MWSUG 2017 - Paper BF02

Computing Risk Measures for Loan Facilities with Multiple Lines of Draws

Chaoxian Cai, BMO Harris Bank, Chicago, IL

ABSTRACT

In commercial lending, a commitment facility may have multiple lines of draws with hierarchical loan structures. The risk measures for main, limit, and sublimit commitments are usually aggregated and reported at the main obligation level. Thus finding all hierarchical loan structures in loan and commitment tables is required in order to aggregate the risk measures. In this paper, I will give a brief introduction of commercial loans from simple standalone loans to revolving and non-revolving commitments with complex loan structures. I will present a SAS® macro program, which can be used to identify main obligations and loan structures of future commitments from loan and commitment relational tables using Base SAS DATA steps. Risk measures, such as exposure at default (EAD) and credit conversion factor (CCF), are computed for these complicated loans and illustrated by examples.

INTRODUCTION

In business intelligence, a measure is an aggregable numerical value that an organization is using to monitor its business. A measure needs to be computed from the data element level, and can be either rolled up or sliced regarding to observation dimensions. In the risk management for retail and commercial loans, the risk measures include outstanding loan balance, unused commitment amount, probability of default (PD), loss give default (LGD), exposure at default (EAD), or any other risk drivers which are of interest and importance to the business. For retail loans with simple loan structures, most risk measures could be computed straightforwardly. However, for commercial loans, the loans may have complicated deal structures that need to be identified first before any meaningful calculations. A commitment facility may have multiple lines of draws with hierarchical loan structures. The risk measures for main, limit, and sublimit commitments are usually aggregated and reported at the main obligation level. Tree data structures are commonly used to model data with hierarchical information. However, SAS data sets are relational tables with rows and columns and cannot store tree data structures directly; the hierarchical connections between two objects have to be stored as column attributes in a SAS data set. Therefore, the hierarchical data structures are often hidden and embedded in a SAS data set. Finding all hierarchical loan structures in loan and commitment tables is required in order to compute the risk measures.

In this paper, I will give a brief introduction of commercial loans from simple standalone loans to revolving and non-revolving commitments with complex loan structures. I will present a SAS macro program, which can be used to identify main obligations and loan structures of future commitments from loan and commitment relational tables using only Base SAS DATA steps. Risk measures, such as exposure at default (EAD) and credit conversion factor (CCF), are computed for these complicated loans and illustrated by examples.

BASIC LOAN STRUCTURES

In order to compute risk measures correctly, we better to have some basic knowledge about common loan structures. While most people are familiar with mortgage and credit card, which are common retail banking loans, there are fewer people specialized in commercial lending or having life experience with commercial loans. Figure 1 shows four basic types of loan structures that a borrower may experience when the borrower obtains a loan from a lender.

Case 1 is the type of standalone loans, such as mortgages, vehicle loans, small business term loans, small business demand loans, which are very common in retail banking. This type of loans is close end loans, and does not have commitment set up for the borrower. Cases 2, 3, and 4 are the types of loans having future commitments set up for the borrower. Case 2 is the type of revolving single line of credit, such as personal credit card, home equity line of credit, small business line of credit, and this type of loans is also common in retail banking. In this case, usually the same account is set up for both commitment and drawn loan. Case 3 is non-revolving single letter of credit, such as executive letter of

credit and financial standby letter of credit. In this case, the commitment account may be same or different from the drawn account depending on how the lender set up the accounts. If the same account is set up for both the commitment and the drawn loan, it appears that a commitment facility has been converted to a loan after the drawn loan is taken. If different accounts are set up for the commitment and the drawn loan, Case 3 is a simple scenario in Case 4 with only a single draw and one level of commitment. Case 4 is the type of loans with multiple lines of drawn loans or takedown loans. Commercial lines of credit and commercial term loans are usually structured with hierarchical main/limit/sublimit commitment accounts, and the loans are drawn from the commitment account along the lines.

