This paper presents a heuristic factor screening method for identifying important input factors in large synchronous data flow simulation models from an analysis based on those models’ underlying structures. Data flow requires encoding a model components’ dependencies in a graph format. The presented method views this graph as a stochastic process and attempts to rank the importance of inputs with respect to an output through the use of weighted random-walks through that graph. A comparison is made against other factor screening techniques, including fractional factorial experiments.

Key words: Simulation; Data flow models; Graph models; Model elicitation

History: Draft completed August 18th, 2009.

1. Introduction

Design and analysis of large-scale, complex systems often leads to the construction of models on which experimental analysis can be performed. The types of models that can be constructed typically have trade-offs among the level of detail, validity, and execution time. In cases where validity is of primary importance, large, detailed models having long execution times can result. Given such a model, one may want to determine which aspects of the model to focus on when conducting an experiment. Factor screening is often used in this case, but in some cases the model execution times may be too long to permit a full factor screening experiment.

In this paper, a method for analysis of node importance on graphs is adapted for application to the structure of models implemented as synchronous data flow programs. The expressiveness and natural concurrency of data flow programs makes this model of computation attractive for the implementation of complex and large simulation models. The presented algorithm, called Weighted Random Walks, approximates factor screening through the use
of random walks on a data flow program’s underlying graph with arcs weighted by studying sub-components of the model.

The remainder of this paper is organized as follows: Section 2 describes the background and related work. Section 3 presents an example model and discusses it’s representation as a graph. Section 4 introduces Weighted Random Walks for factor screening, and Section 5 contains a comparison of Weighted Random Walks against alternatives.

2. Background and Related Work

2.1. Factor Screening in Simulation

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 $|g| << |k|$ in most applications (Montgomery, 1979, Bettonvil and Kleijnen, 1996). For the remainder of this paper, an input will be considered of high importance if the absolute value of that input’s main effect on an output of interest is also high. The goal is to find input factors that, when considered individually, cause the output of interest to change the most.

Typically, factor screening in simulation is carried out through the use of designed statistical experiments. Some of these experiment based 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 Wan, 2006).

One type of designed experiment popularly applied for 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). One popular fractional factorial experiment are resolution III experiments, which allow all main effects in a model to be estimated in one run greater than the number of main effects to be estimated.

A limitation common to all statistical factor screen techniques is their reliance on model execution. This is especially disadvantageous on large models with long execution times. A further limitation of fractional factorial designs is the required aliasing of main effects with interaction effects.
2.2. Synchronous Data Flow

In data-flow programs, modules react only to the existence of data at their inputs. This is different from other models of computation such as “imperative”, where modules are executed in sequential order, and “discrete event”, where modules are executed based on a global schedule (Chang et al., 1997, Lee and Sangiovanni-Vincentelli, 1998). A data-flow program consists of a directed “data-flow graph” containing a set of processing nodes called “actors” connected by message passing arcs called “relations” (Dennis, 1980, Lee and Messerschmitt, 1987).

A node on the data-flow graph may be categorized by its degree as one of three main types. A node with no input arcs is referred to as a source node, a node with no output arcs a sink node, and a node with at least one input arc and at least one output arc is called an intermediate node.

Although often representative of simple arithmetic, intermediate nodes may perform operations of any complexity. One example of this power is demonstrated by Chang et al. (1997), where it is shown how to mix dataflow models with discrete-event models. The exact behavior of a node may not be known, or even easily expressible. Synchronous data-flow (SDF) programming, as is used in this work, is a special case of data-flow programming (Lee and Messerschmitt, 1987). Time in a SDF model progresses globally to the model in discrete quantities referred to as “ticks” or “iterations” Chang et al. (1997).

Synchronous data flow programs can concisely represent systems of difference equations, a tool widely used in both the social sciences and engineering, with some specific examples being their application to control theory problems, econometric models, queuing problems, and behavior learning (Goldberg, 1958). Data flow programming languages are also 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 is another widely available program that supports data flow programming (Brooks et al., 2008).

2.3. Node Importance on Graphs

The definition of node importance is domain dependent. In literature, the interpretation of importance usually further depends on the means used to compute it.

Numerous methods for determining a node’s importance have been proposed. Global
closeness and betweenness are two methods proposed by Freeman (1979) to determine the
importance of nodes on a graph. Since they rely heavily on distances, Freeman’s work
may not apply well in graphs where distance is not clearly related to importance (Borgatti,
2005). Since distances on a data flow graph do not clearly imply a magnitude of importance,
methods reliant on measures of distance may not apply well to the analysis of data flow
graphs.

