized to one that represent actual importance and pair each of them with a different random number representing estimated importance. When sorted by descending *estimated importance* this list is $\mathbf{\Theta}$, or "Random" due to the means used to generate this list. When sorted by descending *actual importance* this list is $\mathbf{\Theta}^*$ and when sorted by ascending *actual importance*, $\mathbf{\Theta}^0$. Figure 5.1 shows a plot of the cumulative values of these three lists for an $\mathbf{I}$, generated randomly as just described.

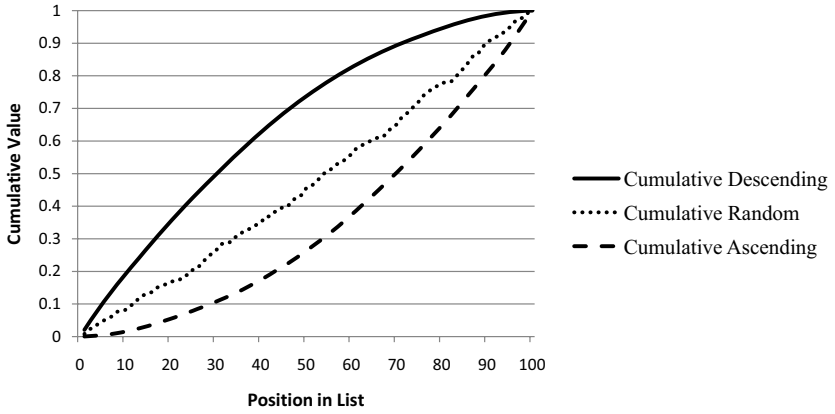

Figure 5.1: Cumulative plots for example list demonstrating accuracy metric.

The best possible ordering is shown by the cumulative descending line, and the worst possible ordering is shown by the cumulative ascending line. The area between the cumulative random and cumulative ascending curves is the numerator, and area between the cumulative descending and cumulative ascending curves is the denominator of Equation (5.1). The ratio of these two areas is the described $AccuracyScore$ and should be interpreted as a measure of

accuracy for the random ordering of this example. In this case, the accuracy score was 0.48. In general, for a randomly generated $\mathbf{I}$, an accuracy score of 0.50 would be expected. The more correctly Θ is ordered in this example, the closer Θ will be to Θ^*, the the closer "Cumulative Random" line would be to the "Cumulative Descending" line, and the closer the ratio described by Equation (5.1) would be to 1.00.

Any inputs that must have a zero main effect due to no reachability are excluded from computation of the accuracy score. Because these inputs do not reach the output under study, the WRW method will rank them last- which is where they belong, since they are guaranteed to have no main effect.

5.1.2 Run Time Metric

It is necessary to compare the run time for the developed factor screening method against alternative methods, including other heuristics and statistical based factor screen experiments. Run time was measured by the internal clock of the test computer, using markers in the Java implementation of the various experimental algorithms to determine when an algorithm sub-step completed. Tasks common to all methods, such as model loading and graph elicitation, were not included in the run time.

Graph elicitation is not counted in method run time because all operations of this step, such as searching for parameters referenced by actors and splitting nodes with multiple outputs into multiple nodes as described in Section 4.3.2.2, could be avoided if supported natively by the data flow programming environment. Data flow programs are naturally represented as graphs; therefore this work considers graph elicitation as a model construction step, not an analysis step.

5.2 Experimental Setup

All experiments were run one at a time on a test computer, using the software implementations described in Section 4.4. To provide a reference point, important specifications relating to test computer's speed and capability are given in Table 5.1.

Table 5.1: Test computer specifications.

Operating System	Microsoft Windows XP Professional with Service Pack 3
CPU	Intel Core2 6700 @ 2.66 GHz
RAM	3.50 GB

Furthermore, all invocations of the Java Virtual Machine were launched with the flag "-Xmx1600M" to increase the maximum Java heap size to the largest value possible, providing the experimental applications with the most RAM memory possible.

5.3 Experimental Models

As mentioned in the scope of work, a variety of models are required to demonstrate the power of the developed heuristic factor screen method. To meet this requirement, a variety of experimental models were constructed.

The experimental models can be split by their size into two categories, depending on whether or not it is practical to experimentally determine the main effects in the model without the potential for aliasing with interaction effects.

1. Models in which it is possible to determine the actual main effects of that model's inputs on its outputs.

2. Models in which it is not possible to determine the actual main effects of that model's inputs on its outputs.

To be included in the first group, a model must either have few enough inputs that a full factorial experiment can be run on it in a feasible amount of time, or be designed such that certain assumptions can be made about interaction effects in that model. For instance, if a model only contains nodes that implement a linear function it is known that model will not contain interaction effects and the actual main effects in the model can be determined through a resolution III factorial experiment.

A summary of the different experimental models is given by Table 5.2. The "Interaction" Effects column denotes if a given model contains interaction effects, which will complicate

experimentation on that model. If a model contains cycles, as shown by the "Cycles" column then time, or the number of iterations a model is run for, may be important to that model's eventual output. If a model contains "Random" elements, there may be variability in its outputs, and multiple replicates will be needed for each experimental configuration. Finally, if the "Ground Truth" is known, it is possible to find the actual importance through experimentation on the model; these models belong to group 1 above. Details of the experimental models, including their structure, design, and high and low values for their input factors are given in the sections below.

Table 5.2: Summary of important categorical properties of the experimental models.

Model Name	**Interaction**	**Cycles**	**Random**	**Ground Truth**
Tree	No	No	No	Yes
Big Tree	No	No	No	Yes
Digraph	Yes	Yes	No	Yes
Serial Queue	Yes	Yes	Yes	Yes
Queue Network	Yes	Yes	Yes	No
Predator-Prey System	Yes	Yes	No	Yes

Additional information on important model properties is given by Table 5.3. The "# Inputs" column lists the number of input factors to each model. "Iterations" refers to the number of iterations a given model will be run for in each replication. Finally, "# Nodes" and "# Edges" denotes the number of nodes and edges that exist in the graph structure elicited from a given model using the process described by Section 4.1.4.

Table 5.3: Summary of important properties of the experimental models.

Model Name	**# Inputs**	**Iterations**	**# Nodes**	**# Edges**
Tree	9	1	22	22
Big Tree	6561	1	16404	16404
Digraph	7	5	14	16
Serial Queue	7	5000	43	53
Queue Network	18	5000	127	163
Predator-Prey System	19	20000	70	92

5.3.1 Tree and Big Tree Models

The "Tree" model was shown in Figure 4.13 and used as an example in Section 4.3.2.5 to demonstrate the mechanics of the Weighted Random Walks method. This model was shown as a linear signal flow graph, as described in Section 3.3.1, and implemented as a synchronous data flow program. Each of the expression nodes in the "Tree" model (*Y1*, *Y2*, *Y3*, and *Z1*) takes three input values, multiplies each input by the weight of its respective input arc, then returns the sum of the resulting values.

The "Tree" and "Big Tree" models were generated in the following way. Starting with a root linear-expression actor, three additional expression actors were created. These three actors were connected to the root actor and randomly assigned a coefficient between 0 and 2 in the root actor's expression. This expansion process continued recursively until the desired number of actors was created. Finally, three parameters (inputs) were created for each of the top layer's expression actors and these parameters given a randomly assigned coefficient between 0 and 2 in its referencing actor's expression. Only one output, the root node *Z0*, will be analyzed in each of these models.

In both of these models, all inputs to this model have high values of 2 and low values of 1, which will be used during analysis and experimentation on these models.

5.3.2 Digraph Model

The "Digraph" model was adapted from the "Tree" model to include cycles and non-linear terms. As with the "Tree" model described in Section 5.3.1, each intermediate node in the "Digraph" model implements a function of its inputs consisting of taking an input, performing some action on that input as a function of the arc it was received on, then taking the sum of the results. The following modifications were made to the "Tree" model to obtain the "Digraph" model. Input nodes *X8* and *X9* were removed to lower the total number of required replicates for a full factorial experiment to be 128 instead of 512. Next, an arc was added from *Y2* to *Y1* and from *Z1* to *Y2*. The *Z1* to *Y2* arc includes a delay with an initial value of zero, to satisfy the requirements of a well formed data flow program. Finally, the intermediate input values on some arcs were modified in ways beyond a simple coefficient. The *Y2* to *Y1* arc was changed to

denote squaring the output of *Y2* before multiplying it by 0.60 in the expression of the *Y1* actor. The function evaluated by the *Y1* actor will now be: $1.88X1 + 1.24X2 + 1.38X4 + 0.60Y2^2$. Similarly, a $\log(x)$ was added to the *Y1* to *Z1* arc.

The full structure of the "Digraph" example model is given by Figure 5.2. Only one output, *Z1*, will be analyzed in this model.