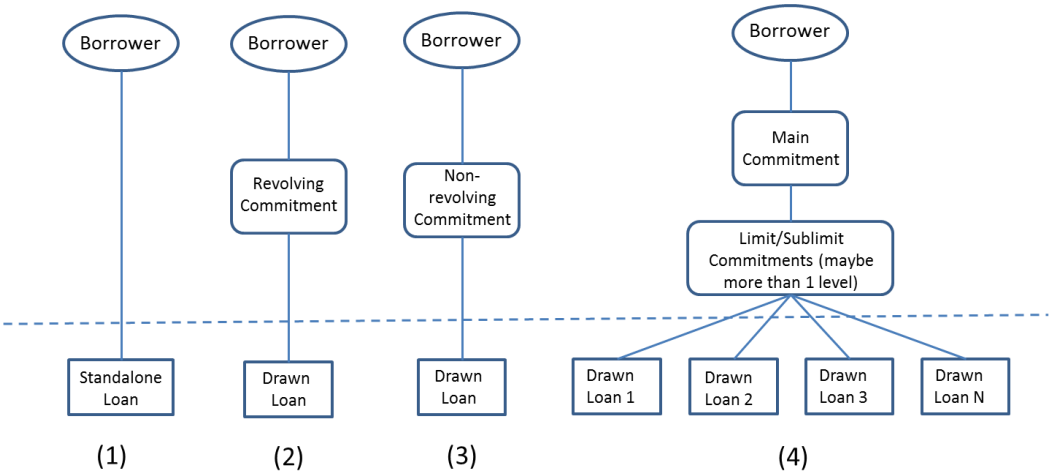

Figure 1. Basic types of loan structures.

The borrowers' authorized commitment amount and unused commitment amount need to be computed from the main commitment level. The future unused commitment amount is a bank's exposure off the balance sheet while standalone loans and takedown loans are current and realized on the balance sheet. Accordingly, in risk modeling, PD/LGD models are based on data sets from current loans (standalone and drawn loans) while EAD model is only applied to future commitment accounts. For example, the credit conversion factor (CCF) is not applicable to the standalone loans in Case 1, CCF is either 0 or 1 in Case 3 which the commitment is non-revolving and only has a single draw, and CCF is between 0 and 1 in Case 2 and Case 4 which have either revolving commitments or multiple lines of draws.

In commercial lending, the loans may have been structured in any of the above forms. In addition, for large commercial loans, there are scenarios that a large commercial commitment or loan may be shared or financed by several lenders which involve in loan sales, participations, and/or syndications, resulting in more complicated lending deal structures.

EXAMPLE

Standalone loans and single line of credit loans (Cases 1 to 3 in Figure 1) are simple loan structures, and we can compute risk measures for these loans without much of difficulty. However, in commercial lending, lending facilities include often current loans and future commitments with several levels of hierarchies. Figure 2 shows some examples of expanded loan structures of Case 4. In Case (4a), there is only a single takedown loan from a main commitment; in Case (4b), there are two takedown loans withdrawing from a limit commitment which is a line of credit from a main commitment; in Case (4c), both limit and sublimit lines of commitments have been set up by the lender. Each line can finance takedown loans not exceeding its line limit. In such cases, risk measures such as unused commitment amount and outstanding balance are aggregated and reported at the highest advised level (main commitment) while commitment fees may need to be percolated down and proportionally applied to current drawn loans.

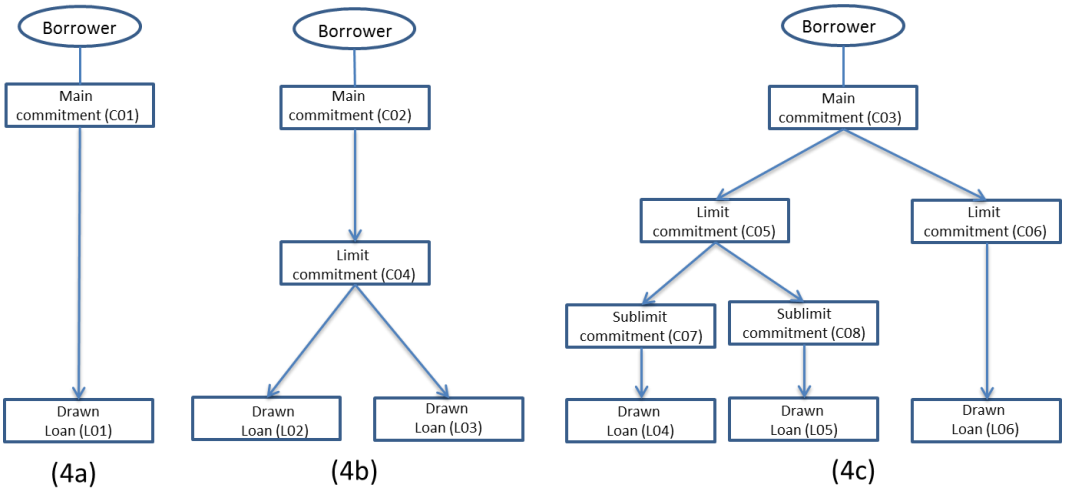

Figure 2. Loan structures with future commitments and drawn loans.

