**Replication Package for Causal Analysis**

**Created February 9, 2022**

The starting point for preparing a dataset for causal discovery is this dataset (CSV file) produced by the issue-tracking and mailing-list scraping tools we were using; henceforth called the “scraping tool.” This copy of the dataset is dated August 13, 2020: **openssl\_social\_smells\_timeline.csv.**

Here is a Data Dictionary explaining the variables in the dataset; both those variables (1) included in the initial dataset produced by the scraping tool; or (2) newly derived variables for input to causal discovery. While the second column, Definition, duplicates material already covered better elsewhere, note that for any time periods when there were zero commits, these definitions imply that many of the measures should be zero; and thus it is safe to impute zero for this particular source of missing values (what is called MD Reason 2 further below).

|  |  |  |
| --- | --- | --- |
| **Variable** (*column*) | **Definition** | **How treated in preparing for and conducting causal discovery** |
| cve\_id (*A*) | The identifier of the CVE being remediated within the OpenSSL project, which maintains the Secure Socket Layer code critical to the Chromium Project; as described in our article.  Figure 1 depicts overall timeline for a single CVE’s remediation. | In the original dataset, cve\_id was represented as a string. Converted to an integer due to a limitation in what datatypes are supported by the software packages we were using (including Tetrad).  Detail: the earliest cve is from year 2006, and so the “20” is redundant, so concatenate the last two digits of the year with the last four digits of the cve\_id and convert into an integer. The specific Excel formula used was “=NUMBERVALUE(CONCAT(LEFT(RIGHT(cve\_id,7),2),RIGHT(cve\_id,4)))”. The resulting conversion is an injection: records with the same unique cve\_id are mapped to the same unique integer.  How used: later, cve\_id is converted into a set of dichotomous indicator variables, one per cve\_id; and the variable itself was deleted. This conversion allows searching the entire dataset as a single CSV file (rather than one per cve\_id). While making the application of causal discovery much simpler, there is a downside to this approach, in that it presumes that there is a single underlying causal model relating sociotechnical measures to outcomes within consecutive time periods for CVE remediation. However, the extent to which such a common-across-CVEs structural causal model exists is also concurrently tested by this inclusion of the dichotomous CVE indicators in the dataset being searched; and seeing whether and how many edges form between a CVE dichotomous indicator. So, in what follows, we pay particular attention to how many such edges form between particular CVEs and any sociotechnical or outcome variables. |
| commit\_interval (activity\_0: *B*) (activity\_2: *C*) | Concatenation of the hashes of the first and last instances of CVE-related activity (especially commits) during the time period. If there is only one commit within the 3-month period associated with a given CVE remediation, then the hashes will be identical. If there are none, then the field will just be blank. | Replaced with indicator variables:   * **activity\_0** (equals 1 if and only if zero hashes appear) and * **activity\_2** (equals 1 if and only if two hashes appear)   Later, both indicator variables were removed following their use in preparation of the dataset for causal discovery, given their redundancy.  Detail: variable commit\_interval is the concatenation of two 40-character hashes separated by a hyphen; or blank if there are no commits to form hashes from. Here are the Excel formulas used to derive the two activity indicator variables:   * **activity\_0** - used “=IF(commit\_interval="",1,0)” * **activity\_2** - used “=IF(AND(NOT(0\_act),LEFT(commit\_interval,40)<> RIGHT(commit\_interval,40)),1,0)” |
| start\_day (*D*) | First day of the time period | Initially, renamed “Start” for brevity.  Replaced with date ordinal value because our causal discovery tool doesn’t support the datetime datatype. The conversion was achieved by using an Excel formula: “=datevalue(left(start\_day,10))” |
| end\_day | 90 days after the start\_day | Deleted due to deterministic relationship with start\_day |
| (3 Communication smells-related variables) | The next three variables (org\_silo, missing\_links, and radio\_silence) represent “communication smells” and are derived in a similar way: the scraping tool we used builds two graphs: Graph A for who posts and who responds, Graph B for relationships between code units, and then maps Graph A into Graph B (developer D worked on code C).  All three communication smells are counts of particular features of the connectivity among these graphs. (The main text of the article provides a more precise explanation for radio\_silence [silence].) | |
| org\_silo (E) | Count of “the number of single collaborations between different developers in which at least one of them does not participate in the [appropriate] communication channel” [14] for the current time period | No change to variable name |
| missing\_links (F) | Count of instances where two people were working on same file within the time period, but there’s no evidence of communication from looking at issue tracker and mailing list. | Renamed “mis\_link” for brevity |
| radio\_silence (G) | Count of delays due to lack of direct communication between two sub-communities engaged in a CVE’s remediation. | Renamed “silence” for brevity |
| primma\_donna | (Not yet fully operationalized at the time of our analysis and thus dropped from further consideration.) | Deleted |
| st\_congruence (H) | Short for “Socio-technical congruence.” A single **continuous measure** of overall similarity between collaboration graph and the file graph. **Scale** from **0..1** (1 means perfect congruence). | Renamed “congruence” for brevity |
| Communicability (I) | **A continuous measure** of the diffusion of information. How likely is it that an architectural decision is known by the set of developers that need to know about it. **Scale** from **0..1** (1 means perfect communicability). Is the inverse of incommunicability. | Renamed “communicate” for brevity |
| code\_only\_devs (J) | Count of coders who made commits to the implicated CVE-related files during the time period. (For the CVE identified in column A.) | Renamed “code\_dev” for brevity |
| code\_files (K) | Count of files involved in CVE’s remediation. | Renamed “file” for brevity |
| ml\_only\_devs (L) | Count of people who sent at least one item to mailing list during time period. Not necessarily on topic. | Renamed “mail\_dev” for brevity |
| ml\_threads (M) | Count of email threads in ML during that time window on any topic. | Renamed “thread” for brevity |
| n\_commits (N) | Count of commits (there can be multiple commits for multiple files) during time period | Renamed “commit” for brevity |
| sum\_churn (O) | Count of Lines of Code (LOC) committed to the code file counted in code\_files; summed over the commits in n\_commits. | Renamed “churn” for brevity |
| (next time period-related variables) | The final few variables in this table are a repeat of the 11 variables assigned columns E-O above but instead pertain to the next time period (90 days) in a CVE’s remediation; and their names match the 11 variables above but have a suffix “2” appended to their names.  These 11 new variables represent what happened in the next time period that could (in principle) be partly attributed to what happened in the current time period for the CVE identified in column A, or to idiosyncrasies of the CVE itself (e.g., whose resolution requires a skillset not present in coders involved in its remediation).  In terms of manipulating the dataset, each of these next-time period-related variables is derived in this way: make a copy of the corresponding original variable and shift all of its values up by one row. Of course, the top value of each original-variable column is simply unused; and the bottom value of the new column is blank.  Finally, for each CVE id, the record representing the last time period for a CVE’s remediation is then deleted (of course, it had gotten either the values of the first time period of the next CVE, or blanks, if it was the last row of the entire dataset).  The variables appear in the same order as the corresponding original 11 variables above: org\_silo2, mis\_link2, silence2, congruence2, communicate2, code\_dev2, file2, mail\_dev2, thread2, commit2, churn2. | |
| org\_silo2 (P) | Same count as “org\_silo” but applied to the next time period | |
| mis\_link2 (Q) | Same count as “mis\_link” but applied to the next time period | |
| silence2 (R) | Same count as “silence” but applied to the next time period | |
| congruence2 (S) | Same 0..1 measure as “congruence” but applied to the next time period | |
| communicate2 (T) | Same 0..1 measure as “communicate” but applied to the next time period | |
| code\_dev2 (U) | Same count as “code\_dev” but applied to the next time period | |
| file2 (V) | Same count as “file” but applied to the next time period | |
| mail\_dev2 (W) | Same count as “mail\_dev” but applied to the next time period | |
| thread2 (X) | Same count as “thread” but applied to the next time period | |
| commit2 (Y) | Same count as “commit” but applied to the next time period | |
| churn2 (Z) | Same count as “churn” but applied to the next time period | |

Note the details of how the conversions mentioned in the third column above are achieved won’t matter as long as certain properties of the variables are retained: in particular, the cve\_id conversion need only be an injection (one-to-one); the conversion of start-day to an integer can be any standard datetime conversion to integer (as long as it is linear and monotonic; as many such conversions are).

The original dataset had 6697 rows. We reference this fact in what follows.

**Handling missing data (MD)**

The dataset had many variables that were frequently blank. An analysis with Mplus® [32] returned a summary that looked something like the following table. (Only the variable names have been changed: Mplus only recognizes the first eight characters of a variable name, and so, the variable names when specified in Mplus were shortened; however, to avoid confusing the reader the variable names appearing in the resulting table were then restored back to match those used in the table above.)

SUMMARY OF DATA

Number of missing data patterns 4

SUMMARY OF MISSING DATA PATTERNS

MISSING DATA PATTERNS (x = not missing)

1 2 3 4

cve\_id x x x x

activity\_0 x x x x

activity\_2 x x x x

Start x x x x

End x x x x

org\_silo x

mis\_link x

silence x

congruence x

communicate x

code\_dev x x

file x x

mail\_dev x x

thread x x

commit x x

churn x x

MISSING DATA PATTERN FREQUENCIES

Pattern Frequency Pattern Frequency Pattern Frequency

1 3590 3 1973

2 907 4 227

Note that summing the number of cases across the four Missing Data (MD) patterns, we obtain 6697 cases, which matches the number of rows in the dataset.

From discussions with someone knowledgeable about the opensource project as well as the developer of the website scraping tool, two reasons emerged as to why data should or could be missing from the dataset:

1. The mailing log in use until about 2000-2001 was missing due to failure to preserve the original mailing log during the transition to a new mailing list manager for the project in 2001.
2. For some time periods, there are no commits. We can easily identify such cases because variable “commit” is blank (and also activity\_0 is 1 precisely when commit is blank), which prevents the scraping tool from building the file graph used for computing several measures, resulting in those variables also being blank.

