SuperDecisions Limit Matrix Calculations

Bill Adams
Decision Lens Inc.
DeLand FL, USA
wjadams@decisionlens.com

March 1, 2011

Abstract

The SuperDecisions Software for the Analytic Network Process (ANP) offers several different algorithms for computing the limit matrix in an ANP model. The various algorithms are designed to address two shortcomings with the standard limit matrix calculation. In this paper we describe those issues, the algorithms themselves, and how they address the issues.

1 Introduction

ANP models are defined in [1] and the basic limit matrix calculations are outlined. SuperDecisions is a program that allows one to create, edit, and do various calculations on ANP models. The particular calculation we are focusing on is the limit matrix calculation.

Recall that ANP models are a generalization of AHP models, and even more particularly trees (see [2] for a thorough introduction to AHP theory). For trees, synthesizing global priorities is simply a matter of multiplying the local priority (normally found via pairwise comparisons) by the global priority of the parent. In ANP theory this process is replaced by the limit matrix calculation.

1.1 Naive algorithm

The standard algorithm can be described quickly (if we are allowed to be a bit terse, for further information see [1]). The input to the limit matrix calculation is the scaled supermatrix (that is a matrix of local weights that has each entry scaled by the appropriate cluster priority). The naive definition of the limit matrix calculation is to perform the following steps.

- Raise the matrix to larger powers, and either
- the powers will converge to a given matrix (the limit matrix), or

- the powers will converge to a cycle of matrices and the limit matrix is the average of these.

This algorithm performs remarkably well except in two circumstances which we outline below.

1.2 Problems

The algorithm outlined above has issues with handling hierarchies and networks with sinks. In fact, since hierarchies are networks with sinks, we can simply say networks with sinks cause issues. Of course there is a standard solution to these issues (with its own problems which we discuss in later sections), but for now let us outline the problems our naive limit matrix calculation has. They are as follows.

Hierarchies: In the case of hierarchies the larger powers of the scaled supermatrix eventually go to zero and thus we are left with a limit matrix containing all zeros (in other words no nodes get any scores, a most unsatisfactory result).

Sinks in general: Even with networks with feedback, if we have sinks in the network (nodes without connections emanating from them) then the large powers of the scaled supermatrix will still tend toward zero (or at the least have many columns going to zero which should not).

Both of these problems can be viewed as problems related to sinks, which manifest themselves in the scaled supermatrix's powers going toward zero. There is a single standard fix which addresses both issues simultaneously, but with drawbacks. Let us turn to analyzing this standard solution.

1.3 Standard Solution of the Problem

As outlined in [1], to calculate the limit matrix for hierarchies as well as networks with sinks in general one can add self-loops to the sinks. [1] This single fix makes the limit matrix not go to zero in both cases, and in addition gives the correct values for the alternatives in an AHP model.

However, in the case of AHP models, the only nodes getting non-zero values out from this calculation are the sinks themselves. This means we lose the information about the non-sink nodes (something which is very straightforward to calculate in AHP theory). This is the first draw back.

Secondly, if we add self-loops we have changed the model in a way that the designer of the model may not have wanted. This is more of an aesthetic complaint; however, it should not be overlooked. If a user designs a model in SuperDecisions and does not place a self-loop on a sink and the software

[1] In terms of the super matrix, if node number c is a sink then the scaled supermatrix has all zeros in column c. Putting a self-loop on that node corresponds to replacing the c^{th} column with a column with a one in the c^{th} row, and zeros elsewhere.

automatically puts that self-loop in for a calculation, the user loses control of the model. It is as if Microsoft Word changed random words to bold-face font on a user.

Lastly, adding self-loops can give results that are counter-intuitive, and make the situation seem more difficult than it needs to be. For instance, consider a model with two nodes (Node1 and Node2), with Node1 connected to itself and Node2, and Node2 being a sink. Assume Node1 and Node2 are equally weighted with respect to Node1. Then the weighted supermatrix would look like

$$\mathcal{S} = \begin{bmatrix} 0.5 & 0 \\ 0.5 & 0 \end{bmatrix}.$$

If we add a self loop the matrix becomes the following (with the given limit)

$$\mathcal{S} = \begin{bmatrix} 0.5 & 0 \\ 0.5 & 1 \end{bmatrix} \text{ with limit} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}.$$

In other words all priority ends up flowing to Node2. If we had originally setup our model to have the self-loop on Node2, of course we might expect this result. However, as originally stated we have essentially one priority vector of $(0.5, 0.5)$ for Node1 and Node2 with respect to Node1. Since that is the only information we gave the model it seems very odd that all priority would eventually drift to Node2.

1.4 Proposed Solution

Instead of adding self-loops at sinks (and thus changing the structure of the model) there are many other tactics one can use to address the issues hierarchies and sinks bring. SuperDecisions has several different algorithms the address the issue by changing the basic rules of the naive limit matrix algorithm outlined earlier.