Random walks on a graph have also been proposed for computing relative importance
between nodes (White and Smyth, 2003). Analysis by random walks can use the long-run
proportion of time a random walker is expected to spend at any given node as a measure
of how important that state is. One such example is the PageRank algorithm of Page et al.
(1999).

Similar to the long-run proportion of time a random walker is expected to be at any
given node, measures of a random walker’s expected first passage time and commute time
between states are other cited measures of relative importance between nodes (Fouss et al.,
2007).

Mojtahedzadeh et al. (2004) presents an analysis technique of graph-based models called
Pathway Participation Method, to identify the most important paths through a simulation
with respect to an output. Additionally, reachability between nodes of a graph based sim-
ulation model is a cited analysis technique (Oliva, 2004). An input, or component, in a
model may only effect other components if reachable on the model’s graph representation.
While simple reachability works well in most cases, the causality interfaces of Zhou and Lee
(2008) extends the ability to determine which nodes may effect other nodes in models where
causality information is known for the individual nodes.

3. Elicitation of Graph Structure and Example Model

Ptolemy II was used to build and execute the experimental data flow models. Most of a
data flow program’s data graph structure is clearly represented in a Ptolemy II model. An
exception not as easily retrieved are actor references to parameters and variables. Properly
integrating these variables into a model’s graph structure required an extra step of searching
for in scope variables.

For example, take the Ptolemy II implemented synchronous data flow program shown in
Figure 1. The shown initial parameter values are used as default low factor values. High
factor values are assigned as +50% low factor values. This model will be run for 5 iterations, and a full factorial experiment is used to compute the input’s main effects, which are 75.38 for \( A \), 18.85 for \( B \), and 82.58 for \( C \).

![Example data flow simulation model.](image1)

The graph structure elicited from this model is shown in Figure 2. Note how the parameters \( A \), \( B \), and \( C \) are properly connected to \( D \) and \( E \), the actors that reference them.

![Elicited data flow graph of example model.](image2)

4. Weighted Random Walks

4.1. Random Walks

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 \( p^0 \) (Beichelt, 2006).
The random walks method developed for ranking the relative importance of nodes on a graph requires representing that graph as a Markov chain. 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 defined as some fixed probabilities \( p_{ij} \) that the token will transition to node \( j \) at time \( (n + 1) \). This requires that \( p_{ij} = 0 \) if node \( i \) is not directly connected to node \( j \) and \( 1 \geq p_{ij} \geq 0 \) if \( i \) is directly connected to \( j \). Then, \( P^m \) gives a matrix of \( m \)-step transition probabilities for the system. This \( m \)-step transition probability matrix can be multiplied with \( p^0 \) to determine the \( m \)-step probability of being in a state given an initial probability distribution.

Take for example the data flow graph introduced above in Figure 2 and corresponding Markov chain structure in Figure 3. To satisfy the requirements that the one-step transition probability matrix be a stochastic matrix, as described by

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

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 equal 1.

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

Figure 3: Example one-step transition probability matrix for data flow graph.

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 $p^{(t)} = (0, 0, 0, 0, 0, 1)$ as $t$ approaches infinity (Beichelt, 2006). Or, using the fictional token example, regardless of where a randomly walking token is placed on this example graph, it will eventually end up trapped in State $G$. Another interpretation might be that Actor $G$ has an importance of 1 to each of the input actors $A$, $B$, and $C$.

Now reverse the direction that this fictional token walks across arcs. Transitions will now take place from the head of a directed arc to that arc’s tail. Instead of the sink node, the source nodes must now be made into absorbing states through the addition of self loops. A token starting in State $G$, for example, will transition to State $F$ with a probability of 1. The structure of the Markov chain built under this assumption, shown in Figure 4, is also an absorbing system because an absorbing state is reachable from all non-absorbing states.

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

Figure 4: 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 $C$. 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, random walks on a data flow graph attempt to recursively account for importance. Node $G$ in Figure 2, 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 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.

One way of choosing these missing transition probability values of the reverse Markov chain structure of Figure 2 is to assign them in a uniform way, assigning each unknown in a row to have the same probability.

Consider the actor in Figure 5. 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 2. As mentioned earlier, the specifics of $F$ are not necessarily visible from data flow’s perspective, and may not even be expressible.