A careful analysis of the dataset (using Mplus) concluded that the above two reasons explained all MD in the dataset. That is, there was no MD that we couldn’t ascribe to one of the above two reasons.

The Mplus table helps clarify the nature of the missing data. In this Replicability Package, we chose not to include the Mplus input script we developed for Mplus use, in order to not have to explain a second set of aliases for the variable names, and because we don’t consider it critical to replication, only clarifying the general nature of the missing data returned by the scraping tool.

With respect to MD Reason 1 (specified above), about 17% of the rows (MD Patterns 2 and 4 in the Mplus table above) suffered missing values due to the loss of the mailing log (907 + 227 cases). Developers were presumably communicating during the time period but there is no data on such communication.

* Specifically, this loss of the mailing log prevents computing variables “mail\_dev” and “thread” directly; and variables “org\_silo” through “communicate” indirectly (if mail activity is lost, we can’t compute one of the graphs used in computing these variables). In the rows affected by the loss of the mailing log, the scraping tool should not impute a value for all these missing values; and it does not—they are indeed blank.
* The authors felt that the conditions leading to the loss of the mailing log should have no noticeable causal effect on any of the other variables in the dataset (other than time-related variables such as “Start” and “End”). In other words, the authors deemed that data missing due to MD Reason 1 can be considered Missing Completely at Random (MCAR) [24]. Therefore, Listwise Deletion [24], that is, deleting all rows that had missing data that was MCAR, other than increasing the estimate of standard error, induces no bias. While deleting 17% of the rows is a pretty significant deletion, the authors felt that the retained 83% of the rows was still sufficient to identifying major direct causal relationships. Also, from an explainability and understandability perspective, deleting 17% of the rows seemed preferable to employing pseudo-random number generation to impute values. Finally, choosing deletion rather than imputation also helps ensure the results can be more easily replicated. (The authors did pursue Full Information Maximum Likelihood-based imputation [24] initially, but after such considerations, chose not to use the results but to instead pursue Causal Discovery on the dataset resulting from Listwise Deletion.)

With respect to MD Reason 2, when there is no commit activity within a time period, any measures of features (counts) related to commits should all be 0. While the scraping tool left blanks for their values, the value 0 should be imputed for such missing values.

* Again, the authors considered each variable one by one to verify that this was sensible. Specifically, having absolutely no commits in a time period prevents building the graphs needed for computing variables “org\_silo” through “communicate,” and when there are no commits, the scraping tool also leaves “commit” and “churn” blank. All of these commit-related variables should be zero.
* The activity\_0 variable is equal to 1 specifically in these 2200 (1973 + 227) cases.
* Note that even though there might have been no commit activity in a time period, there might still be mailing list activity and thus measures associated with any activity in the mailing list (mail\_dev and thread) could still be computed, and thus non-blank, that is, after we’ve deleted all rows associated with a lost mailing log (MD Reason 1).

Summarizing, and referring to the table generated by Mplus, we can say this about the dataset:

* Number of cases with no MD at all (corresponds to MD Pattern 1 in the Mplus table): 3590
* Number of cases with MD due to MD Reason 1 only (corresponds to MD Pattern 2 in the Mplus table): 907
* Number of cases with MD due to MD Reason 2 only (corresponds to MD Pattern 3 in the Mplus table): 1973
* Number of cases with MD due to MD Reasons 1-2 (corresponds to MD Pattern 4 in the Mplus table): 227

Thus, with the above changes to the dataset, of deleting about 17% (907 + 227) of the rows affected by the loss of an old mailing log, and imputing zeros for all remaining missing values, we thus have a dataset of 5563 cases) with no MD and can thus continue onward to the next steps in preparing the dataset for applying causal discovery.

Detail for performing Listwise Deletion: In Excel, place a Data > Filter on column “mail\_dev” and in the filter select only “(Blanks)” from the dropdown menu. The number of such rows (after the header row) should be 1134 rows. Note that this seems right because that’s the total number of cases covered by Mplus MD Patterns 2 and 4. Then, selecting all those rows (but not the header row), delete them. The result should 5563 rows.

Detail for imputing zeros: within Excel, we performed a global selection of blank cells within the entire dataset (following Listwise Deletion, and thus the only blank cells left now are due to there being no commits for that time quarter), replacing the (blank) content of all such cells by 0. In further detail: we simply invoked the Replace command configured with these options: “Find what:” empty string, “Replace with:” 0, click on Options >> to check “Match entire cell contents”, etc. 17757 replacements were thus made. Note that this is 9\*1973, the latter number is the number of cases covered by Mplus MD Pattern 3, which is the MD pattern corresponding to blank cells arising only due to there being zero commits for the time period. The 9 blank cells for each of these rows correspond to the variables indicated in the Mplus table above showing the missingness for MD Pattern 3.

**Appending Variables Representing the Next Time Period**

Append to the dataset these additional 11 variables representing the impact on the next time period: org\_silo2, mis\_link2, silence2, congruence2, communicate2, code\_dev2, file2, mail\_dev2, thread2, commit2, churn2.

* These additional variables and how they are created is explained in the data dictionary near the beginning of this section.
* Further detail: each of these 11 variables ending in “2” was created in Excel by first creating a new column having the appropriate column header and then in the column of the corresponding original variable (i.e., pertaining to the current time period), copy the values starting in row 3 through row 5563 into positions row 2 through row 5562 of that new column.
* Of course, when performing this copy and paste, the values in row 2 for the original variables are ignored/lost; while for the corresponding next time-period variables, the values associated with the last time period of any CVE’s remediation no longer make sense (or are blank in the case of the very last row) and should be deleted.
* To accomplish this deletion of the last time-period record for each CVE within Excel, create an indicator variable that indicates when a case represents the last time period for a CVE’s remediation. Detail: in row 2, insert a formula in a new column: “=IF([next-period cve\_id]=[current cve\_id],0,1)”, copy that cell and paste the formula onto the rest of the column; then copy-paste-value-only the entire column so that the formulas are replaced by their values. Then as we explained elsewhere, put a filter on that column, select only the rows with 1s (the rows that are the last cases for a CVE’s remediation) in the pull-down menu, and delete the resulting rows (i.e., the rows appearing under the header row). Then remove the filter and delete the indicator variable column.
* As there were 5563 rows and there are 121 CVEs represented in the dataset, this should leave 5442 rows (after the header row). And instead of 15 columns, there should now be 26.

**Removing Redundant Variables Representing the Current Time Period**

* When the same or similar feature is measured more than one way within a time period, it can create variables that are highly correlated to each other. Prior to applying causal discovery, we will want to combine or remove such variables from the dataset. When a pair of variables are measuring pretty much the same feature of a situation, the causal discovery algorithm we use will tend to “reward” only one of the two variables with a causal relationship with a third variable. In other words, the set of causal relationships between those two highly-correlated variables and any other variable in the dataset will be split between the two variables, making interpretation of the results of causal discovery more challenging.
* For a first example, “activity\_0,” “activity\_2,” and “commit” all gauge the amount of commit activity that occurs within a time period. Which of these three variables should we keep, or should we combine them into a new composite variable that better represents the amount of commit activity within a time period? In this case, the solution is very simple: “activity\_0” and “activity\_2” are fully determined by the number of commits (“commit”) and thus we can simply delete the less informative versions of “commit” from the dataset. In other words, we simply delete “activity\_0” and “activity\_2.”
* For a second example, consider the two variables “mail-dev” and “thread.” Both represent the amount of mail activity during a time period. Are they both measuring the same thing? Not quite. The correlation between “mail-dev” and “thread” is about 0.8, which sounds high but is still pretty far from the threshold of 0.9 [or -0.9] that we have been using in our work for determining when two variables are so highly correlated that one of them should be removed from the dataset to reduce the risk of splitting discussed above.

The above deletions leave us with 5442 rows (still) and 24 columns.

**Remove CVEs Whose Remediation Timeline is Too Short**

* Prune away any CVEs (i.e., their associated records from the dataset) having insufficient representation in our dataset for discovering casual relationships specifically for that CVE.
  + If a CVE’s remediation timeline is too short then key underlying causal relationships cannot be discovered or any that are discovered will not be as credible (more likely to be false positives). Within our dataset, seven CVEs are only represented by seven time periods or less (six, after deleting the last row of each CVE timeline as a last step of appending variables from the next time period). Discovering a structured causal model from such a small sample that faithfully represents the underlying direct causal relationships (relative to the variables collected) might be difficult. Thus, these CVEs with only seven time periods are a good candidate for deletion.
  + On the other hand, we want to be careful not to delete too many CVEs or what we learn from the remaining subset of CVEs might not adequately represent the OpenSSL Project in general.
  + Also, recollect that we deleted about 17% of the records, but these were for CVEs whose remediation timelines started before 2007. If their timeline continued well past 2007, then we’ll have plenty of time periods to analyze and for more recent time periods; but if not, then we don’t really have many time periods anyway (and none too recent) to draw useful (and timely) conclusions about the underlying causal relationships of interest; and thus the latter are also hardly bad candidates for deletion.
  + Beyond these seven CVEs, there are 114 CVEs with longer remediation timelines represented in our dataset. The shortest of these timelines includes 18 time periods (17 after the appending variables from next time period step), which may be more than sufficient to discovering any significant causal relationships (significant enough to have an observable change in outcomes of interest).
  + Thus 7 or fewer cases seems like a reasonable threshold for guiding us in which CVEs to prune from the dataset that will still leave us with a large majority of the dataset for analysis but with each retained CVE well represented in terms of the number of time periods available for analysis (at least 18).
  + Therefore, we deleted the 7 CVEs (their associated rows) with 7 or fewer time periods. Their deletion leaves us with a total of 35 fewer cases (rows). (Actually, we delete 28 rows if we do the deletion after appending variables from the next time period and deleting the last row from each CVE remediation timeline.)
  + Detail: specify headers for two new columns: CountTPs (count of time periods) and LastTPisLT8 (length of last time period is less than 8). CountTPs has value 1 in row 2 and otherwise value “=IF([next-period cve\_id]<>[current cve\_id], 1, [previous-row CountTPs+1)”; and LastTPisLT8 has value “=IF([next-row CountTPs]=1,IF([current-row CountTPs]<8,1,0),0)”. Then filtering on the latter for non-zero values, we find these cve\_ids for CVEs having rather short available remediation timelines (with parentheses hyphen inserted before the last four digits because recall that we converted the original cve\_id by concatenating only the last two digits of the year with the last four digits of the CVE id number):   
    10(-)0742 (6 TPs), 10(-)1633 (6 TPs), 16(-)6307 (3 TPs), 16(-)6305 (4 TPs), 16(-)6309 (3 TPs), 16(-)7054 (2 TPs), 19(-)1543 (4 TPs).   
    And then it’s easy to simply delete the associated (28) rows from the dataset.