2 Limit Matrix Calculations

The issues mentioned above break down rather naturally into Hierarchy and Sink types so we will discuss the limit matrix algorithms for hierarchies and sinks.

2.1 Hierarchies

The problem of a hierarchy's supermatrix going to zero can be fixed by adding self-loops at the sinks (as mentioned before), and this does give correctly synthesized values for the sinks, but all other nodes get zeros from this calculation. There is a straight forward way to remedy this situation.

Rather than calculate the limit matrix using the naive definition, we could do a slightly different calculation that has the same effect as synthesizing the hierarchy by hand (without using the supermatrix at all). The modified calculation is given by the formula

$$\text{limitMatrix} = \sum_{i=1}^{L} \mathcal{S}^i \tag{1}$$

where $\mathcal{S}$ is the scaled supermatrix, and L is the number of levels in the hierarchy (or alternatively L is the first integer for which $\mathcal{S}^{L+1} = 0$). In essence, one raises the supermatrix to powers to find the power at which all entries become zero; the sum of those powers is the limit matrix. All limit matrix options except *Identity at sinks* uses this method to calculate the limit matrix of hierarchies.

Note 1. *In SuperDecisions the "Identity at Sinks" limit matrix option does the self-loops calculation method for hierarchies. Every other limit matrix option uses the alternate formula written above.*

2.2 Sinks

There are several approaches one can take to handling sinks (without adding self-loops). These approaches break down into the following two general philosophies.

Rescaling: This philosophy essentially calculates the limit matrix using the standard approach, except it rescales each power before checking for convergence or cycling (note, it does not actually use the rescaled matrices to calculate the next power, the rescaled versions are only used for convergence testing and for final result).

Decomposing: This philosophy breaks down the network into component pieces that are easier to understand and compute. Then those pieces are combined together to give back a limit matrix.

Let us consider each philosophy in turn and describe how the various algorithms utilize that philosophy.

2.2.1 Rescaling

There are two limit matrix calculations in SuperDecisions that utilize a rescaling technique. They are the default *Calculus Type* and the *Scaling by Scalar* calculation [2]. In both cases the algorithm is the standard algorithm, except convergence and the actual matrix returned are slightly changed. That is, both algorithms calculate larger and larger powers of the scaled supermatrix, and look for convergence or cycling. The difference is in how they go about looking for convergence.

Calculus type: In this algorithm, after every power is calculated a column normalized version of that power matrix is stored. Convergence is checked on these normalized matrices, and the column normalized result is returned.

[2]In both cases, if the model is actually an AHP model, it merely uses the previously mentioned limit matrix calculation.

It is called the calculus type because the powers may go to zero, but the column normalized versions of the matrix powers may not (like looking at the ratios of ΔY over ΔX as ΔX goes to zero in a derivative calculation).

Scaling by scalar: This algorithm is essentially the same as the previously mentioned one; however, instead of column normalization we normalize by multiplying the entire matrix by a scalar. The scalar we multiply by is the one that makes one column (the one with the largest sum) sum to 1, and the rest sum to something less than one. The scalar normalized versions of the powers are used to check for convergence/cycling, and the scalar normalized result is returned.

2.2.2 Decomposition

These algorithms decompose the input network (and thus the scaled supermatrix) into component pieces, and calculate on those components, then combine to give a final limit matrix. There are two types used, the "New Hierarchy" and the "Sinks formula" (each of which have two subtypes).

Note 2. *Every time a matrix is broken into components as shown below, we first renormalize those components before applying a limit matrix calculation to those components.*

Sinks formula: In this case the network is broken down into sink nodes and non-sink nodes. In that case, if we reorder the nodes so that the sink nodes are last, the supermatirx would be of the form

$$\begin{bmatrix} B & 0 \\ A & 0 \end{bmatrix}$$

where B is the supermatrix corresponding to the non-sink nodes and A is the matrix connecting the non-sink nodes to the sink nodes [3]. Both versions of the sinks formula calculate the limit first as

$$\begin{bmatrix} limitB & 0 \\ A \times limitB & 0 \end{bmatrix}$$

where $limitB$ is the limit of the B matrix using the calculus type calculation [4]. The sinks formula (straight normalize) normalizes this matrix and returns the result. The sinks formula (normalize limitB) rescales the limitB portion of the above matrix to normalize the returned matrix.

New hierarchy: With this algorithm the ANP model is broken up into two pieces, the hierarchical component and the network component. The hierarchical component consists of all nodes whose columns are identically

[3]Notice that the matrices B and A can have different sizes. In fact of there are s sink nodes and n non-sink nodes, then B has dimension $n \times n$ and A has dimension $s \times n$.