These loan structures are stored in the banks' loan and commitment tables with each obligation having a row entry. In such a table, let *AccountID* identify the child obligation which is either a loan (*Loan_ID*) or a commitment (*Commitment_ID*), and let *parentAccountID* be a self-referenced *AccoutID* that identify the parent obligation from which the child obligation draws loans. Table 1 shows a conceived example that represents the drawn loans L01 to L06 and the future commitments C01 to C08 in Figure 2. Some variables of interest, such as credit limit, outstanding balance, and disbursed amount, are included in the table to illustrate how to calculate the actual exposure at default and the credit conversion factor for this example in the next section.

Obs	*Account ID*	*Parent Account ID*	*Credit Limit*	*Disbursed Amount ($, at start time t0)*	*Disbursed Amount ($, at end time t1)*	*Outstanding Balance ($, at start time t0)*	*Outstanding Balance ($, at end time t1)*
1	L01	C01	0	0	0	20	10
2	L02	C04	0	0	0	15	10
3	L03	C04	0	0	0		25
4	L04	C07	0	0	0	40	35
5	L05	C08	0	0	0		290
6	L06	C06	0	0	0	35	30
7	C01		100	30	30	0	0
8	C02		100	20	50	0	0
9	C03		500	100	400	0	0
10	C04	C02	80	20	50	0	0
11	C05	C03	500	50	350	0	0
12	C06	C03	300	50	50	0	0
13	C07	C05	300	50	50	0	0
14	C08	C05	400	0	300	0	0

Table 1. An example of child-parent hierarchy (child: *AccountID*, parent: *parentAccountID*).

The hierarchical loan structures can be read into a SAS data set. The following code is used to import the example in Table 1 into a SAS data set:

```
data loantable;
input AccountID $1-3 parentAccountID $5-7 Credit_Limit 9-11
Disbursed Amount t0 13-15
      Disbursed_Amount_t1 17-19 Outstanding_Balance_t0 21-22
Outstanding_Balance_t1
```

```
      24-26;
datalines;
L01 C01   0   0   0 20  10
L02 C04   0   0   0 15  10
L03 C04   0   0   0     25
L04 C07   0   0   0 40  35
L05 C08   0   0   0    290
L06 C06   0   0   0 35  30
C01     100  30  30  0   0
C02     100  20  50  0   0
C03     500 100 400  0   0
C04 C02  80  20  50  0   0
C05 C03 500  50 350  0   0
C06 C03 300  50  50  0   0
C07 C05 300  50  50  0   0
C08 C05 400   0 300  0   0
;
```

The SAS data set *loantable* includes a few columns that we will use to illustrate the calculations of exposures at default and credit conversion factors at the loan account level. In a bank's loan portfolio, there will be thousands or millions of such rows in the loan and commitment table.

To calculate risk exposures at the highest level of future commitment, we need to find all hierarchical loan structures from the loan/commitment table and identify the highest level of future commitment. A hierarchical structure is a special type of graph with a unidirectional connection. Several methods can be used to find the hierarchical structure in Table 1. One method is just to apply a brute force tracking up along the parent path and create data set for each parent level commitment account. This method is feasible since, in practice, the hierarchical levels are no more than 5 levels above the drawn loans. A second approach is to perform a SQL query on a tree data structure by writing recursive SQL procedures. SAS PROC SQL supports recursive joins that can be programmed to query hierarchies in a SAS data set. In this paper, I will present a modified Union-Find algorithm to find all these hierarchical structures from a SAS data set. This method is good for drawn loans with any levels of future commitments.