The resulting dataset has 5414 rows and 24 columns; covering 114 CVEs (121-7).

**Binarize CVE\_ID**

If we want to understand the impact of sociotechnical behaviors from one time period on the next within the context of a CVE remediation timeline, we will want to treat each CVE remediation as its own recurring situation that we want to apply causal discovery to. Different CVEs involve different problems to solve, which may require different solution approaches and different individuals interacting differently to do the remediation. But within a CVE remediation timeline, we’re likely to have a more shared context for sociotechnical behaviors from one time period to impact the work activities from that and the next time periods.

There are several approaches that we can take; but here are the main three:

1. Split the dataset by CVE and apply causal discovery to each of the 114 resulting smaller CVE timeline-focused datasets to obtain a structural causal model for each; and then employ some kind of aggregation scheme to identify common causal structures.
2. Utilize one of the multi-dataset causal algorithms (e.g., IMaGES) that employs aggregation during causal discovery thereby creating a single structural causal model that represents the causal relationships that are shared across the CVE timelines.
3. For each distinct CVE, append an indicator variable, and then after doing that for all CVEs, apply causal discovery to the resulting single dataset (with 114 new columns) and make note of which variables V have a CVE-related indicator variable pointing into them and for which CVEs: these variables V have a local causal structure (i.e., “parents,” variables with a causal edge coming out of them into V) that differs specifically for those CVEs; otherwise, the variable V has a local causal structure with its parents that is shared across many (perhaps even all) CVEs; and thus can be considered possibly typical of the OpenSSL project as a whole during the years spanned by the dataset.

The challenge with the first approach is to determine which aggregation scheme to use. The challenge with the second approach is that at the time of this research, the multi-dataset algorithm used seemed to not work reliably in the particular way we were using it. Thus, circumstances pushed us towards the third approach, which required the development of code to assist in further dataset processing.

Detail of how we implemented the third approach: the basic idea is to replace the cve\_id column with one indicator variable for each distinct cve\_id value. We call this “binarizing cve\_id” because we are taking a categorical variable (cve\_id) with multiple distinct values (114 distinct values) and replacing it with 114 indicator variables, one per distinct CVE id value.

A couple technical comments before proceeding:

1. Note that we really should drop one of the 114 indicator variables because its values are inferable from the values of the other 113 indicator variables. (In other words, the sum of all 114 indicator variables for any row in the dataset must be exactly 1, and thus this imposes a constraint on the 114 indicator variables.) But as it takes 114 variables to recognize the determinism, it’s really not an issue for our use of causal discovery.
2. A reminder that any causal relationship between two variables discovered by pursuing this approach (or one of the other two approaches) might be replaced by either a chain of causal edges (if we were to add variables in the dataset that included mediators) or a fork—an inverted “V” with a third variable playing the role of apex with two causal edges coming into our two variables (if we were to add potential confounders to our dataset). Thus, any causal edges discovered might actually reflect a more complicated causal structure between source and target variables.

To facilitate binarization, we wrote a Python program: binarize-or-split-first-col.v020article.py. The program takes each distinct value (cve\_id\_number) in the “cve\_id” column of a dataset (dataset.csv) and appends a new column “b\_cve\_id\_number” to the dataset. The “b\_cve\_id\_number” column (the prefix “b\_” is short for “[Binary] indicator”) is simply an indicator variable that identifies which rows of the dataset area associated with the remediation of that particular CVE (and were assigned the same CVE id number in the cve\_id column). Then the “cve\_id” column is deleted. The program is run using Python (version 3.9) with command-line arguments as follows:

python binarize-or-split-first-col.v020article.py dataset.csv B

Here is the program listing:

# ------------------------------------------------------------------------------  
# Input a CSV file and output either:  
# (1) an extended version of the same file that for each distinct value J  
# in the first column has a corresponding binary indicator variable added  
# as a new column appended at the horizontal end of the original dataset,  
# whose name is "b\_J"  
# (2) splits the dataset into multiple datasets according to the value in first  
# column, which itself is deleted, leaving sub-datasets with one less  
# column  
# ------------------------------------------------------------------------------  
# Calling arguments:  
# (1) The first argument after the name of the program must be the name of  
# the CSV file to be acted on in one of the two indicated ways.  
# (2) The second argument must indicate either "B" for binarize; or "S" for  
# split.  
# (3) If "B", this program will output a CSV file bearing the same name but  
# prefixed by "bin-". The output file includes the original variables  
# found in the input file, but after them the indicators b\_J appear, in  
# the same order as the distinct values J in column A.  
# (4) These new variables are binary variables (thus the "b\_" prefix) with  
# b\_J equaling 1 if and only if the associated case has value J in  
# column A. Otherwise it has value 0.  
# (5) If "S", this program will output a number of sub-datasets of the  
# original dataset equal to the number of distinct values in column A;  
# each bearing the same filename as the original but then the column  
# A header, the string ".equals." followed by the value from column A  
# appended whose cases it contains.  
#  
# Known limitations:  
# (1) CSV file must be in same directory as this program.  
# (2) This program works on Windows (and with adjustment, Mac)  
# Excel-created CSV files.  
# (3) Regarding column A, all cases having the same distinct value  
# in column A must be contiguous to each other.  
# (4) The program will either create an output CSV file that has the same  
# name as the input CSV file but with a "bin-" prefix; or multiple  
# CSV files whose filename is built on the original dataset's  
# filename as a prefix. Any existing file(s) of the same name will be  
# overwritten when this program is run.  
# ------------------------------------------------------------------------------  
  
  
import csv  
import sys  
import numpy as np  
  
# When running the program, call it with two arguments, the first  
# indicating the filename of the CSV file to either binarize its first  
# column or split by value in the first column (and discarding it).  
  
INPUT\_FILENAME = sys.argv[1]  
  
BINARIZE\_VS\_SPLIT = sys.argv[2] # Expecting 'B' or 'S'  
  
# Parts of the input filename that we will make use of:  
filenameLen = len(INPUT\_FILENAME)  
mainPart = INPUT\_FILENAME[0:filenameLen - 4]  
  
with open(INPUT\_FILENAME, 'r', newline='') as inputCSV:  
 # The purpose of this first code segment is to determine these  
 # attributes of the dataset:  
 # - # of headers (headerCount)  
 # - # of cases (caseCount)  
 # - list of distinct values appearing in col A, by order of  
 # occurrence (distinctValues)  
 # (Limitation: all cases with same value in col A must  
 # appear continuously to each other.)  
 # - list of the number of cases for each distinct value,  
 # by order of occurrence (cardValues)  
  
 read = csv.reader(inputCSV, dialect='excel')  
  
 caseCount = 0  
 distinctValues = []  
 cardValues = []  
  
 for row in read:  
 if caseCount == 0:  
 # exclude first column, because we won't need it  
 headerCount = len(row) - 1  
 elif caseCount == 1:  
 curVal = row[0]  
 valCount = 1  
 distinctValues.append(curVal)  
 elif row[0] != curVal:  
 cardValues.append(valCount)  
 curVal = row[0]  
 valCount = 1  
 distinctValues.append(curVal)  
 else:  
 valCount += 1  
 caseCount += 1  
 # Note that caseCount includes header, so decrement it.  
 # Also, we never appended the number of entries for last distinct value  
 caseCount -= 1  
 cardValues.append(valCount)  
  
# Next, initialize table to be a 2D-array of the right shape to hold  
# the entire file in memory.  
  
if BINARIZE\_VS\_SPLIT == 'S':  
 numColumns = headerCount  
else:  
 numColumns = headerCount + len(distinctValues)  
  
table = np.zeros(shape=(caseCount, numColumns))  
  
# Re-open the file to make a second pass through it.  
  
with open(INPUT\_FILENAME, 'r', newline='') as inputCSV:  
 # This code segment manipulates two data structures so that collectively  
 # they contain the information to be output:  
 # - headerRow - initially, just header row of dataset minus first header;  
 # but by the time we are ready to output all headers,  
 # indicator variable names (prefixed "b\_") will have been added  
 # if Binarize is selected  
 # - table - initially, will hold the original dataset, minus first col  
 # but if Binarize is selected then by the time we are ready  
 # to output all data, indicator variable values (0 or 1) will  
 # have been added  
  
 # So, first, we fill in headerRow and table from the CSV input file  
 #  
  
 read = csv.reader(inputCSV, dialect='excel')  
 rowCount = 0  
 caseID = []  
  
 for row in read:  
 if rowCount == 0:  
 headerRow = []  
 for i in range(headerCount):  
 headerRow.append(row[i + 1])  
 # we will need to record what 0-th column header was  
 caseIDHeader = row[0]  
 else:  
 table[rowCount - 1][:headerCount] = row[1:]  
 # we will need to record what 0-th column value was  
 caseID.append(row[0])  
 rowCount += 1  
  