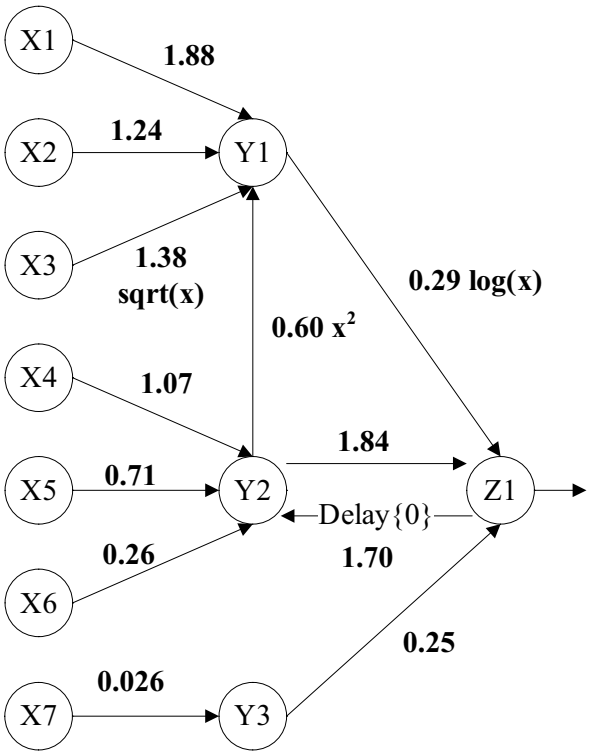

Figure 5.2: Digraph experimental model.

As with the "Tree" and "Big Tree" models, all global inputs to this model will have low values of 1 and high values of 2.

5.3.3 Serial Queue and Queue Network Models

The "Serial Queue" and "Queue Network" models are both based heavily on the same building block representing a single queue. Each individual queue has one state variable, representing the number of customers currently in that queue, and each queue is linked to the rest of a model through two variables; arrivals to the queue and departures from the queue. Additionally, each queue has two settable parameters which are the number of servers and the number of customers

each server can service in a given iteration.

Time in these models, as with all synchronous data flow models, progresses iteratively in discrete portions called "ticks" or "iterations". Each iteration represents a given time period. Although very important to the model's author, the exact amount of time represented by each iteration is not important to analysis of the model (given that a desired time-span for study is known). The factor screening performed here takes into account effects over a time span given in iterations, it does not care what actual time those iterations supposedly represent. These queuing experimental models will be analyzed after 5000 iterations.

The behavior of each queue is governed completely by two difference equations. The first:

$$Departures(t+1) = \tag{5.2}$$
$$min(NumInQueue(t) + Arrivals(t), Pois(Servers \times ServicesPerServer))$$

Computes the number of departures from the queue, and the second:

$$NumInQueue(t+1) = \tag{5.3}$$
$$NumInQueue(t) + Arrivals(t) - Departures(t)$$

Updates the state variable $NumInQueue$ tracking the queue's current size. Note that $Pois(\lambda)$ returns a random sample from a Poisson distribution with a rate of λ and $min(x, y)$ returns the minimum of x and y.

The "Serial Queue" model consists of three queues. Customers arriving to this system must pass through each queue in sequential order before departing the system. Instant travel between queues is assumed; the departures from the first queue in a given time period are the arrivals to the second queue in that same time period. Customers arrive to the first queue following a Poisson distribution with a mean of the input $ArrivalRate$. They then pass through all three queues and depart the system. Two outputs will be examined in this system. They are

1. The total cumulative number of customers to exit the system.

2. The average number of customers waiting in *Queue 3*.

The full system represented by the "Serial Queue" model is depicted in Figure 5.3. The abbreviation *Q#* will be used to refer to the individual queues; *Q1* will be *Queue 1*, etc.

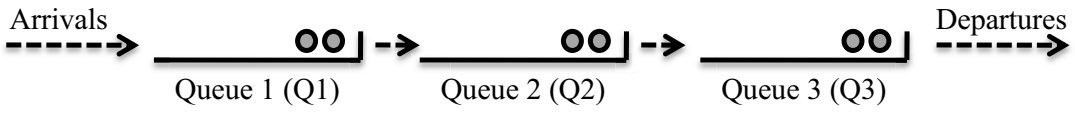

Figure 5.3: Queuing system represented by Serial Queue model.

Figure 5.3 shows the general structure of the serial queue model, not the detailed model as implemented. When implemented as a data flow program, the data flow graph of the Serial Queue model is as shown by 5.4.

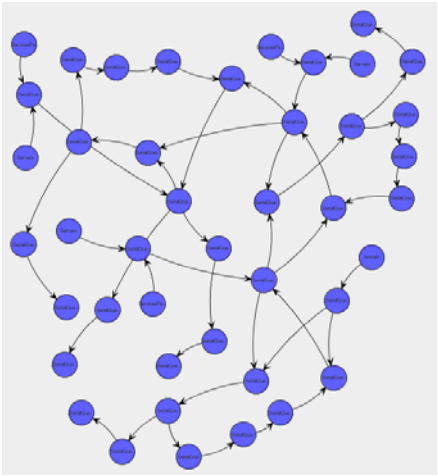

Figure 5.4: Data flow graph of the Serial Queue model.

The "Queuing Network" model is a similar queuing system to that of the "Serial Queue" model, but made up of more queues. An additional complexity over the "Serial Queue" model is that in the "Queuing Network" model, customers are occasionally presented with a choice of which queue to enter next. When given a choice, customers will randomly choose between one of the alternatives, with all alternatives being equally likely. Three outputs will be examined in the "Queuing Network" model. They are:

1. Cumulative system exits through *Out 1*.

2. Cumulative system exits through *Out 2*.

3. The average number of customers waiting in *Queue 6*.

The "Queuing Network" model has two system arrival points, *Arrival 1* and *Arrival 2*. Customers arrive to each following a Poisson distribution with a mean of *Arrival Rate 1* to the first arrival point and *Arrival Rate 2* to the second. The structure of the system modeled by the "Queuing Network" model is shown by Figure 5.5.

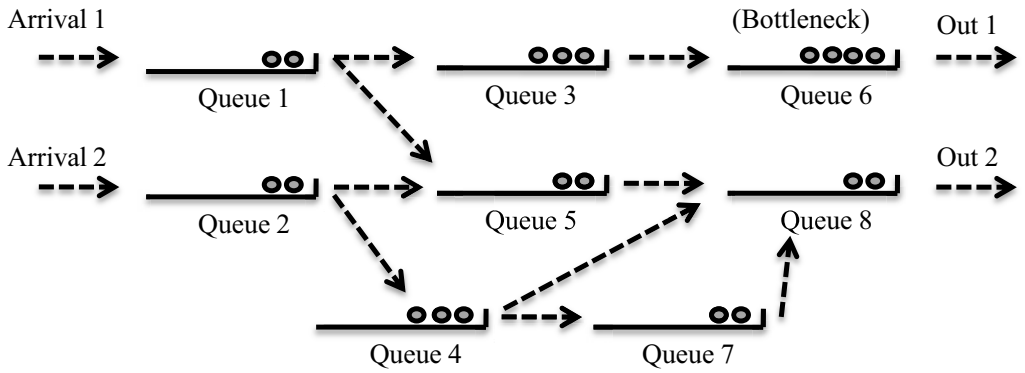

Figure 5.5: Queuing system represented by Queue Network model.

As with the Serial Queue model, Figure 5.5 only shows the general structure of the model, not the detailed model as implemented. Figure 5.6 shows the full data flow graph from the data flow implementation of the Queueing Network model..

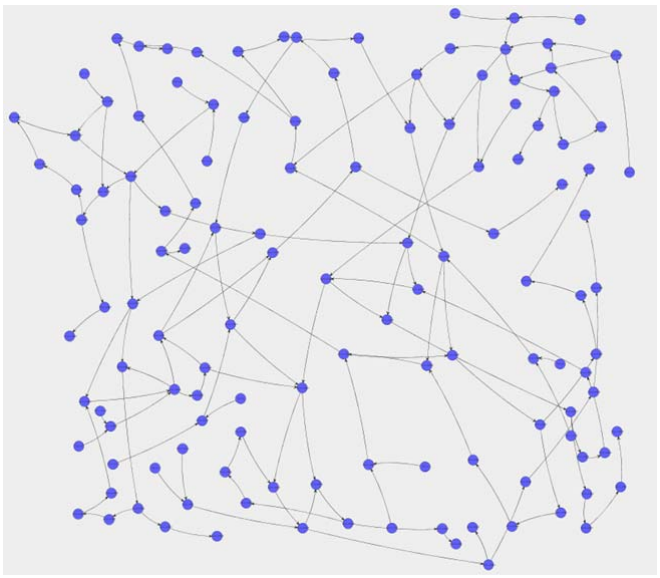

Figure 5.6: Data flow graph of the Queueing Network model.

In both the "Serial Queue" and "Queuing Network" models, the number of servers and their service rates at the individual queues were assigned randomly. The service capacity of all but two queues is set high enough that the system will operate in steady state; the service rate is greater than the arrival rate. The two exceptions are *Queue 3* of the "Serial Queue" model and *Queue 6* of the "Queuing Network" model, both of which have a low service capacity making them bottlenecks in their systems with their default values. The exact high and low factor values chosen for these models are given in Table 5.4 for the "Serial Queue" model and Table 5.5 for the "Queuing Network" model. Fractional servers are allowed, since these inputs are used only to determine the mean of a Poisson distribution, samples from which are always integers.

Table 5.4: High and low factor values for the Serial Queue model.