The Union-Find algorithm is a classic graph algorithm used to join disjoined sets and find connected components of a graph from a given set of vertices and edges (Weiss, 1994; Cormen, 1997). The algorithm requires tree data structures to store connected components. Since SAS does not have tree-type data structures, the implementation of Union-Find algorithm requires an implicit set up of tree structures using one-dimensional array. Previously, I had presented a paper on the title of "Implementing Union-Find Algorithm with Base SAS DATA Steps and Macro Functions", where I gave a detailed explanation of the algorithm, how the algorithm works, and the programming techniques used to implement the algorithm (Cai, 2015). In this paper, I have wrapped the implementation of Union-Find algorithm in a macro function named *%group_connected_components*. The complete macro program is listed in APPENDIX A. Following are the macro parameters and major steps inside the macro function:

```
%macro group_connected_components(edgelist=, vx=, vy=, elistout=elistout,
vlistout=vlistout, hierarchy=TRUE);

/* Step 1: a) Create vertex list from the input edge list;
           b) Label each vertex sequentially from 1 to N;
           c) Crete new edge list with each vertex labeled.
*/

/* Step 2: a) Process edge list with Union-Find algorithm;
           b) Output connected components.
*/

/* Step 3: a) Output updated vertex list with connected components IDs;
           b) Output updated edge list with connected components IDs;
```

```
*/

%mend group_connected_components;
```

The input data is an edge list data set (*&edgelist*) which has vertices defined by two columns (*&vx* and *&vy*); the data set *&elistout* is the output edge list with unique *ConcompID* assigned for each connected component, and the data set *&vlistout* is a vertex list with each vertex (object) identified by *ObjectID* and has row entries of *NodeID* and *ConcompID*. The data set can be set up as a hash table to find the connect component *ConcompID* for each vertex *NodeID*. The macro function includes the following steps: 1) create new vertex list (*vlist*) and edge list (*elist*) and label each vertex with a unique sequential number (*NodeID*), 2) process the edge list (*elist*) and apply the Union-Find algorithm to build connect components, and 3) output the resulted vertex list (*&vlistout*) and edge list (*&elistout*). There are three basic set operations in the Union-Find algorithm: making disjoint sets, finding disjoint sets, and merging disjoint sets. Initially, each vertex is treated as a disjoint set, and *N* disjoint sets are set up at the beginning for *N* numbers of vertices. In each DATA step iteration, an edge that links two vertices (*&vx* and *&vy*) is read, and the disjoined sets that contain the vertices *&vx* and *&vy* are found and merged. The DATA step loop continues until all edges have been processed.

The Union-Find algorithm is for undirected graph. In APPENDIX A, the macro function provides an option to employ union-by-depth optimization to improve the run time efficiency. However, if the connected components have hierarchical structures, the union-by-depth operations will break the hierarchical relation between a child node and its parent node. Since we need to preserve the hierarchical structures, we will not apply the union-by-depth optimization when two disjoint sets are joined. In this case, we will modify the algorithm to let the child node always point to the parent node when two subtrees are merged together. In the macro function, `hierarchy=TRUE` is set as default, which will skip the union-by-depth operations. Setting `hierarchy=TRUE` will use the union operations without union-by-depth optimization and preserve tree hierarchies of all connected components.

Now we can use the macro function *%group_connected_components()* to solve the problem in the example. We first create the edge list *linkednodelist* from the loan/commitment table, and then using *linkednodelist* as the input, we invoke the macro function *%group_connected_components()*.

```
data linkednodelist;
    set loantable (keep=AccountID parentAccountID);
    where parentAccountID is not missing;
run;

%group_connected_components(edgelist=linkednodelist, vx=AccountID,
vy=parentAccountID, hierarchy=TRUE)
proc print data=vlistout; run;
```

Table 2 is the resulted vertex list output, where each vertex (object) is identified by an *ObjectID*, and each *ObjectID* is uniquely identified by a *NodeID*. The *ConcompID* is the *NodeID* of the root of each loan structure. Note that three hierarchical loan structures are discovered: {C01, L01}, {C02, C04, L02, L03}, and {C03, C05, C06, C07, C08, L04, L05, L06}. These disjoint sets are identified by *ConcompID* 1, 2, and 3, respectively.

Obs	*ObjectID*	*NodeID*	*ConcompID*
1	C01	1	1
2	C02	2	2
3	C03	3	3
4	C04	4	2
5	C05	5	3
6	C06	6	3
7	C07	7	3
8	C08	8	3

9	L01	9	1
10	L02	10	2
11	L03	11	2
12	L04	12	3
13	L05	13	3
14	L06	14	3

Table 2. The vertex list output.