# Second, if BINARIZE is selected, fill in the rest of the headerRow  
# with the names of the indicator variables.  
  
if BINARIZE\_VS\_SPLIT == 'B':  
 for i in range(len(distinctValues)):  
 headerRow.append('b\_' + distinctValues[i])  
  
 # Third, recall that we initialized table to be full of zeros; including  
 # the very portion past the first headerCount columns.  
 # Thus, if Binarize is selected then for each row (after first),  
 # we only need to set a single cell to 1; namely, for the b\_column  
 # corresponding to the distinct value (that was) in column A.  
  
 for i in range(caseCount):  
 groupID = int(caseID[i])  
 numColsOver = distinctValues.index(str(groupID))  
 table[i][headerCount + numColsOver] = 1  
  
 # Fourth, write out the headerRow and table to the file of name:  
 # bin-INPUT\_FILENAME  
  
 outputFilename = 'bin-' + INPUT\_FILENAME  
 with open(outputFilename, 'w', newline='') as csvOutput:  
 writer = csv.writer(csvOutput)  
 writer.writerow(headerRow)  
 for i in range(caseCount):  
 writer.writerow(table[i, :])  
  
if BINARIZE\_VS\_SPLIT == 'S':  
 # Generate a sub-dataset for each value of column A,  
 # writing out the headerRow and just those cases matching  
 # that value for column A to the file of name:  
 # INPUT\_FILENAME.columnA.b\_J  
  
 distinctValuesIndex = 0  
 currentCase = 0  
  
 while distinctValuesIndex < len(distinctValues):  
 outputFilename = INPUT\_FILENAME + '.' + caseIDHeader + '.is.' + \  
 str(distinctValues[distinctValuesIndex]) + '.csv'  
 with open(outputFilename, 'w', newline='') as csvOutput:  
 writer = csv.writer(csvOutput)  
 writer.writerow(headerRow)  
 for i in range(cardValues[distinctValuesIndex]):  
 writer.writerow(table[currentCase][:headerCount])  
 currentCase += 1  
 distinctValuesIndex += 1

The resulting dataset, whose name is as before but extended with the prefix “bin” has 5414 rows and 137 columns; covering 114 CVEs and 23 other variables, which include Start and two sets of 11 sociotechnical variables plus outcomes (one set of 11 variables each for the current time period and next time period).

**Append Null Variables to a Dataset Prior to Bootstrapped Search**

What are null variables? Null variables are copies of the variables in our dataset, whose values have been scrambled (permuted) randomly (pseudo-randomly) and independently of the other variables. If they correlate, it should be weakly at best, with the other variables in the dataset (unless the dataset is very small).

How are they used? Null variables are inserted into a dataset, prior to search. Then when we search, we do so with bootstrapping, generating many random samples of the original dataset (drawn with replacement), and then searching each. During search, we track for each edge how many times it occurs across the bootstrap samples (out of however many searches). The key idea is this: those edges among the original variables that form at a lower rate than a large proportion of edges among the null variables are likely to be false positives and we can prune (remove) all such edges, and then afterwards, we dispense (remove) all the null variables as they have served their purpose. We then report the graph (structural causal model) as well as the pruning threshold utilized (a fixed percentile against which edge frequencies are compared and only those higher than the threshold are retained).

Use of null variables with bootstrapped search are particularly useful when the dataset being analyzed is small. When dealing with smaller datasets, the evidential support for a particular claim of a direct causal relationship (or the lack of one) can be much weaker than for a larger dataset and is susceptible to small variabilities in the actual specific search executed. A small change in the values of a dataset may cause an edge that easily formed for one pattern of data values to not form for another. In other words, which edges appear as a result of search can be quite sensitive to small changes in values. Comparing edge-formation frequencies among the original variables against those among/with null variables helps us a get a sense of when we’ve gone as far as we should in “claiming” edges among the original variables; the idea being that if a particular edge frequency is not atypical for null variables, that same frequency might not be such a good indicator of their being a particular edge among the original variables. So, by evaluating the distribution of edge frequencies among/with null variables, we can get at least a qualitative idea of the sufficiency of our dataset for identifying edges among the original variables; and how sensitive their formation is to small changes in our dataset values.

In practice, we can go further, and based on our risk tolerance for occasional false positive edges, we can set a particular edge-formation frequency threshold from the distribution of edge-formation frequencies among/with null variables and employ this threshold as a filter on which edges among the original variables we should accept; and estimate the false positive rate among the edges we retained. But this is a bit complicated to do and it can be time consuming as we set the number of bootstraps pretty high to try to obtain more precise frequencies for edge formation and improve the likelihood of their replication; and doing so requires additional programming. All of this has been done, but requires more patience, care, and time for the analysis than would a much larger dataset. Having to track edge frequencies so carefully and evaluate the resulting distributions also injects more uncertainty into the analysis, more opportunities for error, and of course, complicates replication.

Fortunately, the situation is greatly simplified in the case of larger datasets; as is the case here. As datasets get larger many edges form across searches of nearly all randomly-drawn samples of the original dataset, whose edge frequencies are almost universally higher than those for null variables, whose edge frequencies fall as the dataset gets larger (the correlations they form with other variables, original and null alike, are likely to be even weaker). Then a simple criterion might be used: only retain edges that form the majority of the time in bootstrapped search. We can double check that we have a sufficiently large dataset to have broad credence to the edges that form the majority of the time by still injecting null variables prior to search, and seeing if any (or a very small number of) edges with a null variable even appear.

In our initial effort to search the dataset, we treated the dataset as small and went the more complicated route. In the results presented here, we instead report what we obtained by pursuing the more direct route at filtering which edges to retain: those that appear in the majority of bootstrapped searches. Bootstrapped FGES search even has a setting that applies this “majority decides” rule in determining which edges to retain: the Majority Ensemble Rule, and so when we apply causal discovery, it will be with this setting turned on.

With the rationale for generating null variables and employing bootstrapped search established, in the remainder of this subsection we will focus on which null variables we created and appended to our dataset.

When a dataset doesn’t have too many variables, we usually create one null variable for each existing variable; however, with 137 variables, a bootstrapped search could take a while and so might interpreting its results, and so in our situation we will generate null variables for only a subset of the original variables. Taking inventory of our variables, we have 137 variables consisting of the Start variable (the date when a time period started), the set of 11 sociotechnical and output variables for the current time period, the corresponding set of 11 variables for the next time period, and the 114 binarized variables (CVE-being-remediated indicator variables). We will create a null variable for each of the first 23 variables but only a small selection of the 114 binarized variables.

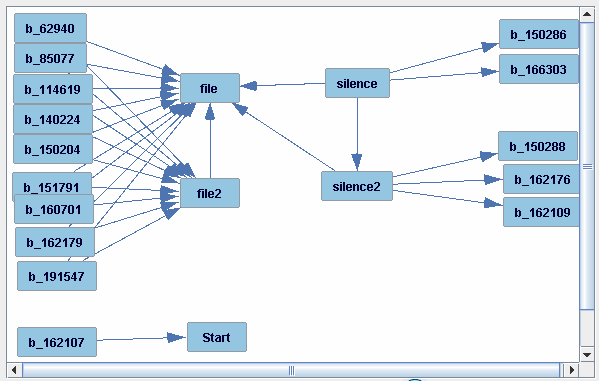
In actuality, we will run a program that generates a full set of null variables (used in the other scenario of a smaller dataset where we want careful tracking of edge frequencies among/with null variables), but then, for the binarized variables only, we only retain those null variables corresponding to the more active binarized variables; that is, one for each original variable, but then cull (remove) null variables not meeting a simple test for being “active.”

The test for which binarized variables could be considered as “active” is simple: without adding any null variables, we apply causal discovery with a reasonable set of search settings (including for bootstrapped search) on the original dataset, prune all edges that fail to appear a majority of the time, and declare a binarized variable as active if any of its remaining edges terminate on a non-binarized variable. (If a binarized variable proves not so useful from a causal perspective, meaning that there’s nothing particular about the associated CVE that influences sociotechnical or outcome variables in some idiosyncratic way, maybe it’s not such a good candidate for creation of a null variable to include in search.)

Here are the settings used for this first application of causal discovery against the 137-variable dataset (no null variables yet):

* Tetrad GUI (version): tetrad-gui-6.8.0-20200828.153437-93-launch.jar
* Knowledge box:
  + No knowledge box is used for this initial application of causal discovery. Therefore, don’t be surprised if some directed edges have a reverse orientation (what should be the edge’s source appears as its target, and conversely).
  + Later, after we’ve added null variables, we will utilize an appropriate knowledge box to reduce the probability that a given edge will be given an incorrect orientation but for this first application, it’s not so important to get the orientation correct. We’re only trying to identify an active subset of the binarized variables to guide in selecting which null variables to retain.
* Algorithm: Fast Greedy Equivalent Search (FGES) – one of the most written-about causal discovery algorithms with good precision and speed.
* Score method: Structural Equation Modelling Bayesian Information Criterion (SEMBIC)
* Penalty Discount: 2 rather than the usual default 1
  + Given the large number of cases (our dataset has over 5000 cases), we chose to be a bit more conservative regarding which binarized variables are “active” (and for the full search with null variables, later, as well)
  + A higher penalty discount results in a sparser graph because the inclusion of an edge must sufficiently improve the current graph’s score for that inclusion to happen; a larger penalty discount reduces the size of that improvement in score, making the graph grow more slowly—and sparsely—during search.
* Symmetric First Step: set to Yes
  + In the Tetrad GUI, this setting is labelled this way:
    - “Yes if the first step for FGES should do scoring for both X->Y and X<-Y”
    - In theory, for a larger dataset, this shouldn’t be necessary, but it only adds slightly to the compute time.
* Number of Bootstraps = 100
  + We’ll use a larger number of bootstrap samples when we care more about precision, but for this first run, 100 bootstrap samples drawn from the original dataset should be adequate to getting an approximate idea of which binarized variables are more active.
  + Bootstrapped search tests the “neighborhood” of the data in the dataset for sensitivity to slightly different values
* Percentage of Resample Size = 90
  + For determining how many cases each bootstrapped sample should contain relative to the full dataset
  + 90% is the newer default; 100% used to be the default, and so we mention this setting to help achieve fuller replicability in case a different version of Tetrad is being used
* Ensemble Method: Majority (2)
  + For determining which edges and frequency statistics to keep following a bootstrapped search; that is, an Ensemble Method is a way to consolidate the graphs that result from searching each bootstrapped sample of the original dataset.
  + The Tetrad GUI says this about the Ensemble Method parameter, giving three options:
    - “This parameter governs how summary graphs are generated based on graphs learned from individual bootstrap samples.
    - If “Preserved”, an edge is kept and its orientation is chosen based on the highest probability.
    - If “Highest”, an edge is kept the same way the preserved ensemble one does.
    - If “Majority”, an edge is kept…” only if it appears in the majority of bootstrap sample search results.
* Otherwise, use default settings for the search
  + Information about causal discovery (a.k.a., “search”), the Tetrad tool (Tetrad GUI), and search algorithm settings are found here: <https://cmu-phil.github.io/tetrad/manual/>.