![Diagram of Actor](image)

Figure 5: 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. This can be accomplished by normalizing each row in the one-step transition probability matrix so all non-zero elements of a row are equal and their sum is 1. Applying this assumption to the model depicted in Figure 2 results in the extension of Figure 4 into the complete one-step transition probability matrix of Figure 6.

The only additional 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 $p^0 = (0,0,0,0,0,0,1)$; or a randomly-walking token will start at Node $G$ with a probability of 1.00. The vector representing the solution to the long term behavior of this system can be approximated by recursively computing $p^{(t+1)} = p^t \times P$ for a sufficient number
\[
P = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0.33 & 0.33 & 0.33 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\]

Figure 6: One-step transition probability matrix for example data flow graph, generated using uniform assumption.

of iterations. Numerical approximation is favored over an exact solution since calculating an exact solution is computationally difficult on large models. For this example, \( p^t \) will approach \( p = (0.22, 0.22, 0.56, 0, 0, 0, 0) \) 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.

Intuitively, it makes sense to rank Node A and Node B as having equal importance with respect to Node G. 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 Node C to be ranked as more important than Node A and Node B with respect to Node G.

One of the main weaknesses of assuming uniform contributions 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 likely invalid for many actors in a data flow program. Consider again Node D in Figure 5. While we don’t know the behavior of this node from the program’s structure alone, a more detailed examination of the model might reveal the output of Node D to follow \( D(A, B, C) = 2A + B + C \). Although using uniform
contributions assumes $A$, $B$, and $C$ to have equal importance to $D(A, B, C)$, in reality $A$ appears more important than $B$ or $C$. Given this knowledge, the transition probability from Node $D$ to Node $A$ could have be adjusted to be twice that of Node $B$, resulting in a long term residence probability vector of $p = (0.29, 0.15, 0.56, 0, 0, 0, 0, 0)$, which correctly distinguishes the differing magnitude of main effect between Node $A$ and Node $B$. Weights can be systematically added to all arcs on the data flow graph to further improve accuracy.

4.2. Arc Weighting Process

To be useful, weights assigned to a graph for use in determining importance on that graph must themselves be related in some way to importance. To be of practical benefit on large models, any weighting must be automated. If importance of an input with respect to an output in a simulation model is defined as that input’s magnitude of main effect on that output, the arc weights used for importance identification should be related to main effects in the model under study. To accomplish this, arc weights will be assigned as the main effect a given arc contributes to its target node’s output.

The actor shown in Figure 7 takes three inputs, computes the specified function, and produces the result of the specified function on its output arc. 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 $X$.

\[
2X + Y + Z
\]

Figure 7: 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 always be examined, it can still be evaluated by executing its implementing actor. In this case with no interaction effects, a resolution III $2^k$ fractional factorial experiment would be sufficient for this purpose. In general, a full factorial experiment can be used.
The main effect of an input on the output of this example actor can also depend on the magnitude of the input factor levels. The main effect values of 1 for inputs $Y$ and $Z$, and 2 for $X$ 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, how these global inputs interact to determine the ranges which will be seen by the individual intermediate actors is not straightforward. If inputs $X$, $Y$, and $Z$ in Figure 7 take their values from other actors, their ranges will not be known ahead of time.

The ranges for 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. Second, inputs to an intermediate actor may change between iterations of a replication if the model contains recursion. 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 for use in factor screening. Two factor levels are required to use a two level experiment, as indicated for this arc weighting process. These two levels will 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, such as with the actor in Figure 8, shown with possible input ranges. Since it is squared in the given expression, the main effect of 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 $100^2 - 99^2 = 199$.

$$F(X,Y,Z) = 2X + Y + Z^2$$

Figure 8: 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 additional analysis. Luckily, as is the case with many phenomenon, it is possible to find an approximation of the intermediate input distributions through a small random sample of input settings to
the full model.

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 through a directed cycle, an actor with internal state may exhibit time-varying behavior in isolation. A common example of an actor with internal state is an accumulator actor, which 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, all weighting experiments performed on an actor with internal state will be carried out for the same number of iterations as the full model under examination. The response variable of the weighting experiment will be the mean output over this time period. 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.