Input Name	**Low Value**	**High Value**
Arrivals	10	20
Q1 Servers	5	10
Q1 Services / Server	4	8
Q2 Servers	5	10
Q2 Services / Server	5	10
Q3 Servers	5	10
Q3 Services / Server	3	6

5.3.4 Predator and Prey Model

The final example model is one of a multiple-species Lotka-Volterra system, based heavily on the examples presented by (Harrison, 1979) and referred to here as the "Pred / Prey" model. The model used in this work contains 4 species; two predator species feeding on two prey species. The population of each prey species grows in each time interval by the given species birth rate and prey do not die from natural causes. Likewise, each predator specie's population decreases in each time interval by that species death rate and predator species do not reproduce by natural causes. Predators interact with prey by eating them. When a predator eats a prey, the population of prey decreases and the population of predators increases by some scale factor called efficiency in this work.

In this example the two predator species are Fox and Tigers and they feed on the two prey

Table 5.5: High and low factor values for the Queuing Network model.

Input Name	Low Value	High Value
Arrival Rate 1	10	15
Arrival Rate 2	7	10.5
Q1 Servers	3	4.5
Q1 Services / Server	1	1.5
Q2 Servers	5	7.5
Q2 Services / Server	6	9
Q3 Servers	7	10.5
Q3 Services / Server	3	4.5
Q4 Servers	9	13.5
Q4 Services / Server	3	4.5
Q5 Servers	9	13.5
Q5 Services / Server	3	4.5
Q6 Servers	2	3
Q6 Services / Server	10	15
Q7 Servers	7	10.5
Q7 Services / Server	6	9
Q8 Servers	4	6
Q8 Services / Server	4	6

species, Mice and Deer. Tigers prefer to eat Deer and Fox prefer to eat Mice, but both predator species will eat either prey- just at varying levels of success. The full data flow graph of the data flow implementation of this model is shown in Figure 5.7.

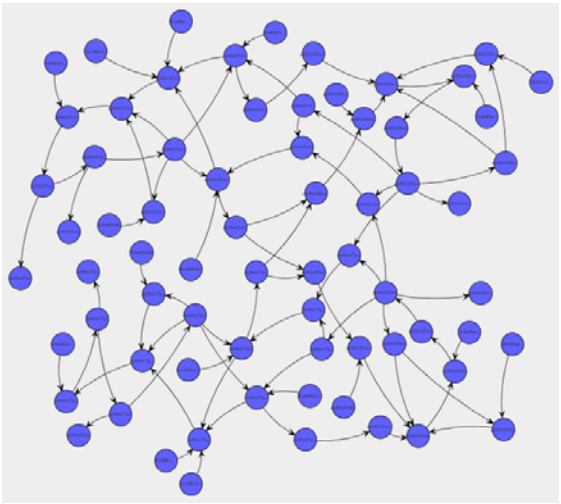

Figure 5.7: Data flow graph of the Predator and Prey model.

The high and low factor values that will be used on this model are given in Table 5.6.

The $PrefKillRate$ refers to the rate at which a predator species consumes its preferred prey while $GenKillRate$ refers to the rate at which a predator species consumes its non-preferred prey. Likewise, "Pref and Gen Efficiency" refers to the scale factor that determines how many predator species are created for each prey consumed.

Table 5.6: High and low factor values for the Predator and Prey model.

Input Name	**Low Value**	**High Value**
Deer BirthRate	0.00500	0.00750
Mouse BirthRate	0.07000	0.10500
Tiger DeathRate	0.01000	0.01500
Tiger PrefKillRate	0.02000	0.03000
Tiger GenKillRate	0.00100	0.00150
Tiger PrefEfficiency	0.25000	0.37500
Tiger GenEfficiency	0.01000	0.01500
Fox DeathRate	0.02000	0.03000
Fox PrefKillRate	0.00800	0.01200
Fox GenKillRate	0.00010	0.00015
Fox PrefEfficiency	0.10000	0.15000
Fox GenEfficiency	0.50000	0.75000

Chapter 6

Investigation of Method Behavior

During the development of the Weighted Random Walks (WRW) method for factor screening in Section 4.3.2, a number of questions were proposed about best practices for setting the various adjustable parameters of that algorithm. Although many of these questions were addressed theoretically during the method's initial description, empirical testing is required to more fully understand and demonstrate their connotation. Note that because this chapter is not vital to understanding the procedures of the second experimental chapter, Chapter 7, readers may skip this chapter if they are primarily interested in the performance of the WRW method compared to other factor screen techniques.

This section experimentally explores the settable parameters of the presented heuristic method, and their associated performance trade-offs. Because these questions are application specific, a variety of possible different settings are tested in a variety of applications. The specific parameters explored are:

1. How many sample input configurations should be used to determine the distribution of intermediate values that will be needed for determining high and low factor values for the arc weighting experiments?

2. How large a proportion of iterations computed for a given input configuration need to be recorded and does recording a smaller proportion of these iterations significantly reduce the heuristic's run time?

3. How should the high and low factor values to be used for the arc weighting experiments be chosen from the sampled intermediate values?

4. What type of two-level experiment should be used for determining arc weights, and thereby one-step transition probabilities?

Although highly application dependent, these parameters do have an important impact on the performance of the WRW method and should be carefully considered when performing analysis. This chapter does not directly address either of this thesis' hypothesis. Through an exploration of parameters to the Weighted Random Walks method, what this chapter does address is the first objective of this thesis as proposed in the problem statement, which is to apply tools from the domain of graph theory to the identification of important factors in a synchronous data flow program.

6.1 Experimental Procedure

Experiments were designed and carried out for each model where actual importance is known from a full factorial experiment. The first set of experiments looks at the trade-off between run time and accuracy resulting from changes to the sample size of input configurations is studied by systematically varying this parameter and observing the resulting accuracy and run times. The effect of varying the percentile of observed intermediate values chosen as high and low factors is also varied between the minimum and maximum values observed (effectively the 0^{th} and 100^{th} percentiles) and the 1^{st} and 99^{th} percentiles.

It is known with certainty that the *Range Percentiles* factor can have no effect if only small samples are taken. The 99^{th} percentile of only 10 samples is usually rounded to be the same as the 100^{th} percentile. As such, the (1, 99) factor level for the *Range Percentiles* parameter will not be examined for experimental combinations where there are fewer than 100 samples for each input arc; in cases where the 99^{th} percentile will round to the maximum, the 99^{th} case will be ignored.

The next set of experiments explore the type of experiment used to assign arc weights for an actor's input arcs given the factor values are known for those arcs. While previously

described experiments focus on parameters affecting the high and low factor values for an actor's input arcs, this set of experiments will focus on how to best use those factor values to assign meaningful arc weights.

The final set of experiments takes the most promising combination of parameters realized through earlier experimentation and explores the trade-off of recording fewer simulated iterations on accuracy and time. Since this trade-off is only expected to benefit models requiring many iterations, only the "Serial Queue" model will be examined in these experiments.

During the sampling of intermediate values phase of the WRW method, input configurations are generated through a random balance experiment. This introduces uncertainty into the method's results. To obtain a good estimate of average performance, as well as the variability of performance, five replicates will be run for each experimental combination of the algorithm's parameters. The resulting mean of the accuracy scores and run times will be examined. Detailed information about run time uncertainty will be expressed through standard statistical hypothesis tests. Unfortunately, it is not possible to assume that accuracy scores as presented in Section 5.1 are normally distributed, or even continuous, in all cases. Because the distribution of these scores is unknown, it is not be possible to use parametric statistical tests.

A summary of all factors and their levels that will be examined are given in Table 6.1.

Table 6.1: Parameters and factor levels for experiment to investigate WRW performances.

Parameter	**Factor Levels**
Input Configuration Samples	2, 4, 6, 8, 10, 20, and 100
Range Percentiles	$\{0, 100\}$ and $\{1, 99\}$
Weighting Experiment Design	Full Factorial, Resolution III
Proportion Iterations Recorded	100%, 10%

6.2 Results

This section presents a selection of the results from the above experiments. For the remaining results, including uncertainties associated with presented averages, see Appendix A.

The observed run time of the weighted reverse random walks heuristic method by number

of random input configuration samples is given for the relevant experimental models by Table 6.2. Intermediate values were recorded for all iterations, high and low factor values used for the arc weighting experiments were chosen as the maximum and minimum observed values, and full factorial experiments were used to assign arc weights.

Table 6.2: Weighted Random Walks method run time (ms) for each tested model by number of randomly sampled input configurations.

	Number of Sampled Input Configurations						
Model Name	2	4	6	8	10	20	100
Tree	287	791	310	313	331	375	978
Big Tree	78900	82700	88100	95900	104000	136000	401000
Digraph	447	493	519	572	607	831	2254
Serial Queue	766	1153	1460	1810	2030	3800	15100

To study accuracy requires examination on the output level, instead of study at the model level used above for run time. The Tree, Big Tree, and Digraph models only have one output, so accuracies can be expressed on one line for these models. The Serial Queue experimental model, however, has two outputs of interest and will require two rows; one row labeled Serial Queue[1] for this model's Cumulative System Exits output and one row labeled Serial Queue[2] for the model's Average Waiting in Q3 output.

The accuracy scores for the Weighted Random Walks method by number of random input configuration samples and the percentile of sampled data used as high and low factor values in the weighting experiments is given for the relevant experimental models by Table 6.3 for the Range Percentiles factor setting of (0, 100). As with the runs above, all of these experiments were based on the use of full factorial experiments for arc weighting and intermediate values were recorded for all iterations. Accuracy is assigned following the method described in Section 5.1, and as ranked against ground truth as found through full factorial, or equivalently accurate, experiments on the experimental models. An accuracy score will be 1.00 if the list output by the Weighted Random Walks method perfectly matches the ground truth order, and 0.00 if the generated list is completely opposite the order returned by the ground truth experiment.