Applying causal discovery to the original dataset configured as in the above bulleted list produces a busy search graph but if we only display those binarized variables that have formed edges with any of the non-binarized variables in the original dataset, we obtain a simpler graph that looks something like this (after moving the variables around so that the graph can be read more easily):



Note that 15 (of 114) binarized variables are shown. Here is the list of the 15 “active” binarized variables, converting their names back to something that looks more like a CVE identifier:

2006-2940, 2008-5077, 2011-4619, 2014-0224, 2015-0204, 2015-0286, 2015-0288, 2015-1791, 2016-0701, 2016-2107, 2016-2109, 2016-2176, 2016-2179, 2016-6303, 2019-1547.

But how might or should we interpret the causal graph above? The other 99 CVEs are not shown, but only 15 of 114 CVEs have any edges with non-binarized variables. These 15 CVEs seem to have some kind of near-unique causal relationship with one of these variables arising during their remediation: “Start,” “file” (or “file2”), and “silence” (or “silence2”).

* Let’s examine one of these CVE variables a bit more closely: b\_162107 (for CVE 2016-2107). It is for a CVE whose remediation consists of 17 consecutive time periods (the shortest possible—see subsection “Remove CVEs Whose Remediation Timeline is Too Short” above) that has a “Start” value that is near the maximum time period start for the entire dataset, indicating that it is one of the most recent CVE remediation timelines. Of course, for such a CVE, the shortness of its timeline and very high recency tell us a lot about what “Start” values to expect: among the highest ones. A correlation table shows this CVE id to be the one with the largest correlation (positive or negative) with the Start variable. And that’s why we see this causal relationship—it simply represents the fact that for this CVE, we can predict the “Start” value pretty precisely. From the algorithmic scoring perspective, this seems to be pretty strongly causal and thus appears in the graph. Of course, once one such edge is found, the algorithm’s scoring function tends to discount any other CVE id from having a similar relationship with “Start;” and so the search algorithm looks for elsewhere to add an edge.
* For the other 14 of 15 depicted CVE ids the situation is similar: each occupies a near extreme in terms of the values attained by “file” or “silence” and that’s why we find such edges.
* Note that for 9 of these depicted CVE ids, there are edges with both “file” and “file2”. These two variables (file and file2) are pretty strongly correlated (0.74), so perhaps it’s not surprising that if the relationship between a CVE id and one of these two variables tends to form, there’s likely a strong relationship with the other variable and an edge will form with it too. In contrast, the relationship between “silence” and “silence2” is far weaker (correlation 0.39) and so for the five CVE ids with edges forming with one of these two variables, we shouldn’t be surprised to see no edge with the other.
* Recall that we did not employ a knowledge box here and thus we imposed no user-defined constraints on the causal search, and thus we will not interpret the orientation of the edges in the above figure.

To generate such null variables, we wrote a Python program: csv-add-null.py. The program creates a copy of each variable in the dataset, randomizes the order of values within each using an appropriate pseudo-random number generator, and appends the null variable to the dataset. Then among the null variables generated from the binarized variables, retain only the null variables corresponding to the 15 active variables identified above. The other 99 (114-15) null variable-versions of the binarized variables can be deleted. The result is thus 175 (137+137-99) variables in the dataset. The program is run using Python (version 3.9) as follows:

python csv-add-null.py bin-dataset.csv

Where bin-dataset.csv is the output of the previous python program run.

Here is the program listing:

#------------------------------------------------------------------------------

# Input a CSV file and output an extended version of the same file so that

# each original variable has a corresponding null variable added as a new

# column appended at the horizontal end of the original dataset.

#------------------------------------------------------------------------------

#

# (1) When running the program, call it with only one argument:

# the filename of the CSV file to extend with nvs.

# This program will output a CSV file bearing the same name but

# prefixed by "nv-". The output file includes the original variables found in

# the input file, but after them a copy of the null variables appear, in the

# same order, but with these differences:

# (a) Each new variable has a name that is the same as its corresponding

# original variable but with "nv-" prefixed at the front of it.

# (Thus the header row is now twice as long with variables appearing in the

# sequence twice, but in the second appearance of each variable name,

# "nv-" has been prepended to the variable name.)

# (b) These new variables are called null variables (thus the "nv") and

# have the same values as their corresponding original variable, but the order

# of the values has been permuted (pseudo)randomly and independently (ideally)

# from the permutation applied to the other null variables.

#

# Known limitations:

# (1) None of the original variables should have a name beginning with

# "nv-"

# (2) CSV file should be in same directory as this program.

# (3) This program works on both Windows (and with adjustment, Mac)

# Excel-created CSV files.

# (4) The program will create an output CSV file that has the same name

# as the input CSV file but with "nv-" prepended. Any existing

# file of the same name will be overwritten when this program is run.

#

#------------------------------------------------------------------------------

import csv

import sys

import numpy as np

INPUT\_FILENAME = sys.argv[1]

with open(INPUT\_FILENAME, 'r', newline='') as inputCSV:

# The purpose of this first code segment is mostly to determine the dimensions of the input dataset.

read = csv.reader(inputCSV, dialect='excel')

rowCount = 0

for row in read:

if(rowCount==0):

headerCount = len(row)

rowCount += 1

# Thus rowCount and headerCount indicate the dimensions of the original

# dataset.

# Next, initialize table to be a 2D-array of the right shape to hold

# the entire dataset in memory. We'll need twice as many columns to make

# room for the null variables.

table = np.zeros(shape=(rowCount-1,headerCount\*2))

# Re-open the file to make a second pass through it.

with open(INPUT\_FILENAME, 'r', newline='') as inputCSV:

# This code segment manipulates two data structures so that collectively

# they contain the information to be output:

# - headerRow - initially, just the header row of the original dataset

# but by the time we are ready to output all headers,

# null variable names (prefixed "nv-") will have been added

# - table - initially, will hold the original dataset values,

# but by the time we are ready to output all data,

# null variable values will have been added

# So, first, we fill in headerRow and table from the CSV input file

#

read = csv.reader(inputCSV, dialect='excel')

rowCount = 0

for row in read:

if(rowCount==0):

headerRow = row

else:

table[rowCount-1][:headerCount] = row

rowCount += 1

# Second, fill in the rest of the header row with the names of the null

# variables.

for i in range(headerCount):

headerRow.append('nv-'+headerRow[i])

# Third, one column at a time, permute the original column's values (i.e.,

# ignore the header row) and store the permuted column of values

# headerCount columns later.

for i in range(headerCount):

table[:,headerCount+i] = np.random.permutation(table[:,i])

# Fourth, write out the headerRow and table to the file of name:

# nv-INPUT\_FILENAME

outputFilename = 'nv-'+INPUT\_FILENAME

with open(outputFilename, 'w', newline='') as csvOutput:

writer = csv.writer(csvOutput)

writer.writerow(headerRow)

for i in range(rowCount-1):

writer.writerow(table[i,:])

Then we delete the 99 nv-b\_\* null variable-versions of the 99 binarized variables among the 114 not appearing in the list of 15 active binarized variables above. The result, if explained (and followed) correctly should be that the dataset file now has 175 variables and 5414 cases (not counting the header row.

**Creating the Knowledge Box for Search (Causal Discovery)**

A Knowledge Box typically specifies which direct causal relationships are *not allowed*. For example, if two variables represent events at different points in time, we would not expect a variable representing the later event to have a direct causal relationship oriented into a variable representing the event earlier in time: the edge between the two should not be oriented from later count into an earlier count. And we can specify such constraints in what Tetrad GUI calls a Knowledge Box, which is then used to constrain orientation of edges during search.

Knowledge boxes can present such constraints in layers (called “tiers”). Many of the variables in a dataset can be assigned to different tiers with the understanding that a variable in a later (higher-numbered) tier cannot have an edge going into a variable in an earlier (lower-numbered) tier. The variables in the dataset that are not assigned to a tier are not so constrained.

With respect to the dataset that we just created, there is only one such type of constraint: variables representing what happened in the next time period (there are 11 such) cannot have a direct causal relationship to any variable in the current time period (there are 11 such, and “Start”).

The other variables: the 38 null variables (nv-\*) and 114 binarized variables (b\_\*) generally have no obvious relationship, chronologically or causally, with any other variable. In such circumstances, it’s best not to assign them to a tier but allow them to “float” during causal discovery.