4.3. Weighted Random Walks

After selecting arc weights, the process of importance ranking continues as described above for the uniform transition probability case. A one-step probability matrix is built by normalizing the arc weights to conform to the requirements detailed in Equation (1). Because only the magnitudes of main effects are 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 continues exactly as described for the uniform variation of this method. The full process of applying Weighted Random Walks 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.

Step 2: Generate the reverse Markov chain model structure. 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 settings for use in arc weighting and generate the list of required random settings following a random balance structure.
Step 4: Execute the simulation model at least once for each random input settings, 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 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, Steps 1 and 2:

**Require:** Graph representation $G$ of model under study.
1: $A \leftarrow$ List of all actors in $G$.
2: $P \leftarrow |A| \text{ by } |A|$ 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, Steps 3 and 4:

**Require**: Stage 1 variables.
**Require**: An integer \( nReps > 0 \) number of random input settings 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: \( RB_{\text{Results}} \leftarrow \) Execute all replications of \( RB \) for \( TimeSpan \) iterations, recording all values passed across input arcs to actors in list \( \text{NeedsWeights} \).

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

**Require**: Stage 1 and Stage 2 variables.
**Require**: A number \( 0 < \text{factorPercentile} \leq 0 \) to select high / low factor values for individual arcs.

1: for all Actor \( a \) in \( \text{NeedsWeights} \) do
2:   for all Actor \( p \) in \( \text{Predecessors}(a) \) do
3:     \( Vals \leftarrow RB_{\text{Results}}(p,a) \) \{Assign to \( Vals \) the list of all values passed from \( p \) to \( a \) during execution of \( RB \).\}
4:     \( \text{SortAscending}(Vals) \)
5:     \( \text{LowFactor}(p,a) \leftarrow Vals[\lfloor |Vals| \times \text{factorPercentile} \rfloor] \)
6:     \( \text{HighFactor}(p,a) \leftarrow Vals[\lceil |Vals| \times (1 - \text{factorPercentile}) \rceil] \)
7:   end for
8: end for
9: for all Actor \( a \) in \( \text{NeedsWeights} \) do
10:  \( FF \leftarrow \) Generate full factorial experiment for node \( a \) using factor values for arcs as previously defined in \( \text{LowFactor} \) and \( \text{HighFactor} \).
11:  \( FF_{\text{Results}} \leftarrow \) List of main effects resulting from execution of \( FF \).
12:  \( \text{TransitionProb} \leftarrow \text{Normalize}(FF_{\text{Results}}) \)
   \{Normalize main effects into single-step transition probabilities for arcs into \( a \). Direction of transition will be wrong until transposed on line 14.\}
13:  for all Actor \( p \) in \( \text{Predecessors}(a) \) do
14:   \( P_{a,p} \leftarrow \text{TransitionProb}(p,a) \)
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.
**Require**: Node \( y \) representing output of interest.
1: $p^0 \leftarrow \text{Vector of size } |A| \text{ with elements initialized to 0}.$
2: $p^0_y = 1 \{ \text{Set initial probability vector to indicate starting at output of interest} \}
   \{ \text{with probability of 1} \}.$
3: repeat
4: \hspace{1em} $p^{(t+1)} = p^t \times P$
5: until StoppingCondition \{ \text{For example, until } p^t \text{ converges to some threshold} \}
   \{ \text{or until some maximum number of loop iterations.} \} \}
6: return $p$

When applied to the example of Figure 1, Weighted Random Walks yields the results of Table 4.3. The arc weights used for the Weighted Random Walks procedure on this example were based off 6 replications in the intermediate arc sampling stage and intermediate high / low factors were chosen as the minimum and maximum observed intermediate values, respectively.

Table 1: Results of random walk methods for factor screening in example model.

<table>
<thead>
<tr>
<th>Input</th>
<th>Main Effect</th>
<th>Random Walks</th>
<th>Weighted Random Walks</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>75.38</td>
<td>0.22</td>
<td>0.32</td>
</tr>
<tr>
<td>B</td>
<td>18.85</td>
<td>0.22</td>
<td>0.08</td>
</tr>
<tr>
<td>C</td>
<td>82.58</td>
<td>0.56</td>
<td>0.60</td>
</tr>
</tbody>
</table>

As shown, both Random Walks and Weighted Random Walks correctly identify $C$ as the most important input. Weighted Random Walks additionally identifies $A$ and $B$ correctly as the second and least most important inputs, respectively.

5. Comparison to Alternative Factor Screen Methods

5.1. Measuring Performance

Run times for the tested factor screening methods were measured by the internal clock of the test computer, using markers in the Java implementation of the 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.