Table 6.3: Weighted Random Walks method accuracy for each tested model by number of randomly sampled input configurations for the Range Percentiles factor setting, (0, 100).

	Number of Sampled Input Configurations						
Model Name	2	4	6	8	10	20	100
Tree	0.98	0.94	1.00	1.00	1.00	1.00	1.00
Big Tree	0.85	0.96	0.97	0.98	0.98	0.99	1.00
Digraph	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Serial Queue[1]	0.75	0.74	0.69	0.69	0.57	0.51	0.46
Serial Queue[2]	0.86	0.67	0.63	0.73	0.63	0.81	0.63

Table 6.4 shows the result of varying the Input Configuration Samples factor value for the Range Percentiles factor setting of (1, 99). Accuracy is again presented as the average of five samples. In the following table, a * in a cell is used to indicate a case that will be equivalent to the respective value displayed in Table 6.3 due to rounding, or where the 1^{th} percentile of the samples will round to the minimum value and the 99^{th} percentile will round to the maximum.

Table 6.4: Weighted Random Walks method accuracy for each tested model by number of randomly sampled input configurations for the Range Percentiles factor setting, (1, 99).

	Number of Sampled Input configurations						
Model Name	2	4	6	8	10	20	100
Tree	*	*	*	*	*	*	1.00
Big Tree	*	*	*	*	*	*	1.00
Digraph	*	*	*	*	*	*	1.00
Serial Queue[1]	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Serial Queue[2]	0.87	1.00	1.00	1.00	0.95	1.00	1.00

Next, to explore the relationship between weighting experiment design, run time, and resulting accuracy, Big Tree model's run time was broken up by method stage. As described in Section 4.3.2.4, the Weighted Random Walks method has four main stages; pre-processing, intermediate range sampling, arc weighting experiments, and node importance sub-algorithm. The pre-processing stage is not being counted in run time, as described by the run time metric of Section 5.1.2. The run time for the other three stages were measured and are presented by weighting experiment type in Table 6.5. Note that this table, as the tables above, is based on an

average from 5 samples.

Table 6.5: Breakdown of runtime (in ms) by method stage for *Big Tree* model. The only statistically significant difference, at $\alpha = 0.05$, was observed for the Weighting Experiments stage.

	Weighting Experiment Type		
Stage	**Full Factorial**	**Resolution III**	**Difference**
Range Marking (ms)	20500	20200	348
Weighting Experiments (ms)	67400	61700	5670
Importance Algorithm (ms)	250	244	6
Accuracy Score	0.97	0.96	0.01

An overview of the impact of weighting experiment type is given for all models tested in this section by Table 6.6. All trials were run with an input configuration sample size of 6, and the range percentiles factor was set to (0, 100) for the Tree, Big Tree, and Digraph models, and (1, 99) for the Serial Queue model.

Table 6.6: Method run time and accuracy by weighting experiment type; Full Factorial (FF) or Resolution III (III). Serial Queue[1] row is for Cumulative System Exits output, Serial Queue[2] for Average in Q3 output.

Model Name	**Run Time FF (ms)**	**Run Time III (ms)**	**Accuracy FF**	**Accuracy III**
Tree	310	281	1.00	1.00
Big Tree	88100	82100	0.97	0.96
Digraph	519	472	1.00	1.00
Serial Queue[1]	1460	1430	1.00	0.45
Serial Queue[2]	1460	1430	1.00	0.53

Finally, 90% of the values passed during the input configuration sampling process were ignored at random. The results from this process, alongside the results if none of these values are ignored, is presented in Table 6.7.

6.3 Discussion

Increasing the number of input configurations sampled resulted in an increase to mean method run time in all cases shown by Tables 6.2. This was expected, as larger input configuration

Table 6.7: Examination of effect on accuracy from proportion iterations recorded for Serial Queue model.

Percent of total data sampled:		100%	10%
Cumulative System Exits	Accuracy	1.00	1.00
	Run Time (ms)	1960	1790
Average in Q3	Accuracy	1.00	0.93
	Run Time (ms)	1960	1790

samples require running more simulation replications compared to smaller input configuration samples. Similarly, increasing the number of input configurations sampled by the presented method also usually resulted in an increase to accuracy, as shown by Tables 6.3 through 6.4.

The rate at which run time and accuracy increased was found to depend greatly on the model under study. The increase to method run time seen from an increased number of sampled input configurations seems strongly related to the size of the model under study. For instance, increasing the number of sampled input configurations from 2 to 4 resulted in a mean increase in run time of 6810 ms for the Big Tree model, but only 4 ms for the structurally similar, but significantly smaller, Tree model.

The relationship between number of sampled input configurations and the resulting accuracy score is less clear. In all but one case, increasing from 2 sampled input configurations to 100 resulted in an equal, or better, accuracy score. The only observed exceptions to this trend were seen in the outputs from the only model tested that contains random elements. Specifically, for the Serial Queue model's Cumulative Exits output, shown in Table6.3, an increase from 2 sampled input configurations to 100 when the Range Percentiles factor was set to (0^{th}, 100^{th}), resulted in a decrease of mean accuracy from 0.75 to 0.46 for the Cumulative System Exits output a decrease of accuracy from 0.86 to 0.63 for the Average Waiting in Q3 output. Since this unfavorable behavior was not seen in any of the other models tested for this section, or in the Serial Queue model when the Range Percentiles factor was set to the (1^{th}, 99^{th}) level, it is hypothesized that the poor performance in the (0^{th}, 100^{th}) case is due to actors that produce random variables. As the number of samples taken from a continuous random distribution increases, so does the expected range of those samples.

In the context of the Weighted Random Walks method, this increase in sampled range that results from an increased number of sampled input configurations is undesirable, as the goal of the intermediate range sampling stage of the method is to approximate the range of *expected values*, not the range of *possible values*. Luckily, it appears that choosing low and high factors for arc weighting experiments as the (1^{th}, 99^{th}) percentiles instead of the (0^{th}, 100^{th}) mitigates this unfavorable behavior. As seen for the Serial Queue model in Tables 6.3 and 6.4, this change in percentiles resulted in a vast improvement to the resulting accuracy scores.

Changing the 2-level factorial experimental design used for the arc weighting experiments from a full factorial to fractional factorial of resolution III lowered the run time of the Weighted Random Walks method for all studied models, as shown by Table 6.6. As expected, using the design that requires fewer replicates resulted in quicker overall run times. More specifically, using the less time consuming Resolution III design when applying the Weighted Random Walks method saves a considerable amount of time over the Full Factorial design in that method's required Weighting Experiments stage, as shown in detail for the Big Tree model by Table 6.5. The exact relationship between the Weighting Experiment Design factor and total method run time appears to also be very application specific.

Furthermore, changing the Weighting Experiment Design factor had no appreciable effect on the Weighted Random Walks method's accuracy for the Tree, Big Tree, and Digraph models. This factor did, however, have a considerable effect on accuracy for both studied outputs of the Serial Queue model. Specifically, while using a full factorial weighting experiment both studied outputs of the Serial Queue model, Cumulative System Exits and Average in Q3, had accuracy scores of 1.00 under the factor setting of full factorial weighting experiment. When the weighting experiment design was changed to a resolution III design, the accuracy scores for Cumulative System Exits and Average in Q3 dropped to 0.45 and 0.53 respectively, as shown in Table 6.6. One possible explanation for this behavior is that the Tree, Big Tree, and Digraph models only contain nodes with no interaction effects, and therefor have main effects that can be accurately characterized by a resolution III design. Many of the Serial Queue model's nodes, on the other hand, do contain significant interaction effects. In the presence of these interaction effects, a full factorial experiment is required to prevent aliasing of main

effects with any potentially important interaction effects.

Finally, it appears possible to decrease the run time of the Weighted Random Walks method by randomly sampling only 10% of iterations during the intermediate range sampling phase. When this practice was applied to the Serial Queue model, overall run time was decreased by an average of 8.2% with no degradation in accuracy for the Cumulative System Exits output, and only a 0.07 decrease in accuracy for the Average in Q3 output, shown by Table 6.7.

All of the settable parameters to the Weighted Random Walks method studied in this chapter were found to be highly application dependent. While no firm conclusions can be made from the presented data, four general guidelines are suggested here.

1. Use fewer than 20 input configuration samples.

For all experimental models studied in this section, using greater than 20 input configuration samples in the intermediate range sampling stage of the Weighted Random Walks method yielded at best small marginal benefits to accuracy. For example, increasing from 20 to 100 input configuration samples for the Big Tree model resulted in an increase to accuracy of 0.01, from 0.99 to 1.00, but it took 401 seconds to sample 100 input configurations instead of 136 seconds for only sampling 20. While this trade off may be worthwhile in certain instances, given the broad goal of the Weighted Random Walks method, such an unfavorable trade off should be avoided. From the data presented in this chapter, it seems reasonable to select 6 as the number of input configurations to sample.

2. For large models, only sample 10% of all iterations during the intermediate range sampling stage.

Only sampling 10% of iterations during the intermediate range sampling stage of the Weighted Random Walks method was shown in Table 6.7 to produce a significant reduction in run time at little harm to the resulting accuracy of results for the Serial Queue model. Sampling 10% instead of 100% of iterations will also consume less computer memory, permitting the application of the Weighted Random Walks method more efficiently to larger models.