Assuming all commitments in the data set *loantable* are in default status and the data have been collected within a data collection window, typically a 12-month period, starting at time *t0* and ending at the default time *t1*. For a main obligation, the actual/realized exposure at default (EAD) is the sum of outstanding balances of all takedown loans under the main obligation at the default time:

EAD = sum(Outstanding_Balance_t1).

The credit conversion factor (CCF) can be computed with the following formula:

CCF = [EAD – sum(Outstanding_Balance_t0)]/Unused_Commitment_Amount_t0,

where *Unused_Commitment_Amount_t0 = Credit_Limit – Disbursed_Amount_t0*, and *Disbursed_Amount_t0* is the disbursed amount at the start of data collection window and *Credit_Limit* is the advised credit limit of a commitment at the start of data collection window (Brown, 2014; Yang and Tkachenko, 2014). After all main obligations and their loan structures have been identified, the computations of EAD and CCF for a future commitment are straightforward. The following SAS code is used to accomplish the task. The results are listed in Table 3.

```
proc sql noprint;
create table loantable2 as
select a.*,
       sum(a.Outstanding_Balance_t1) as groupOS_t1,
       sum(a.Outstanding_Balance_t0) as groupOS_t0,
       b.ConcompID
from loantable as a
left join vlistout as b
on a.AccountID=b.ObjectID
group by ConcompID;
quit;

data loantable3;
   set loantable2;
   if parentAccountID=' ' or substr(AccountID, 1, 1)='L';
   if substr(AccountID, 1, 1)='L' then do;
         EAD=Outstanding_Balance_t1;
         CCF=0;
   end;
   else do;
          EAD=groupOS_t1;
         CCF=(EAD-groupOS_t0)/(Credit_Limit-Disbursed_Amount_t0);
   end;
   if CCF ne . and CCF<0 then CCF=0;
   if CCF ne . and CCF>1 then CCF=1;
   drop ConcompID;
run;

proc sort data=loantable3;
by AccountID;
run;
```

```
proc print data=loantable3;
run;
```

Obs	AccountID	Parent AccountID	Credit_ Limit	Disbursed_ Amount_t0	Disbursed_ Amount_t1	Outstanding_ Balance_t0	Outstanding_ Balance_t1	EAD	CCF
1	C01		100	30	30	0	0	10	0.00
2	C02		100	20	50	0	0	35	0.25
3	C03		500	100	400	0	0	355	0.70
4	L01	C01	0	0	0	20	10	10	0.00
5	L02	C04	0	0	0	15	10	10	0.00
6	L03	C04	0	0	0	.	25	25	0.00
7	L04	C07	0	0	0	40	35	35	0.00
8	L05	C08	0	0	0	.	290	290	0.00
9	L06	C06	0	0	0	35	30	30	0.00

Table 3. Loan/Commitment table with actual EAD and CCF.

CONCLUSION

In summary, I have introduced some basic concepts in loan structures and a SAS macro program that can be used to identify the hierarchical loan structures from a SAS data set. I have used a simple example to demonstrate how to apply the macro program to find these complicated loan structures. The computations of exposure at default and credit conversion factor have been illustrated with example. The algorithm and the macro program presented in this paper are well scalable. It can be used to find loan structures in a relational table with thousands to millions of rows with practically linear processing time. The application of this macro program is not just limited to the problem discussed herein. It may be extended to solve other problems that involve hierarchies, trees, networks, and connected components.

REFERENCES

1. Weiss, M. A. (1994). *Data Structures and Algorithm Analysis in C++*. Menlo Park, CA: Addison-Wesley Publishing Company.
2. Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1997). *Introduction to Algorithms*. Cambridge, MA: The MIT Press.
3. Cai, C. (2015). "Implementing Union-Find Algorithm with BASE SAS DATA Steps and Macro Functions". PharmaSUG 2015 Conference proceedings. Available at http://www.pharmasug.org/proceedings/2015/BB/PharmaSUG-2015-BB06.pdf.
4. Brown, I. L. J. (2014). *Developing Credit Risk Models Using SAS® Enterprise Miner™ and SAS/STAT®: Theory and Applications*. Cary, NC: SAS Institute Inc.
5. Yang, B. H., Tkachenko, M. (2014). "Modeling of EAD and LGD: Empirical Approaches and Technical Implementation". Available at http://mpra.ub.uni-muenchen.de/57298/.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Chaoxian Cai, Ph.D.
Enterprise: BMO Harris Bank.
Address: 111 West Monroe Street
City, State ZIP: Chicago, IL 60603
E-mail: cai.charles.x@gmail.com

Other brand and product names are trademarks of their respective companies.