Accuracy of factor screening is measured through the formulation of an Accuracy Score for measuring the correctness of factor screening to place a set of actual factor main effects into their correct order by magnitude. The rationale for this comparison criteria for factor
screening methods is that, regardless of the magnitude of effect chosen as the threshold between important and unimportant, if an analyst selects the top \( n \) inputs as ranked by a factor screen method, those \( n \) inputs should be the actual \( n \) inputs with the greatest magnitude of main effect.

For each input to a simulation model there is a pair of importance \( I = (a, e) \) such that \( a \) is the actual main effect magnitude for an input as measured by running a full factorial experiment on the simulation, and \( e \) is the estimated importance of that input returned by a factor screening method. Let \( I \) be the set of all \( (a, e) \) in a model; \( |I| = k \). Any inputs that must have a zero main effect due to no reachability in the data flow graph are excluded from \( I \) and \( k \). There is then a list of all \( I \in I, \Theta = \{(a_1, e_1), (a_2, e_2), \ldots, (a_i, e_i), \ldots, (a_k, e_k)\} \) ordered by \( e_1 \geq e_2 \geq \ldots \geq e_k \) that describes the ordered-list output from factor screening. 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 \( \Theta \) 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 receive an *Accuracy Score* of zero. The ordered list \( \Theta \) 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, \( \Theta \), let \( \Theta^* \) be a list of \( I \) ordered such that \( a_1 \geq a_2 \geq \ldots \geq a_k \) and \( \Theta^0 \) to be a list of \( I \) ordered such that \( a_1 \leq a_2 \leq \ldots \leq a_k \).

If \( \Theta(i), \Theta^*(i), \text{ and } \Theta^0(i) \) return element \( a_i \) of \( \Theta, \Theta^*, \text{ and } \Theta^0 \) respectively then *Accuracy Score* is defined as

\[
\text{Accuracy Score} = \frac{\sum_{i=0}^{k} \sum_{j=0}^{i} (\Theta(j) - \Theta^0(j))}{\sum_{i=0}^{k} \sum_{j=0}^{i} (\Theta^*(j) - \Theta^0(j))}.
\]

For example, given a list of 100 random numbers normalized to one that represent actual main effect, each paired with a different random number representing estimated importance. When sorted by descending *estimated importance* this list is \( \Theta \), or “Random” due to the means used to generate this list for this example. When sorted by descending *actual main effect* this list is \( \Theta^* \) and when sorted by ascending *actual main effect*, \( \Theta^0 \). Figure 9 shows a plot of the cumulative values of these three lists for an \( I \), generated randomly as just described.

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 (2). 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, which is 0.48. In general, for a randomly generated $I$, an accuracy score of 0.50 would be expected. The more correctly $\Theta$ is ordered in this example, the closer $\Theta$ will be to $\Theta^*$, the the closer “Cumulative Random” line would be to the “Cumulative Descending” line, and the closer the ratio described by Equation (2) would be to 1.00.

### 5.2. Experimental Models

Already shown useful for the small example model of Figure 1, the following sections show Weighted Random Walks tested on a variety of more complex experimental models. A summary of the different experimental models is given by Table 2. The “Interaction” Effects column denotes if a given model contains interaction effects. 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 output. If a model contains “Random” elements, there may be variability in its outputs requiring multiple replicates for each experimental settings.

Information on size and complexity properties of the experimental models is given by Table 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,
Table 2: Summary of important categorical properties of the experimental models.

<table>
<thead>
<tr>
<th>Model Name</th>
<th>Interaction</th>
<th>Cycles</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Tree</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Queue Network</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Predator-Prey System</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

“# Nodes” and “# Edges” denotes the number of nodes and edges that exist in the graph structure elicited from a given model.

Table 3: Summary of important properties of the experimental models.

<table>
<thead>
<tr>
<th>Model Name</th>
<th># Inputs</th>
<th>Iterations</th>
<th># Nodes</th>
<th># Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Tree</td>
<td>6,561</td>
<td>1</td>
<td>16,404</td>
<td>16,404</td>
</tr>
<tr>
<td>Queue Network</td>
<td>18</td>
<td>5,000</td>
<td>127</td>
<td>163</td>
</tr>
<tr>
<td>Predator-Prey System</td>
<td>19</td>
<td>20,000</td>
<td>70</td>
<td>92</td>
</tr>
</tbody>
</table>

The “Big Tree” model was generated in the following way. Starting with a root actor, three additional actors were created. Each of the actor nodes in the “Big Tree” model are expressions that take three input values, multiply each input by the weight of its respective input arc, then return the resulting sum. These three additional 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 were given a randomly assigned coefficient between 0 and 2 in its referencing actor’s expression. Only one output, the root node, is analyzed in this model, and all inputs to this model are assigned high values of 2 and low values of 1.