3. Choose low and high factors for use by the weighting experiments as the minimum and maximum observed values respectively for deterministic models and as the 1^{th} and 99^{th} percentiles for models with random elements.

It is shown by Table 6.3 that the minimum and maximum scheme works well for deterministic models although Tables 6.4 seems to suggest some upper and lower percentiles work best if the model under study contains random elements. As with sampling a smaller proportion of iterations described above, using the maximum and minimum observed values instead, of a middle percentage, for use in the weighing experiments will result in less memory consumption as a fewer number of values needs to be recorded; specifically, only the current minimum and maximum values.

4. If it is known that no nodes in a model implement interaction effects, use a resolution III fractional factorial design for the arc weighting experiments. If in doubt about the model's internal workings, use a two-level full factorial experiment for weighting.

When possible, the reduced run time offered by resolution III designs should be exploited. If unsure about the exact behavior of the actors in the model under study, use full factorial designs for the weighting experiments, as failure to do so may be catastrophic to the resulting accuracy. Such a catastrophe was shown by Table 6.6 for the Serial Queue model and Cumulative System Exits output, which went from an accuracy score of 1.00 to 0.45 after switching weighting experiment design from full factorial to resolution III, and for the Average in Q6 output which went from 1.00 to 0.53.

Ultimately, the controllable parameters of the Weighted Random Walks method are application dependent and will need to be set as such. Careful study, or a prior understanding, of the model under investigation will help determine appropriate parameter values. Luckily, the Weighted Random Walks method was shown to be robust against incorrect settings for many of its parameters, such as number of sampled input configurations and proportion of iterations recorded. The parameters to which the method was not shown to be robust, namely the design of weighting experiments, have a default safe value that could be applied if an analyst were unsure of the proper setting.

Chapter 7

Comparison to Alternative Factor Screen Techniques

In order to be useful as a factor screening method, the Weighted Random Walks (WRW) method must provide some benefit over currently available methods. Such a benefit may be to either accuracy of information returned, breadth of information returned, or the speed at which such information can be generated.

This chapter contains an experimental comparison of competing factor screen methods. The specific methods for factor screening that will be tested against each other are:

- Random selection,
- Resolution IV and III 2^k fractional factorial experimental designs,
- Random Balance experimental design,
- Shortest Path,
- Random Walks, and
- Weighted Random Walks.

Comparison of these methods will be used to help answer the following questions:

1. How does the WRW method compare to random chance,

2. How does the WRW method compare to statistical designed experiment factor screening techniques,

3. Does the WRW method provide any benefit over the small Random Balance experiment it requires to populate arc weights, and

4. How does the WRW method perform when compared to the strictly structural factor screen techniques: "Shortest Path" and the unweighted version of the WRW method, "Random Walks"?

The first question is important, as any method that does not perform more accurately than random is effectively useless for factor screening.

The second question is the main focus of this chapter. The current standard for factor screening in simulation is the employment of designed statistical experiments, as reviewed in Section 3.1. The WRW method does not return information as detailed as most of these designed experiments, such as an exact estimate of effect, but this information is not specifically required in the scope of identifying important input factors. To be of benefit, the WRW method must perform more accurately, more quickly, or both, than designed experiments for factor screening. Since the WRW method is not thought likely to perform more accurately, a focus will be placed on a comparison of its speed against statistical factor screening techniques.

The WRW method, as described in Section 4.3.2, requires the execution of a small Random Balance experiment on the model under study. WRW then uses information returned from this experiment to compute arc weights for the data flow graph, then performs a structure based analysis. The third question proposed above addresses the benefits, or lack of, using information returned from the Random Balance experiment to seed a structure based factor screen method over direct statistical analysis of that information. Since the additional computation required for analysis by the WRW method means it will always take longer to run compared to a simple Random Balance designed experiment, for the WRW method to have a competitive advantage over Random Balance, it must produce more accurate results.

Finally, two other methods were proposed in this work as potential factor screening techniques. The first, called here "Shortest Path", was presented in Section 4.2. The Shortest Path

method measures the shortest path length between the output node under study and each of the model's input nodes. An interpretation is then made that the these path lengths are an estimate of importance, with the shortest path length being the most important. The second structural method proposed is called "Random Walks" and, as described by Section 4.3, is simply the unweighted version of the featured Weighted Random Walks method for factor screening.

By completing the second two objectives introduced by the problem statement in Chapter 2, this chapter directly addresses this thesis' two hypotheses, described by the same section. Through a direct comparison to random, this chapter shows that the first proposed hypothesis is correct, or that a measure of relative importance on a data flow graph can be used to help identify important inputs to a simulation model. Furthermore, a direct comparison to other factor screen techniques shows that the second hypothesis, or that using the structure of a data flow simulation model to aid in factor screening produces results more quickly than statistical methods of comparable accuracy, is also correct.

7.1 Experimental Procedure

Various factor screen methods are applied to all experimental models introduced by Section 4.1.1. Ultimately, each applicable method will be applied to each experimental model and the resulting performances compared.

Uniformity in setting individual method parameter's between models is practiced when appropriate. This consistency allows for a more powerful comparison of method performances between models than would be possible of parameters were custom set for best performance. The only "tuning" of methods to a specific model will be based off guidelines previously established.

The WRW method is always run with Weighting Experiment Design parameter set to full factorial, and the Input Configuration Samples parameter set to 6. For methods with random elements, the Range Percentiles parameter is set to (1, 99), and for deterministic models this parameter is set to (0, 100). The Proportion Iterations Recorded parameter is set to 100% for all models run for fewer than 1000 iterations per replication. These parameter settings were

chosen based on the guidelines proposed in Section 6.3, which are based on the experiments of Chapter 6.

When running statistical experiments on models with random elements, five replicates will be run for each experimental configuration, or corner point, unless stated otherwise. For deterministic models, only one replicate per corner point is required.

As with Chapter 6, accuracy is assigned following the method described in Section 5.1. When possible, accuracy is ranked against ground truth as found through 2^k full factorial, or equivalently accurate, experiments on the experimental models. If it is not possible to find ground truth due to impractically long run times, accuracy will be ranked against "Assumed Truth", which will be found through the highest resolution fractional factorial experiment feasible.

In the remainder of this chapter, examined methods are parametrized for convenience. Method parameters are as follows:

- Full Factorial (Number of Replicates per Corner Point)
- Resolution III or IV (Number of Replicates per Corner Point)
- Random Balance (Number of Input Configurations)
- Weighted Random Walks (Input Configuration Samples, (Middle) Range Percentiles, Proportion Iterations Recorded)

For models with multiple outputs, a superscript number is used to indicate which of a model's outputs a label refers to. When required, these references are given by Table 7.1.

All accuracy scores are expressed as the average of five trials. Run times are based off the average from a minimum of five trials, although run times were pooled when a method had to be run more than five times on a given model. Such pooling occurred on models with multiple studied outputs, as each method had to be applied five times for each output to measure accuracy.

Table 7.1: Model label and referenced output for models with multiple outputs under study.

Label	Output
Serial Queue[1]	Cumulative System Exits
Serial Queue[2]	Average Waiting in Q3
Queue Network[1]	Cumulative Exits from Out 1
Queue Network[2]	Average Waiting in Q6
Queue Network[3]	Cumulative Exits from Out 2
Pred / Prey [1]	Tiger Population Density
Pred / Prey [2]	Mice Population Density

7.2 Results

This section contains a summary of the results obtained from applying each selected factor screen method to each experimental model from Section 4.1.1. Full results are given in Appendix B.

The 99.99^{th} percentile accuracy score expected if list order were assigned randomly was determined through simulation. The list of an output's actual importance was randomly shuffled 50 million times, grouped into sets of five, and the resultant 99.99^{th} percentile scores for the five-sample averages were recorded. These percentiles for the experimental models are shown alongside the accuracy returned from the Weighted Random Walks method in Table 7.2.

Table 7.2: 99.99^{th} percentile accuracy scores expected if order is determined randomly, and the actual WRW accuracy scores obtained from application to relevant experimental models.

Model	99.99^{th} Percentile Random Score	WRW Score
Tree	0.81	1.00
Big Tree	0.52	0.97
Digraph	0.87	1.00
Serial Queue [1]	0.91	1.00
Serial Queue [2]	0.87	1.00
Queue Network [1]	0.86	0.95
Queue Network [2]	0.85	0.95
Queue Network [3]	0.87	1.00
Pred / Prey [1]	0.74	0.79
Pred / Prey [2]	0.80	0.84

Next, using a 2^k fractional factorial design of resolution III for factor screening is compared

against the WRW method; with parameters set as described above. The results of this comparison are shown in Table 7.3. "Queue Network*" in the following tables indicates a comparison against Resolution III (1) design, opposed to Resolution III (5) design used for the "Queue Network" row.

Table 7.3: Comparison of Weighted Random Walks method for factor screening versus Resolution III 2^k fractional factorial design for experimental models.