The content presented in this paper is solely for presenting programming concepts and the content herein is not a representation of the opinion and practice of the author's associated employer.

APPENDIX A

The following is the SAS macro program discussed in the context:

```
%macro group_connected_components(edgelist=, vx=, vy=, elistout=elistout,
vlistout=vlistout, hierarchy=TRUE);
/*************************************************************************
Program: group_connected_components.sas
Purpose: For a given undirected graph G(V, E), find all connected components
Author: Chaoxian Cai, 5/15/2016

Parameters:
edgelist: The input edge list data set with edges defined by vx and vy
vx: Left vertex of an edge; for hierarchical tree structure, the child node
vy: Right vertex of an edge; for hierarchical tree structure, the parent node
hierarchy: Whether to preserve hierarchical structures in the connected
    components, TRUE or FALSE. Default is set to TRUE.
elistout: The output edge list data set with each connected component labeled
    by a unique identification number (ConcompID). Default name: elistout.
vlistout: The output vertex list data set with each vertex (object)
    identified by ObjectID, NodeID, and ConcompID. Default name: vlistout.

Notes:
Union-Find algorithm is implemented to find the connected components. By
default, union by depth optimization is applied to shorten tree depth. As a
result, the connected components do not preserve hierarchical tree
structures; If hierarchy=TRUE is set, then union by depth is not applied. The
union of two disjoint sets preserves child-parent hierarchy, and the resulted
connected components have hierarchical tree structures. The connected
components are identified by their root node IDs.
*************************************************************************/

/* create vertex list from given edge list */
proc sql noprint;
create table vlist as
select &vx as ObjectID
from &edgelist
union
select &vy as ObjectID
from &edgelist
order by ObjectID;
quit;

/* label each vertex sequentially from 1 to N */
data vlist;
    set vlist end=last;
      NodeID + 1;
      if last then call symputx('nNode', NodeID);
run;

%put nNode=&nNode;

/* attach labels to each node in the edge list */
```

```
proc sql noprint;
create table elist as
select a.*,
       b.NodeID as leftnode,
         c.NodeID as rightnode
from &edgelist as a
left join vlist as b
on a.&vx=b.ObjectID
left join vlist as c
on a.&vy=c.ObjectID;
quit;

/* apply Union-Find algorithm */
data ConcompID (keep=NodeID ConcompID);
array p[&nNode] _temporary_;
set elist (keep=leftnode rightnode
           rename=(leftnode=x rightnode=y)) end=last;
if _n_=1 then do;
    do i=1 to &nNode;
          p[i]=0;
      end;
end;

/* find current root of node x */
do while (p[x]>0);
    x=p[x];
end;

/* find current root of node y */
do while (p[y]>0);
    y=p[y];
end;

/* union subtrees of x and y */
%if %upcase(&hierarchy)=TRUE %then %do;
if (x ne y) then p[x]=y;
%end;
%else %do;
if (x ne y) then do;
    if (p[y]<p[x]) then p[x]=y;
      else do;
          if (p[y]=p[x]) then p[x]=p[x]-1;
            p[y]=x;
    end;
end;
%end;

/* output connected components */
if last then do;
    do i=1 to &nNode;
          NodeID=i;
            root=i;
            do while (p[root]>0);
                root=p[root];
            end;
            ConcompID=root;
            output ConcompID;
```

```
      end;
end;
run;

/* create vertex list output data set */
data &vlistout (keep=ObjectID NodeID ConcompID);
merge vlist (in=a) ConcompID;
by NodeID;
if a;
run;

/* create edge list output data set */
data &elistout;
    if 0 then set &vlistout (keep=ObjectID ConcompID);
      if _N_=1 then do;
          declare hash gidhash (dataset: "&vlistout");
          gidhash.definekey("ObjectID");
          gidhash.definedata("ConcompID");
          gidhash.definedone();
      end;

      set &edgelist;
      if gidhash.find(key: &vx)=0 then do;
      end;

      drop ObjectID;
run;

%mend group_connected_components;
```