The “Queue Network” model relies heavily on a 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.
The behavior of each individual queue is governed completely by two difference equations. The first:

\[ \text{Departures}(t + 1) = \min(\text{NumInQueue}(t) + \text{Arrivals}(t), \text{Pois}(\text{Servers} \times \text{ServicesPerServer})) \]

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

\[ \text{NumInQueue}(t + 1) = \text{NumInQueue}(t) + \text{Arrivals}(t) - \text{Departures}(t) \]

Updates the state variable \( \text{NumInQueue} \) tracking the queue’s current size. \( \text{Pois}(\lambda) \) returns a random sample from a Poisson distribution with a rate of \( \lambda \) and \( \min(x, y) \) returns the minimum of \( x \) and \( y \).

The “Queuing Network” model has two system arrival points, \( \text{Arrival 1} \) and \( \text{Arrival 2} \). Customers arrive to each following a Poisson distribution with a rate of \( \text{Arrival Rate 1} \) to the first arrival point and \( \text{Arrival Rate 2} \) to the second. The structure of the system modeled by the “Queuing Network” model is shown by Figure 10. Arcs in Figure 10 denote the flow of customers between queues. When given an option of two queues to proceed to, customers randomly choose one alternative with each being equally likely.

![Figure 10: Queuing system represented by Queue Network model.](image)

Figure 10 only shows the general structure of the “Queueing Network” model, not the detailed model as implemented. Figure 11 shows the full data flow graph from the data flow implementation of the “Queueing Network” model. Each node on this graph represents an actor in the data flow program.
All inputs to the “Queueing Network” model were chosen in a random manner. The resulting inputs put all queues in steady state, where $ServiceRate > ArrivalRate$, except for queue 6 which is a bottleneck when either its $Servers$ or $ServicesPerServers$ inputs are set to their low values.

The final example model is one of a multiple-species Lotka-Volterra system, based on the concepts presented by (Harrison, 1979) and referred to here as the “Predator / 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 specie’s 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 example model the two predator species are Fox and Tigers and they feed on the two prey species, Mice and Deer. Tigers prefer to eat Deer and Fox prefer to eat Mice, but both predator species will eat either prey at varying levels of success. The full data flow graph of the data flow implementation of this model is shown in Figure 12.

The high and low factor values that will be used in this model were generated in a random manner, resulting in the extinction of any species group being rare.
5.3. Experimental Results

The factor screening performance of the Weighted Random Walks (WRW) method on the experimental models was tested against both expected random, and $2^k$ fractional factorial experiments. In all applications of Weighted Random Walks, the experimental models were run for 6 random input settings in the intermediate range sampling stage. To account for the variability in WRW accuracy performance introduced by randomly generating an input settings sample, WRW was applied 5 times for each model and the minimum, mean, and maximum accuracy scores were recorded. In the Predator and Prey model, high and low factor values were selected for the arc weighting experiments as the minimum and maximum intermediate values observed during the arc sampling stage. For the Queueing Network, high and low factor values for the arc weighting experiments were chosen as the 1st and 99th percentiles of observed values.

Multiple outputs were studied for the Queue Network and Predator and Prey experimental model. These outputs are indicated by a superscript number after the model name, as given in Table 4.

The 99.99th percentile accuracy score expected if list order were assigned randomly was determined through simulation. The input factors were randomly shuffled 50 million times, grouped into sets of five, and the resultant 99.99th percentile accuracy scores for the five-sample averages were recorded. These percentiles for the experimental models are shown in Table 5 alongside the accuracy returned from the Weighted Random Walks results. In
Table 4: Model label and referenced output for models with multiple outputs under study.