	Accuracy Score		**Run Time (ms)**	
Model	**Resolution III**	**WRW**	**Resolution III**	**WRW**
Tree	*	1.00	131	310
Big Tree	*	0.97	24900000	89700
Digraph	1.00	1.00	938	519
Serial Queue [1]	1.00	1.00	7800	1500
Serial Queue [2]	1.00	1.00	7800	1500
Queue Network [1]	1.00	0.95	73900	12500
Queue Network [2]	1.00	0.95	73900	12500
Queue Network [3]	1.00	1.00	73900	12500
Queue Network [3 *]	1.00	1.00	27300	12500
Pred / Prey [1]	0.98	0.79	81900	34800
Pred / Prey [2]	0.98	0.84	81900	34800

The run times of the comparison shown by Table 7.3 are expressed as the ratio of resolution III run time to WRW run time in Table 7.4. This ratio is called the "Speed Ratio" and is the measure of how many times faster the WRW method is than the resolution III method.

Table 7.4: Comparison of speed advantage of Weighted Random Walks method versus Resolution III 2^k fractional factorial design for experimental models.

Model	**Speed Ratio**
Tree	0.42
Big Tree	280
Digraph	1.8
Serial Queue	3.4
Queue Network	5.9
Queue Network*	2.2
Pred / Prey	2.3

Next, the WRW method's accuracy is compared against a random balance (6) design. The

results of this comparison are shown in Table 7.5.

Table 7.5: Comparison of Weighted Random Walks method for factor screening versus random balance (6) design for factor screening on experimental models.

	Accuracy Score	
Model	**Random Balance**	**Weighted Random Walks**
Tree	0.73	1.00
Big Tree	0.45	0.97
Digraph	0.74	1.00
Serial Queue [1]	0.73	1.00
Serial Queue [2]	0.68	1.00
Queue Network [1]	0.81	0.95
Queue Network [2]	0.68	0.95
Queue Network [3]	0.70	1.00
Pred / Prey [1]	0.75	0.79
Pred / Prey [2]	0.76	0.84

Finally, the WRW method is compared against the two proposed structural screening methods; one based on shortest path lengths and a second which is the WRW method with no weights. A comparison of these method's accuracies is given by Table 7.6.

Table 7.6: Comparison of Weighted Random Walks method for factor screening versus other proposed structural methods for factor screening on experimental models.

	Accuracy Score		
Model	**Shortest Path**	**Random Walks**	**Weighted Random Walks**
Tree	0.00	0.00	1.00
Big Tree	0.00	0.00	0.97
Digraph	0.00	0.77	1.00
Serial Queue [1]	0.16	0.46	1.00
Serial Queue [2]	0.41	0.49	1.00
Queue Network [1]	0.88	0.94	0.95
Queue Network [2]	0.45	0.47	0.95
Queue Network [3]	0.00	0.47	1.00
Pred / Prey [1]	0.40	0.53	0.79
Pred / Prey [2]	0.56	0.59	0.84

7.3 Discussion

The first goal of this chapter is to establish that the WRW method performs better than average. Table 7.2 contains a by-model direct comparison against the observed average WRW performance and expected random performance with respect to accuracy. Run time is not shown, since randomly assigning importance estimates is considered instantaneous.

Since the the distribution of accuracy scores described in Section 5.1.1 is not known, parametric statistics can not be used. Because parametric statistics can not be used, traditional tests of significance are more difficult. Even so, by showing the average WRW score exceeds the 99.99^{th} percentile of expected random performance for each tested model, the contents of Table 7.2 strongly suggests better-than random performance of the WRW method.

Next, the WRW methods performance is compared to that of more traditional factor screening techniques. Table 7.3 contains a detailed comparison between the WRW method and a resolution III 2^k partial factorial experiment. As previously mentioned, no accuracy comparisons are available between the WRW method and the resolution III method for the Tree and Big Tree models, since the lack of interaction effects makes the resolution III results the ground truth in these models.

In many cases, the WRW method performed exactly as accurately as the resolution III experiments at magnitude order ranking. As implied by the accuracy scores, most deviations between the two methods was due to a mis-ordering of relatively unimportant input factors. In no experimental model did the average WRW method performance exceed the resolution III experiment performance with respect to accuracy.

The largest gap in accuracy between these two methods was seen for the Pred / Prey model. This model contains a large amount of oscillation, an environment where the WRW method was not expected to work well.

The WRW method performed faster than resolution III designed experiments on all but the single smallest tested model. Table 7.4 shows how many times faster the WRW method ran when compared to a resolution III experiment. There appears to be some correlation between model size and the speed benefit of the WRW method over a resolution III experiment. The Tree model is both the smallest model tested and, with a ratio of 0.42, the only tested model on

which the WRW method had a longer run time. With a ratio of 280, the largest tested model, Big Tree, was the simulation in which the largest run-time benefit for the WRW method over a resolution III experiment was seen.

When compared to strictly structural methods, Table 7.6 shows the WRW method does very well. Particularly compared to the Random Walks method, the increased accuracy offered by the WRW method demonstrates the value of adding weights to the one-step-transition probabilities as described in Section 4.3.2.

Since it involves additional steps, it is known with certainty that the WRW method will have a longer run time than the Random Balance (RB) experiment it requires. As previously mentioned, if direct analysis of the RB design yields better accuracy than the WRW method, there would be no benefit to the additional time required by the WRW method. As shown by Table 7.5, this is not the case; the WRW method performed more accurately than a direct analysis of its required RB experiment when applied to all test models.

In all cases, the WRW method performed well. This chapter showed the WRW method to be preferable with respect to accuracy over a random guess of importance, quicker and of comparable accuracy to resolution III screening experiments, and preferable over a simple random balance design of similar run time. Standard statistical factor screening experiments, such as resolution III 2^k partial factorial experiments seem preferable over the WRW method if accuracy is extremely important. But, if reductions in run time are more important than accuracy, the WRW method offers speed advantages on large models with minimal sacrifice to accuracy.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis successfully developed a method for identifying important input factors in large system dynamics models from an analysis based on those model's underlying structures. The use of random walks through the structure of a graph-based simulation model results in a powerful notion of relative importance on that graph. The power of random walks are greatly strengthened by the addition of importance-related weights to the arcs in the data flow graph. It is also shown that utilizing a model's graph structure in this way results in an analysis that can be considerably faster than the use of statistical experiments alone. Specifically, the resulting Weighted Random Walks for Importance Identification method was found to be comparably accurate to statistical factor screen experiments. Run time of the WRW method compared against a resolution III fractional factorial design was found to take between twice as long for the smallest model, but many times faster for the largest model.

8.2 Future Work

Using graph theoretic tools for the ranking of a node's importance in a data flow program appears to be an area of research where the potential for future work is great. Some examples for possible future research are given here, including suggestions for the improvement of the Weighted Random Walks method developed by this thesis.

In this work, an assumption was made that no prior information was available about the behavior of any node comprising a data flow program. While this assumption may be true in certain specific models, in general it is not the case. Many actors in a data flow program often belong to a well defined class, the behavior of which could be concisely exploited by a method, such as the Weighted Random Walks method. A brief investigation was made in Chapter 6 into using different resolution fractional factorial designs based on prior knowledge about the program under study, but this decision was limited to an entire-model scale; a weighting experiment design was chosen at the beginning of analysis and used consistently for each actor in the subject model. A more flexible technique would be to maintain some library of actor types and choose weighting experiment designs for each actor appropriately.

Furthermore, the method for assigning arc weights, as proposed in Section 4.3.2.1, poorly handles actors that implement strongly non-continuous, oscillating, or otherwise poorly behaved functions. This is a general limitation of two level factorial designs used to assign weights, and can be partially addressed by manipulation of the settable parameters to the Weighted Random Walks method. Although Chapter 6 showed the overall Weighted Random Walks method was fairly robust to poor weights, a better method than two-level fractional factorial designs likely exists for the assignment of weights. The weighting method on actors with internal state, as described in Section 4.3.2.4, could also be improved. Future work should focus in more detail on how best to assign arc weights to the elicited data flow graph.

As declared in the scope of work, Section 4.1, the goal of the Weighted Random Walks method developed by this thesis is to place a list of simulation inputs in order by their actual main effect for a given output. The ability to successfully attain this goal makes the Weighted Random Walks method a powerful input factor screening tool, albeit it is somewhat less detailed when compared against strictly statistical screening experiments. Future work should focus on the analyst's inability to set a meaningful cut-off when deciding which inputs to study further.

Additionally, this work focused on the atomic entities of a data flow program, or that model's individual actors. A simplistic explanation of the Weighted Random Walks method might be that it tries to extrapolate whole model behavior by studying a data flow program's individual components. Considerable efficiency may be realized if the individually studied

components were of a more course granularity. Multiple atomic actors could be grouped together to form one larger composite actor for use by the Weighted Random Walks method. An example of when this might be desirable is shown by Figure 8.1.

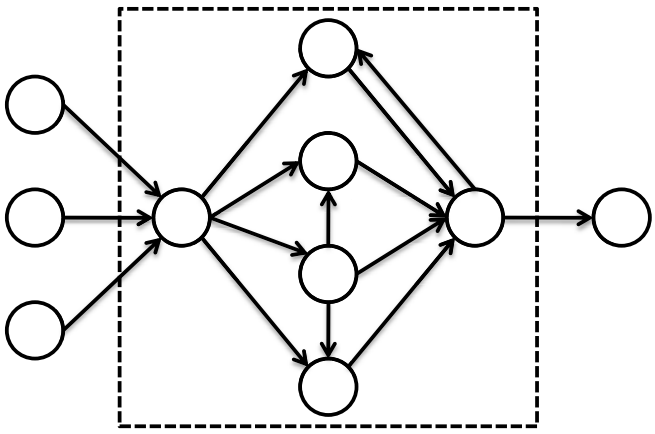

Figure 8.1: Example model where abstraction would speed up analysis by the WRW method.