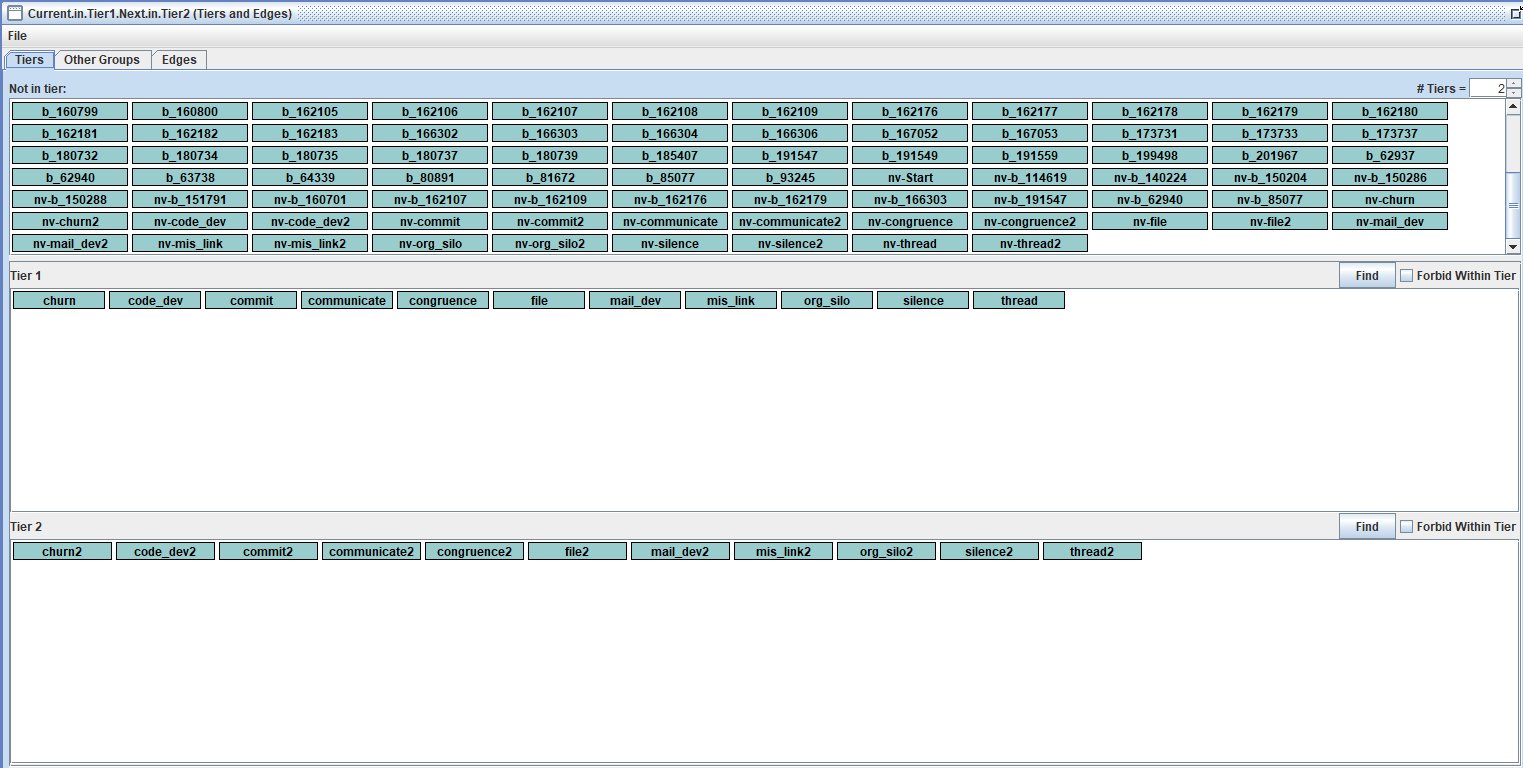
With that in mind, the appropriate tiering structure for the variables in our dataset is:

* **Not in tier**: 153 variables in total, including all 38 null variables (i.e., 1 (for “Start”) + 11 (for current time period) + 11 (for next time period) + 15 (active binarized variables)); the 1 (non-null) Start variable; plus all 114 binarized (non-null) variables.
  + The co-authors debated whether to keep the “Start” variable in the dataset, but it was thought that perhaps its presence would help highlight sociotechnical behaviors’ effects that were maybe more true at the beginning of the overall open-source project time period or at the end (e.g., a causal relationship that was more true 20 years ago vs. something more true today), which might shed some light on whether sociotechnical behaviors and their effects might be changing over the 20 years. And as noted, in the initial search, we found one such relationship, with CVE 2016-2107, though, as noted, this is just due to its recency and shortness of timeline; but perhaps there are other such CVEs.
* **Tier 1**: all 11 current time period-based sociotechnical and outcome variables
* **Tier 2**: all 11 next time period-based sociotechnical and outcome variables

The co-authors wrestled with this question: within a time period, should the 11 variables be placed in some causal order or their causal relationships be constrained in some way?

* It’s hard to judge what that order should be for these counts and measures of what went on within the same time period and it’s hard to say definitively that whatever one variable measures that is causing whatever a second variable is measuring to vary in some way within the same time period is a one-way relationship and that the first variable doesn’t respond in some way to changes in the second variable within that same time period.
* Later, we will see that although, searching with a very large number of bootstraps focused only on the current time period (abbreviated “current.TP”) or only on the next time period (abbreviated “next.TP”) allows a preferred replicable causal order to emerge, it’s better to consider these variables within the time period as occurring concurrently and no meaningful causal order should be attributed to them within the same time period (but we don’t make this same claim across successive time periods, of course). In general, the 11 variables associated with a particular time period seem to generally have no one-way oriented causal relationship with any other variable in the same time period. Thus, it is best to retain these variables within the same Knowledge Box tier and not try to artificially constrain how the causal discovery algorithm might choose to orient them.

With these cautions in mind, here is the Knowledge Box we will be using to constrain certain edge orientations during Causal Discovery, taken from the Tetrad GUI tool that we’ve been using:



In the above knowledge box, note the scroll bar in the top panel. The top panel lists the variables “Not in tier,” which is 175-22 or 153 of the variables, which obviously are not all visible in the small area afforded to the panel, but we see the bottom half (approximately) of these 153 variables, including 43 of the binarized variables (thus, out of view are 6 rows of 12 binarized variables each and “Start”) and all 38 of the null variables. If one were to click on the scroll bar to see the other variables, we’d see the other 6 rows of 12 variables each, which are all binarized variables except for the very first, which is “Start”, and which are sorted in ASCII order. (When you initially open a knowledge box, the variables are not in ASCII order, but briefly selecting a variable within a panel and just “shaking” it a tiny bit so that you haven’t left the panel and then releasing it will result in the variables appearing in ASCII order.)

**Apply Causal Discovery (Bootstrapped Search) to the Dataset Expanded with Null Variables**

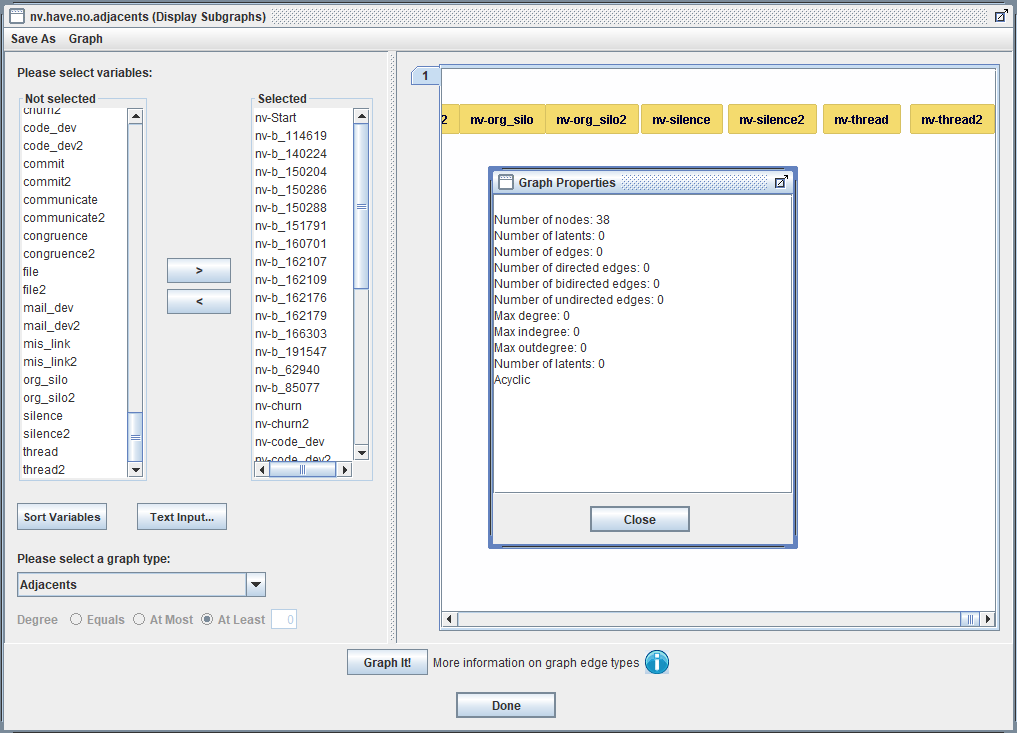
Now that we have a Knowledge Box with what seems a reasonable set of constraints (edges cannot form from variables in the next time period into variables in the current time period), we conduct our search. The settings used are as follows, which generally are the same as for our pre-null variable-injected search but this time performed on the 175-variable and 5414-case dataset:

* Tetrad GUI (version): tetrad-gui-6.8.0-20200828.153437-93-launch.jar
* Knowledge box:
  + We utilized the knowledge box presented in the previous subsection to help ensure that the edges formed have reasonable or at least not unreasonable orientations
* Algorithm: Fast Greedy Equivalent Search (FGES)
* Score method: Structural Equation Modelling Bayesian Information Criterion (SEMBIC)
* Penalty Discount: 2 rather than the usual default 1
  + Results in a sparser graph than the default setting of 1, reducing the number of false-positive edges (at the cost of more missing, that is, false-negative, edges; but that are also weaker in cause-effect)
* Symmetric First Step: set to Yes
* Number of Bootstraps = 1000
  + We use a larger number of bootstrap samples to improve precision and replicability of search results
  + Because the search setting “Include the original dataset as an additional bootstrap sample” is set by default, there will actually be 1001 bootstrap samples searched, an odd number, which is relevant to interpreting the result from the Majority Ensemble Method setting
* Percentage of Resample Size = 90
* Ensemble Method: Majority (2)
  + An edge is kept only if it appears in the majority of bootstrap sample search results in other words, if it appears in at least 501 of the 1001 searches.
* Otherwise, use default settings for the search
  + More information about causal discovery (a.k.a., “search”), the Tetrad tool (Tetrad GUI), and search algorithm settings are found here:   
    <https://cmu-phil.github.io/tetrad/manual/>.