<table>
<thead>
<tr>
<th>Label</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Network 1</td>
<td>Cumulative Exits from Out 1</td>
</tr>
<tr>
<td>Queue Network 2</td>
<td>Average Waiting in Queue 6</td>
</tr>
<tr>
<td>Queue Network 3</td>
<td>Cumulative Exits from Out 2</td>
</tr>
<tr>
<td>Pred / Prey 1</td>
<td>Tiger Population Density</td>
</tr>
<tr>
<td>Pred / Prey 2</td>
<td>Mice Population Density</td>
</tr>
</tbody>
</table>

all cases, WRW performed better than would be expected if input factor importance were guessed at random.

Table 5: 99.99th percentile mean accuracy scores expected if order is determined randomly and mean WRW accuracy scores for experimental models.

<table>
<thead>
<tr>
<th>Model</th>
<th>99.99th Percentile Random</th>
<th>WRW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Tree</td>
<td>0.52</td>
<td>0.97</td>
</tr>
<tr>
<td>Queue Network 1</td>
<td>0.86</td>
<td>0.95</td>
</tr>
<tr>
<td>Queue Network 2</td>
<td>0.85</td>
<td>0.95</td>
</tr>
<tr>
<td>Queue Network 3</td>
<td>0.87</td>
<td>1.00</td>
</tr>
<tr>
<td>Pred / Prey 1</td>
<td>0.74</td>
<td>0.79</td>
</tr>
<tr>
<td>Pred / Prey 2</td>
<td>0.80</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Next, using a $2^k$ fractional factorial design of resolution III for factor screening is compared against WRW with parameters set as described above. The results of this comparison are shown in Table 6. “Queue Network*” in the following tables indicates a comparison against resolution III design with one replicate for each corner point, opposed to the 5 replicates per corner point resolution III design used for the “Queue Network”, and every other row.

WRW performed of comparable accurately to the resolution III experiments at magnitude order ranking. Most deviations between the two methods was due to the incorrect ordering of relatively unimportant input factors. In no experimental model did the average WRW performance exceed the resolution III experiment performance with respect to accuracy.

The run times of the comparison shown by Table 6 are expressed as the ratio of resolution III run time to WRW run time in Table 7. This ratio is called the “Speed Ratio” and is
Table 6: Comparison of Weighted Random Walks (only mean accuracy shown) for factor screening versus Resolution III $2^k$ fractional factorial design for experimental models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Resolution III</th>
<th>WRW</th>
<th>Resolution III</th>
<th>WRW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Min</td>
<td>Mean</td>
<td>Max</td>
</tr>
<tr>
<td>Big Tree</td>
<td>1.00</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>Queue Network</td>
<td>1.00</td>
<td>0.88</td>
<td>0.95</td>
<td>1.00</td>
</tr>
<tr>
<td>Queue Network</td>
<td>1.00</td>
<td>0.72</td>
<td>0.95</td>
<td>0.99</td>
</tr>
<tr>
<td>Queue Network</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Queue Network</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>Pred / Prey</td>
<td>0.98</td>
<td>0.78</td>
<td>0.79</td>
<td>0.80</td>
</tr>
<tr>
<td>Pred / Prey</td>
<td>0.98</td>
<td>0.92</td>
<td>0.94</td>
<td>0.95</td>
</tr>
</tbody>
</table>

The measure of how many times faster WRW performed than the resolution III designed experiment.

Table 7: Comparison of speed advantage of Weighted Random Walks versus resolution III $2^k$ fractional factorial design for experimental models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Speed Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Tree</td>
<td>280</td>
</tr>
<tr>
<td>Queue Network</td>
<td>5.9</td>
</tr>
<tr>
<td>Queue Network$^*$</td>
<td>2.2</td>
</tr>
<tr>
<td>Pred / Prey</td>
<td>2.3</td>
</tr>
</tbody>
</table>

As seen from Table 6, WRW performed at least twice as fast as the resolution III screening experiments on all experimental models tested. The most substantial speed benefit was observed for the largest experimental model, “Big Tree”.

A more detailed experimental analysis of Weighted Random Walks is given in Tauer (2009). This more detailed analysis includes an investigation into the impact of varying WRW’s parameters, such as the number of random input settings to run during the intermediate range sampling stage, as well as the application of WRW to additional experimental models.
6. Conclusions

This paper presented a heuristic technique for identifying important input factors in synchronous data flow models 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 application of random walks for factor screening is greatly strengthened by the addition of meaningful weights to the arcs in the data flow graph. The resulting Weighted Random Walks method was found to be comparably accurate to statistical factor screen experiments, with run times faster for all experimental models tested.

References