In this fictional model, studying the actors bounded by the dotted box would significantly reduce the number of weighting experiments required, although each individual weighting experiment would take longer to complete.

Although already shown to be highly useful when applied as described by this work, the notion of ranking some nodes on a data flow graph as more important than other nodes has the potential to be of use for more applications than factor screening. One such potential application, similar in spirit to the work of (Faloutsos et al., 2004), would be to use some measure of intermediate node importance, combined with the measure of source node importance introduced by this work, in an attempt to find the most important sub-graph connecting important inputs to the output(s) under study. Any portions of the model not part of this most important sub-graph could be considered of low importance and excluded from the simulation for experimental purposes. The result would be a smaller, and therefore more quickly running, meta-simulation that may be able to predict the output of interest within an acceptable error.

Bibliography

J. Banks. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, Inc., New York, 1998.

F. Beichelt. *Stochastic Processes in Science, Engineering and Finance*. Taylor & Francis Group, London, 2006.

B. Bettonvil and J. Kleijnen. Searching for important factors in simulation models with many factors: Sequential bifurcation. *European Journal of Operational Research*, 96:180–194, 1996.

R. Bishop. *LabVIEW 8 Student Edition*. Pearson Prentice Hall, Upper Saddle River, NJ, 2007.

S. Borgatti. Centrality and network flow. *Social Networks*, 27:55–71, 2005.

C. Brooks, E. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous concurrent modeling and design in java. Technical report, Electrical Engineering and Computer Sciences, University of California Berkeley, 2008.

J. Buck. *Scheduling Dynamic Dataflow Graphs With Bounded Memory Using the Token Flow Model*. PhD thesis, University of California at Berkely, 1993.

W. Chang, S. Ha, and E. Lee. Heterogeneous simulation - mixing discrete-event models with dataflow. *Journal of VLSI Signal Processing*, 15:127–144, 1997.

L. Cook. Factor screening of multiple responses. In J. Swain, D. Goldsman, R. Crain, and J. Wilson, editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 174–180, Piscataway, NJ, 1992. IEEE.

J. Dennis. Data flow supercomputers. *Computer*, pages 48–56, 1980.

J. DeStefano. *Schaum's Outline of Theory and Problems of Feedback and Control Problems*. McGraw-Hill, New York, 1990.

E. Dijkstra. A note on two problems in connexion with graphs. *numerische Mathematik*, 1(1): 269–271, 1959.

S. Elaydi. *An Introduction to Difference Equations*. Springer-Verlag, 1996.

C. Faloutsos, K. Mccurley, and A. Tompkins. Fast discovery of connection subgraphs. In *Conference on Knowledge Discovery in Data*, pages 118–127, 2004.

T. Feng. Engineering structurally configurable models with model transformation. Technical report, Unversity of California at Berkeley, 2008.

F. Fouss, A. Pirotte, J. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with applications to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, 2007.

L. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, pages 215–239, 1979.

S. Goldberg. *Introduction to Difference Equations*. John Wiley & Sons, Inc., New York, 1958.

D. Greening. Modeling granularity in data flow programs. Master's thesis, University of California Los Angeles, 1988.

F. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software*, 4(3):250–269, 1978.

G. Harrison. Global stability of food chains. *The American Naturalist*, 114(3):455–457, 1979.

J. Kleijnen. *Handbook of Simulation*, chapter Experimental Design for Sensitivity Analysis, Optimization, and Validation of Simulation Models, pages 173–223. John Wiley & Sons, Inc., New York, 1998.

J. Kleijnen and W. Van Groenendaal. *Simulation: A Statistical Perspective*. John Wiley & Sons, Inc., New York, 1992.

E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

E. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12): 1217–1229, 1998.

J. O. Madahain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Jung - the java universal network / graph framework. *Journal of Statistical Software*, VV:1–35, 2005.

C. Mauro and D. Smith. Factor screening in simulation: Evaluation of two strategies based on random balance sampling. *Management Science*, 30(2):209–221, 1984.

M. Mojtahedzadeh, D. Anderson, and G. Richardson. Using digest to implement the pathway participation method for detecting influential system structure. *System Dynamics Review*, 20 (1):1–20, 2004.

D. Montgomery. Methods for factor screening in computer simulation experiments. Technical report, Georgia Institute of Technology, 1979.

D. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Inc., New York, 6th edition edition, 2005.

R. Oliva. Model structure analysis through graph theory: Partition heuristics and feedback structure decomposition. *System Dynamics Review*, 20(4):313–336, 2004.

L. Page, S. Brin, M. Rajeev, and W. Terry. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford, 1999.

J.-M. Proth and X. Xie. *Petri Nets: A Tool for Design and Management of Manufacturing Systems*. John Wiley & Sons, Inc., New York, 1996.

H. Shen and W. Hong. A hybrid method for simulation factor screening. In L. Perrone, F. Wieland, J. Liu, B. Lawson, D. Nicol, and R. Fujimoto, editors, *The Proceedings of the 2006 Winter Simulation Conference*, pages 382–389, Piscataway, NJ, 2006.

H. Shen and H. Wan. Controlled sequential factorial design for simulation factor screening. In M. Kuhl, N. Steiger, F. Armstrong, and J. Joines, editors, *The Proceedings of the 2005 Winter Simulation Conference*, pages 467–474, Piscataway, NJ, 2005.

M. Tadao. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4): 541–580, 1988.

T. Taylor, D. Ford, and A. Ford. Model analysis using statistical screening: Extensions and example applications. In J. Sterman, R. Oliva, R. Langer, J. Rowe, and J. Yanni, editors, *The 2007 International Conference of the System Dynamics Society*, Boston, 2007.

G. Watson. A study of the group screening method. *Technometrics*, 3:371–388, 1961.

S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Conference on Knowledge Discovery in Data*, pages 266–275, 2003.

Y. Zhou and E. Lee. Causality interfaces for actor networks. *ACM Transactions on Embedded Computing Systems*, 7(3):2–35, 2008.

Appendix A

Full Results for Application Specific Model Behavior Experiments

This appendix contains the full results of the experiments run for Chapter 6. If not otherwise mentioned, all run times are in milliseconds as described by Section 5.1.2 and all accuracy scores are expressed between 0.00 for worst performance and 1.00 for best performance, as described by Section 5.1.1.

The observed run time of the weighted reverse random walks heuristic method by number of random input configuration samples is given for the relevant experimental models by Table A.1 through Table A.4.

Table A.1: Weighted Random Walks method run time and 95% confidence interval half width by number of randomly sampled input configurations (*N* Samples) for *Tree* experimental model. Each run time is based off of an average of five samples, the Range Percentiles factor was set to (0, 100), the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

***N* Samples**	**Average Run Time (ms)**	**95% C.I. H.W. (ms)**
2	287	15
4	791	26
6	310	42
8	313	1
10	331	13
20	375	9
100	978	104

Table A.2: Weighted Random Walks method run time and 95% confidence interval half width by number of randomly sampled input configurations (N Samples) for *Big Tree* experimental model. Each run time is based off of an average of five samples, the Range Percentiles factor was set to (0, 100), the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

***N* Samples**	**Average Run Time (ms)**	**95% C.I. H.W. (ms)**
2	78900	2280
4	82700	853
6	88100	2810
8	95900	1360
10	104000	3430
20	136000	8850
100	401000	28930

Table A.3: Weighted Random Walks method run time and 95% confidence interval half width by number of randomly sampled input configurations (N Samples) for *Digraph* experimental model. Each run time is based off of an average of five samples, the Range Percentiles factor was set to (0, 100), the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

***N* Samples**	**Average Run Time (ms)**	**95% C.I. H.W. (ms)**
2	447	83
4	493	79
6	519	12
8	572	46
10	607	82
20	831	65
100	2250	245

The accuracy scores for the heuristic algorithm by number of random input configuration samples and the percentile of sampled data used as high and low factor values in the weighting experiments is given for the relevant experimental models by Table A.5 through Table A.9.