Applying causal discovery to the 175-variable dataset produces a busy search graph, but we can display the graph, one portion (subgraph) at a time to get a better sense of what’s going on. We use the Graph box Display Subgraphs feature of the Tetrad GUI to focus only on a group of variables of interest at a time, selecting a group of variables and then selecting “Adjacents” to display which variables are adjacent to the members of the group (i.e., either have an edge going into a member of the group or are a destination of an edge coming out of a member of the group). We do this selection/display process for five variable groupings:

* Null Variables
* Binarized Variables
* Time Period variables
* Current Versus Next Time Period Variables
* Heartbleed CVE (Current and Next Time Period Variables)

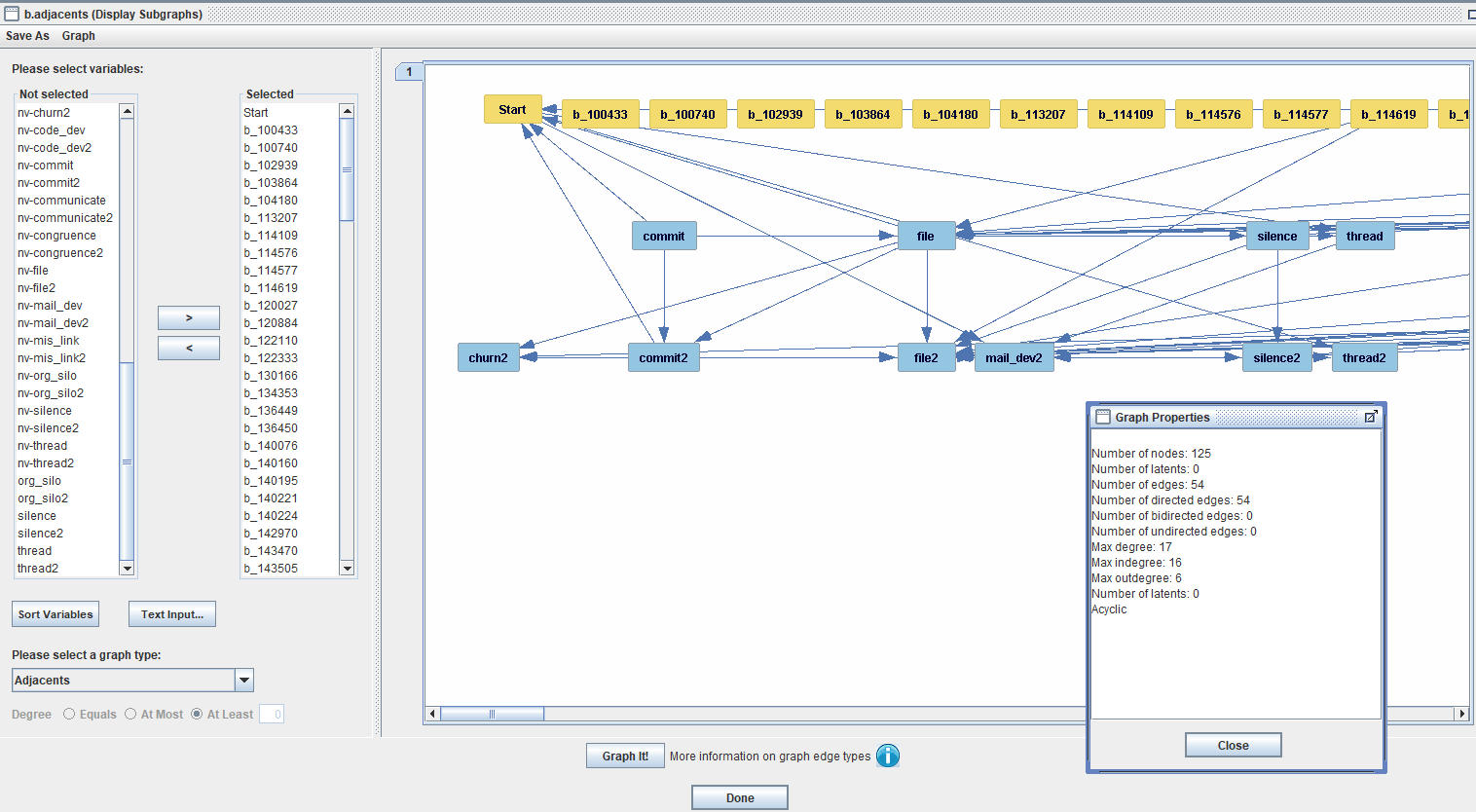
**Here are the null variables and their adjacencies:**



The graphical panel only shows a few of the null variables, but in the vertical “Selected” list, we see some of the nv\* variables and in the pop-up (obtained by clicking “Graph” on the ribbon and then “Graph Properties”) shows the count 38 for the “Number of nodes” and 0 for the “Number of edges.” The null variables, which in theory will be at best weakly correlated with any other variables, fail to form edges with any other variables—even themselves—that appear a majority of the time when bootstrapping.

If we had encountered adjacencies among the null variables, it could indicate that there was a problem in the pseudo-random number generation (or the way we set it up) of not generating numbers with sufficient independence. If we had encountered adjacencies with any non-null variable, it would indicate that our dataset is too small or our penalty discount was insufficiently high thereby allowing false-positive adjacencies to arise, and if they arise among the null variables with known statistical properties (apparent independence from all other variables), then it would imply that some or even many of the adjacencies that arise among the binarized variables, current time-period variables, and next time-period variables are also false positives. Fortunately, this didn’t happen, and we can go forward with more confidence. (Remembering, though, that confounders, such as the idiosyncratic attributes of a particular person, or their training, might create correlations among variables both within and across time periods that might lead causal discovery to impute adjacencies where there should be none. There is of course, never a large perfect dataset for any analysis and shortcomings of any analysis that might be performed. In Causal Inference with observational data, such confounding is the “Achilles Heel.”)

**Here are the 114 binarized variables and “Start”, and their adjacencies:**



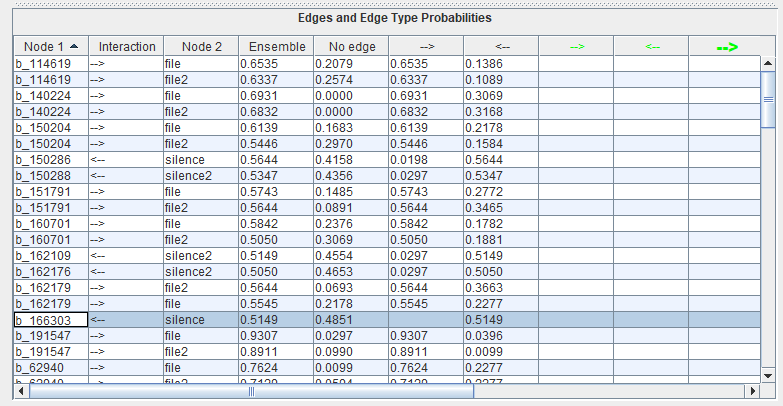
The graphical panel only shows “Start” and a few of the binarized variables—all of them in gold (that’s the way “Graph It!” works; the selected variables appear in gold following a “Graph it!” click); and the other variables (the non-Start and non-binarized variables adjacents to these) appear in blue. All of the blue variables appear in this graphical panel; if we were to scroll to the right (note the horizontal scroll bar at the bottom of the graphical panel), we’d see a long line of binarized variables (114 in total, of course) but no non-binarized variables.

So, the binarized variables and “Start” are adjacent to only 10 of the 22 time-period variables; including these pairs: “commit” and “commit2”; “file” and “file2”; “silence” and “silence2”; and “thread” and “thread2”; plus “mail-dev2” and “churn2.”