[4]Since there are no sinks, this is just the naive limit matrix calculation

zero in a large enough power of the scaled supermatrix. The network component is the rest of the ANP model. If we reorder the nodes so that the hierarchical nodes occur in the last columns of the supermatrix, the scaled supermatrix would be of the form

$$\begin{bmatrix} B & 0 \\ A & C \end{bmatrix}$$

where B is the supermatrix for the network component and C is the super matrix for the hierarchical component and A is the matrix connecting the network component to the hierarchical component [5]. The new hierarchy (no limit) calculation is then

$$\begin{bmatrix} limitB & 0 \\ A \times limitB + B \times C & limitC \end{bmatrix}$$

where $limitC$ and $limitB$ are the limit matrices of C and B respectively (calculated using the calculus type calculation, although that agrees with the standard calculation since neither C nor B has any sinks). For the new hierarchy with limit calculation everything is the same except that the lower left corner's multiplication and add is performed repeatedly until it converges.

3 Synopsis

The following is a table of the various limit matrix calculations and a brief recapitulation of how the calculations behave. Further details about those calculations can be found elsewhere in this paper.

Calculus Type: This calculation essentially calculates larger and larger powers of the scaled supermatrix, and checks for convergence or cycling. It checks for convergence by comparing the entries of the powers of the scaled supermatrix (after renormalizing each column). It returns the limit after renormalizing each column. However it does not use those renormalized matrices for calculating further powers. In addition it uses the sum formula (see equation 1) to calculate the limit matrix of a hierarchy.

Scaling by scalar: This is the same as the calculus type except that it checks for cycling by renormalizing by multiplying the entire matrix by a constant so that all columns add to something less than or equal to one, and at least one column sums to precisely one. It returns that rescaled matrix as its result, but does not use the rescaled matrices for calculating further powers.

[5] Notice that the matrices A, B, and C may be off different sizes. In particular if there are h hierarchical nodes and n non-hierarchical nodes, then B has dimension $n \times n$, A has dimension $h \times n$ and C has dimension $h \times h$.

New hierarchy (no limit): This breaks up the matrix into two pieces, the hierarchical piece and the network piece, so that the scaled supermatrix breaks up into a form like

$$\begin{bmatrix} B & 0 \\ A & C \end{bmatrix}.$$

where B is the supermatrix for the network component, C is the supermatrix for the hierarchical component and A is the matrix connecting the network to the hierarchical component. Then it uses the calculus type calculation to calculate $limitB$ and $limitC$. It returns the following as the limit matrix

$$\begin{bmatrix} limitB & 0 \\ A \times limitB + C \times A & limitC \end{bmatrix}.$$

It uses the sum formula to calculate the limit of a pure hierarchy.

New hierarchy (with limit): This is the same as the New hierarchy without limit, except the the lower left hand corner is calculated repeatedly until it converges.

Identity at sinks: This is the standard solution for handling hierarchies and sinks in general as outlined in [1]. That is, all sinks are given self-loops and the standard limit matrix calculation is applied.

Sinks formula (straight normalize): This calculation breaks up the matrix into two pieces, the sinks and the rest of the matrix. In that way the matrix can be written in this way

$$\begin{bmatrix} B & 0 \\ A & 0 \end{bmatrix}$$

where B is the matrix for the non-sink nodes and A is the matrix connecting the non-sink nodes to the sink nodes. The limit matrix returned is based upon the matrix

$$\begin{bmatrix} limitB & 0 \\ A \times limitB & 0 \end{bmatrix}$$

where $limitB$ is the limit of the matrix B calculated using the calculus type calculation. The only additional step is that the columns are renormalized.

Sinks formula (normalize limitB): This is the same as the previous formula except that only the *limitB* portion of the returned matrix is rescaled (column by column) so that the entire result adds to one on each column. The point of this difference is to keep the bottom left corner with the same scaling it originally had.

Pre-2001: This calculation is kept for historical purposes only, but should not be used.

Pre-2000: This calculation is kept for historical purposes only, but should not be used.

The above algorithmic choices for limit matrix calculation in SuperDecisions address the complexities of calculating a limit matrix when one is not willing to add self-loops to sinks. We believe adding self-loops to sinks is problematic on two fronts. First, it is adding connection structure to a model that the user did not originally intend. Secondly it gives raise to calculations where all the priorities flow to a single node, even though the original model gave no indication that this would occur.

In order to address the problem of sink nodes (without adding self-loops) there are two types of algorithms employed by SuperDecisions. First is the rescaling approach (which tests for convergence of the powers of the scaled supermatrix by first rescaling them in some way). The second approach is to decompose the scaled supermatrix into simpler pieces, compute on those pieces, and finally combine the results.

References

[1] T.L. Saaty. *Decision Making with Dependence and Feedback The Analytic Network Process.* RWS Publications, Pittsburgh, 1996.

[2] T.L. Saaty. *Fundamentals of the Analytic Hierarchy Process.* RWS Publications, 4922 Ellsworth Avenue, Pittsburgh, PA 15413, 2000.