Table A.4: Weighted Random Walks method run time and 95% confidence interval half width by number of randomly sampled input configurations (N Samples) for *Serial Queue* experimental model. Each run time is based off of an average of five samples, the Range Percentiles factor was set to (0, 100), the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

***N* Samples**	**Average Run Time (ms)**	**95% C.I. H.W. (ms)**
2	766	633
4	1150	827
6	1460	979
8	1810	1110
10	2030	1270
20	3810	2150
100	15100	7810

Table A.5: Weighted Random Walks method accuracy by number of randomly sampled input configurations (*N* Samples) for the *Tree* experimental model. Each accuracy score is based off of an average of five samples, the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

(Low Percentile , High Percentile) used as Intermediate Range						
	(0, 100)			**(1, 99)**		
N Samples	Min	Mean	Max	Min	Mean	Max
2	0.98	0.98	0.99	*	*	*
4	0.85	0.94	0.99	*	*	*
6	1.00	1.00	1.00	*	*	*
8	0.98	1.00	1.00	*	*	*
10	0.99	1.00	1.00	*	*	*
20	1.00	1.00	1.00	1.00	1.00	1.00
100	1.00	1.00	1.00	1.00	1.00	1.00

Table A.6: Weighted Random Walks method accuracy by number of randomly sampled input configurations (*N* Samples) for the *Big Tree* experimental model. Each accuracy score is based off of an average of five samples, the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

(Low Percentile , High Percentile) used as Intermediate Range						
	(0, 100)			**(1, 99)**		
N Samples	Min	Mean	Max	Min	Mean	Max
2	0.81	0.85	0.87	*	*	*
4	0.94	0.96	0.97	*	*	*
6	0.97	0.97	0.98	*	*	*
8	0.98	0.98	0.99	*	*	*
10	0.98	0.98	0.99	*	*	*
20	0.99	0.99	0.99	*	*	*
100	1.00	1.00	1.00	1.00	1.00	1.00

Table A.7: Weighted Random Walks method accuracy by number of randomly sampled input configurations (*N* Samples) for the *Digraph* experimental model. Each accuracy score is based off of an average of five samples, the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

(Low Percentile , High Percentile) used as Intermediate Range						
	(0, 100)			**(1, 99)**		
N Samples	Min	Mean	Max	Min	Mean	Max
2	1.00	1.00	1.00	*	*	*
4	1.00	1.00	1.00	*	*	*
6	1.00	1.00	1.00	*	*	*
8	1.00	1.00	1.00	*	*	*
10	1.00	1.00	1.00	*	*	*
20	1.00	1.00	1.00	*	*	*
100	1.00	1.00	1.00	1.00	1.00	1.00

Table A.8: Weighted Random Walks method accuracy by number of randomly sampled input configurations (*N* Samples) for the *Serial Queue* experimental model and the *Cumulative System Exits* output. Each accuracy score is based off of an average of five samples, the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

(Low Percentile , High Percentile) used as Intermediate Range						
	(0, 100)			**(1, 99)**		
N Samples	Min	Mean	Max	Min	Mean	Max
2	0.75	0.75	0.75	1.00	1.00	1.00
4	0.46	0.74	1.00	1.00	1.00	1.00
6	0.46	0.69	0.75	1.00	1.00	1.00
8	0.46	0.69	0.75	1.00	1.00	1.00
10	0.46	0.57	0.75	1.00	1.00	1.00
20	0.46	0.51	0.75	1.00	1.00	1.00
100	0.46	0.46	0.46	1.00	1.00	1.00

Table A.9: Weighted Random Walks method accuracy by number of randomly sampled input configurations (*N* Samples) for the *Serial Queue* experimental model and the *Average Waiting in Queue 3* output. Each accuracy score is based off of an average of five samples, the Weighting Experiment Design parameter was set to Full Factorial, and the Proportion Iterations Recorded parameter was set to 100%.

(Low Percentile , High Percentile) used as Intermediate Range						
	(0, 100)			**(1, 99)**		
N Samples	Min	Mean	Max	Min	Mean	Max
2	0.34	0.86	1.00	0.67	0.87	1.00
4	0.34	0.67	1.00	1.00	1.00	1.00
6	0.16	0.63	0.99	1.00	1.00	1.00
8	0.49	0.73	1.00	1.00	1.00	1.00
10	0.16	0.63	0.99	0.75	0.95	1.00
20	0.58	0.81	0.99	1.00	1.00	1.00
100	0.49	0.63	0.83	1.00	1.00	1.00

Appendix B

Full Comparison of Selected Factor Screen Methods on Experimental Models

The following is a complete comparison of the various tested factor screen methods applied to all experimental models. See Chapter 7 for a complete discussion of the data presented here.

Method parameters are as follows:

- Full Factorial (Number of Replicates per Corner Point)
- Resolution XX (Number of Replicates per Corner Point)
- Random Balance (Number of Input Configurations)
- Weighted Random Walks (Input Configuration Samples, (Middle) Range Percentiles, Proportion Iterations Recorded)

A superscript number is used to indicate which of a model's outputs a label refers to. For models with multiple outputs, these references are given by Table B.1.

A * appears on the ground truth rows of the tables below to indicates an accuracy which is computed against itself, and must therefor be 1.00.

Table B.1: Full comparison of selected factor screen methods applied to *Tree* model.

Label	**Output**
Serial Queue[1]	Cumulative System Exits
Serial Queue[2]	Average Waiting in Q3
Queue Network[1]	Cumulative Exits from Out 1
Queue Network[2]	Average Waiting in Q6
Queue Network[3]	Cumulative Exits from Out 2
Pred / Prey [1]	Tiger Population Density
Pred / Prey [2]	Mice Population Density

Table B.2: Full comparison of selected factor screen methods applied to *Tree* model.

Method	**Sim. Reps.**	**Accuracy**	**Run Time (ms)**	**Type**
Resolution III (1)	10	*	131	Ground Truth
Random Balance (6)	6	0.73	278	Statistical
Shortest Path	0	0	15	Structural
Random Walks	0	0	15	Structural
WRW (6,1,1)	6	1.00	310	Hybrid

Table B.3: Full comparison of selected factor screen methods applied to *Big Tree* model.

Method	**Sim. Reps.**	**Accuracy**	**Run Time (ms)**	**Type**
Resolution III (1)	6562	*	24900000	Ground Truth
Random Balance (6)	6	0.45	20000	Statistical
Shortest Path	0	0	36	Structural
Random Walks	0	0	114	Structural
WRW (6,1,1)	6	0.97	89700	Hybrid

Table B.4: Full comparison of selected factor screen methods applied to *Digraph* model.

Method	**Sim. Reps.**	**Accuracy**	**Run Time (ms)**	**Type**
Full Factorial (1)	128	*	2850	Ground Truth
Resolution III (1)	8	1.00	938	Statistical
Random Balance (6)	6	0.74	353	Statistical
Shortest Path	0	0	36	Structural
Random Walks	0	0.77	15	Structural
WRW (6,1,1)	6	1.00	519	Hybrid

Table B.5: Full comparison of selected factor screen methods applied to *Serial Queue*[1] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Full Factorial (5)	640	*	71700	Ground Truth
Resolution IV (5)	80	1.00	11900	Statistical
Resolution III (5)	40	1.00	7800	Statistical
Random Balance (6)	6	0.73	1150	Statistical
Shortest Path	0	0.16	32	Structural
Random Walks	0	0.46	62	Structural
WRW (6,1,1)	6	1.00	1460	Hybrid

Table B.6: Full comparison of selected factor screen methods applied to *Serial Queue*[2] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Full Factorial (5)	640	*	71700	Ground Truth
Resolution IV (5)	80	1.00	11900	Statistical
Resolution III (5)	40	1.00	7800	Statistical
Random Balance (6)	6	0.68	1150	Statistical
Shortest Path	0	0.41	32	Structural
Random Walks	0	0.49	62	Structural
WRW (6,1,1)	6	1.00	1460	Hybrid

Table B.7: Full comparison of selected factor screen methods applied to *Queue Network*[1] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Full Factorial (5)	640	*	1080000	Ground Truth
Resolution IV (5)	80	0.97	137000	Statistical
Resolution III (5)	40	1.00	73900	Statistical
Random Balance (6)	6	0.81	10900	Statistical
Shortest Path	0	0.88	32	Structural
Random Walks	0	0.94	31	Structural
WRW (6,1,.1)	6	0.95	12500	Hybrid

Table B.8: Full comparison of selected factor screen methods applied to *Queue Network*[2] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Full Factorial (5)	640	*	1080000	Ground Truth
Resolution IV (5)	80	1.00	137000	Statistical
Resolution III (5)	40	1.00	73900	Statistical
Random Balance (6)	6	0.68	10900	Statistical
Shortest Path	0	0.45	32	Structural
Random Walks	0	0.47	31	Structural
WRW (6,1,1)	6	0.95	12500	Hybrid

Table B.9: Full comparison of selected factor screen methods applied to *Queue Network*[3] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Resolution IV (5)	80	*	1080000	Assumed Truth
Resolution III (5)	40	0.97	137000	Statistical
Resolution III (1)	16	1.00	27300	Statistical
Random Balance (6)	6	0.70	10900	Statistical
Shortest Path	0	0.00	32	Structural
Random Walks	0	0.47	31	Structural
WRW (6,1,.1)	6	1.00	12500	Hybrid

Table B.10: Full comparison of selected factor screen methods applied to *Pred / Prey*[1] model.

Method	Sim. Reps.	Accuracy	Run Time (ms)	Type
Full Factorial (1)	4096	*	21700000	Ground Truth
Resolution IV (1)	32	0.56	167000	Statistical
Resolution III (1)	16	0.98	81900	Statistical
Random Balance (6)	6	0.75	29600	Statistical
Shortest Path	0	0.40	31	Structural
Random Walks	0	0.53	109	Structural
WRW (6,1,.1)	6	0.79	34800	Hybrid

Table B.11: Full comparison of selected factor screen methods applied to *Pred / Prey*[1] model.

Method	**Sim. Reps.**	**Accuracy**	**Run Time (ms)**	**Type**
Full Factorial (1)	4096	*	21700000	Ground Truth
Resolution IV (1)	32	0.72	167000	Statistical
Resolution III (1)	16	0.98	81900	Statistical
Random Balance (6)	6	0.75	29600	Statistical
Shortest Path	0	0.56	31	Structural
Random Walks	0	0.59	109	Structural
WRW (6,1,.1)	6	0.94	34800	Hybrid