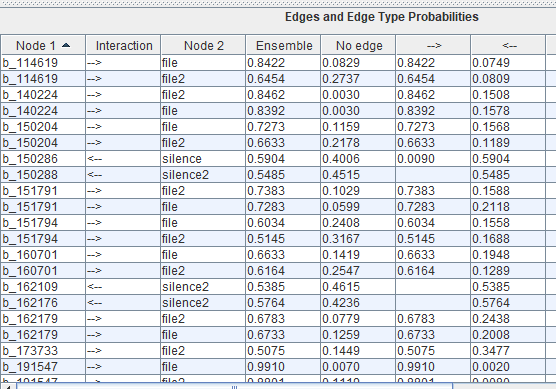
The pop-up Graph Properties box shows that though 125 variables (“nodes”) are displayed (“Start”, 114 binarized variables, 10 time-period variables), there are only 52 adjacent relationships among them; thus, the great majority of binarized variables have no adjacency. Repeating our exercise from the earlier subsection “Append Null Variables to a Dataset Prior to Bootstrapped Search,” we see that only these binarized variables (out of 114) have an adjacency (we explain the one strikeout and two CVEs underlined after the list):

2006-2940, 2008-5077, 2011-4619, 2014-0224, 2015-0204, 2015-0286, 2015-0288, 2015-1791, 2015-1794, 2016-0701, 2016-2107, 2016-2109, 2016-2176, 2016-2179, ~~2016-6303~~, 2017-3733, 2019-1547

Comparing this list to the previous list, recognizing the searches differed a little (no knowledge box earlier; and only 100 bootstraps), we find only a few differences. (Note that this exercise in itself is a kind of focused “replication:” we’re running search slightly differently [with “knowledge”] on a similar dataset [same but with/without null variables] and also looking to see if choosing only 100 bootstraps is enough to give us some kind of idea of the adjacencies present among these variables.) The differences are these: two new CVE ids were identified and appear above underlined, and one earlier-found CVE id no longer appears as having any adjacencies, and appears with a strikethrough (a horizontal line running across it). That CVE id is 2016-6303 and consulting the “Edges and Edge Type Probabilities” panel that appears under the first search graph (scroll down) we find:



Note that the probability of the edge having orientation “<--“ is 0.5149 and the probability of No edge is 0.4851, and so this edge was barely included in the search result with Majority Ensemble Method set. So, its disappearance when we selected 1000 bootstrap samples is hardly surprising. What about the two newly-discovered CVE ids having adjacencies? They are: 2015-1794 and 2017-3733. We find these nodes and their edges by likewise consulting the similar panel under the second search (with 1000 bootstrap samples):



Notice that b\_151794 has edges “-->” going into “file” and “file2” with respective probabilities 0.6034 and 0.5145, thus barely constituting a majority. Likewise, b\_173733 has an edge “-->” going into “file2” with probability “0.5075,” an even less impressive majority.

With only 100 bootstrapped samples, it shouldn’t be too surprising that a different number of adjacent time-period variables were found in the second search, but every time-period variable that was adjacent to a binarized variable according to the first search result is also adjacent in the second.

Once again, “Start” has a single directed edge with a binarized variable, and the same one: b\_162107.

Bottom-line: about 86% of the binarized variables (114-16 of 114) enter into no adjacency (a majority of the time, search-wise) with any of the 22 time-period variables; and thus noting that there are a few exceptions, we can assert that the causal graph constructed from the entire dataset is mostly independent of whatever CVE is being considered, which takes us to considering the graph for the time-period variables.

In particular, note that the Heartbleed CVE does not come up as one of the 16 exceptions. This suggests that what happened with the Heartbleed CVE timeline is not so different from most of the other CVEs, with respect to the causal pathways through time periods.

From a replication perspective there will of course be some variability in search results and a different set (but should be only sightly different) of roughly ten variables can be expect to appear in a replication of this study.

**Here are the 22 time-period variables, and their adjacencies:**

To simplify the overall graph, we take advantage of our results finding that the 98 (114-16) binarized variables and 38 null variables don’t have any (or at least, hardly any) adjacent time-period variables, and remove them from the dataset toward producing a simpler, less-cluttered graph with only 39 variables (“Start,” the 16 binarized variables that are adjacent to some time-period variables, and the two sets of 11 time-period variables—one set for the current time period and the other for the next time period). And we repeat the search from earlier in this section but with a larger number of bootstrap samples (2000) to improve the precision and general replicability of the result. We also search three ways with slightly different knowledge boxes that all differ with respect to where we place the binarized variables:

* No tier: in our previous search we did not assign the binarized variables to any tier.
* Tier 1: we place the binarized variables in their own tier (leaving only “Start” unassigned to a tier) but move the 11 variables previously in Tier 1 (Tier 2) into Tier 2 (Tier 3); the idea being that the idiosyncrasies of some CVEs somehow shaped how sociotechnical behaviors influenced the quality and productivity of a team’s work.
* Tier 3: same as the situation with Tier 1 except here we place the binarized variables in the highest-number tier, with the idea being that the earlier time periods led to a CVE forming, which thus occurred later; and although there are also time periods at the end dealing with the remediation of the CVE, much of the thrust of the sociotechnical behaviors lies in introducing and not immediately catching the vulnerability in the first place.

It will be interesting to see if the first of the three options (No tier) more-or-less replicate our graph from the previous search (but now the number of bootstrap samples = 2000) and whether the three search graph results differ from each other. In particular, which placement for the binarized variables, Tier1 or Tier3, most resembles the result from no placement? Do the three graphs have many edges in common?

An analysis within the Tetrad GUI shows that the three (No Tier/Tier1/Tier3) search results (graphs) have 85 directed edges in common. Focusing on the current-next time period relationships only, each of the 11 current time period variables can have up to 21 edges with other variables (10 from the current time period and 11 from the next time period); whereas, each of the 11 next time period variables can have up to 10 edges with other variables (all from the next time period). Thus, the total number of possible edges in such a two-tiered arrangement of 11 variables each is: (11\*10/2) + (11\*11) + (11\*10/2) = 231 edges; and we found 79 of these to be common (leaving out the 6 among the 85 that involve the Start variable), or almost 4 per variable. That’s a lot of edges and commonality.

Not surprisingly because of the forced assignment to opposite tiers, and the lack of any causal relationship between CVEs, the only binarized variable-terminating edge appearing in all three search results is:

b\_162107 --> Start

Here are the 85 common directed edges, with those edges whose source is the current time period and whose destination is the next time period appearing underlined:

Uncontradicted in 3 graphs...

1. Start --> mail\_dev2

2. Start --> thread2

3. b\_162107 --> Start

4. churn --> churn2

5. churn --> mail\_dev2

6. churn2 --> communicate2

7. churn2 --> file2

8. churn2 --> org\_silo2

9. code\_dev --> code\_dev2

10. code\_dev --> congruence

11. code\_dev --> file

12. code\_dev --> file2

13. code\_dev --> mis\_link

14. code\_dev --> silence

15. code\_dev --> silence2

16. code\_dev --> thread

17. code\_dev --> thread2

18. code\_dev2 --> churn2

19. code\_dev2 --> communicate2

20. code\_dev2 --> file2

21. code\_dev2 --> mail\_dev2

22. code\_dev2 --> mis\_link2

23. code\_dev2 --> silence2

24. code\_dev2 --> thread2

25. commit --> Start

26. commit --> code\_dev2

27. commit --> commit2

28. commit --> file

29. commit --> org\_silo

30. commit --> silence

31. commit --> thread

32. commit2 --> Start

33. commit2 --> churn2

34. commit2 --> code\_dev2

35. commit2 --> file2

36. commit2 --> mail\_dev2

37. commit2 --> org\_silo2

38. commit2 --> silence2

39. communicate --> congruence

40. communicate --> silence

41. communicate --> silence2

42. congruence --> silence

43. congruence2 --> communicate2

44. congruence2 --> file2

45. congruence2 --> mis\_link2

46. congruence2 --> silence2

47. congruence2 --> thread2

48. file --> Start

49. file --> churn2

50. file --> commit2

51. file --> file2

52. file --> mis\_link2

53. file2 --> mail\_dev2

54. mail\_dev --> commit

55. mail\_dev --> mail\_dev2

56. mail\_dev --> silence

57. mail\_dev --> thread

58. mail\_dev --> thread2

59. mis\_link --> code\_dev2

60. mis\_link --> file

61. mis\_link --> mis\_link2

62. mis\_link --> org\_silo

63. mis\_link --> silence

64. mis\_link --> silence2

65. mis\_link --> thread

66. mis\_link2 --> file2

67. mis\_link2 --> org\_silo2

68. mis\_link2 --> thread2

69. org\_silo --> code\_dev2

70. org\_silo --> congruence

71. org\_silo --> mis\_link2

72. org\_silo --> silence2

73. org\_silo2 --> communicate2

74. silence --> file2

75. silence --> silence2

76. silence --> thread

77. silence2 --> communicate2

78. silence2 --> file2

79. silence2 --> mail\_dev2

80. silence2 --> mis\_link2

81. silence2 --> thread2

82. thread --> mail\_dev2

83. thread2 --> communicate2

84. thread2 --> mail\_dev2

85. thread2 --> org\_silo2

Also, as already noted, only one of the 85 edges is with a binarized variable and that is this one: b\_162107 --> Start. However, five more of these 85 edges also involve “Start:” Start->mail\_dev2 and Start->thread2; probably reflecting the loss of the mail log early in the life of the OpenSSL project, which would induce correlations between “mail\_dev” and “thread” and Start (earlier and thus lower values of Start would be correlated with less mail activity, affecting both “mail\_dev” and “thread” variables); and likewise “mail\_dev2” and “thread2” variables. Of course, mail\_dev and mail\_dev2 are very highly correlated, so once one of these edges is seen, it masks the other via the partial correlation of the other, that is the correlation between “Start” and “mail\_dev” is also present, but mostly disappears when we instead consider the partial correlation between “Start” and “mail\_dev” conditioned on “mail\_dev2.” And, likewise, for “thread” vs. “thread2.” And we’ve already discussed b\_162107 --> Start before. The other three edges involving Start have Start as the destination: commit->Start, commit2->Start, and file->Start. There is a nontrivial correlation between Start and commit (about 0.26, p=0.000) and likewise, of course, for commit2. The lack of masking may be due to the strength of these correlations (as the Start date gets higher, so, in general, does the number of commits, perhaps reflecting the OpenSSL project tending to tackle larger tasks, over time). There is also a correlation between Start and file but it is comparatively weak (about 0.07).

Before proceeding, we evaluated whether using 2000 bootstrap samples (what we used in each of the three searches of 39 variables above) was sufficient to achieving some replicability in search results. Towards testing if the results would largely be replicated, we selected one of the three knowledge-based searches of 39 variables, the one least constrained by added knowledge (i.e., the one where the 16 binarized variables were assigned to No Tier), and repeated the search with identical knowledge and settings but with the Number of Bootstrap Samples (NBS) set to 10000 rather than 2000 used previously and compared the result. The new search again resulted in 120 directed edges, of which 118 were in common with the search with NBS set to 2000. Only 4 directed edges were different between the two otherwise identically-configured searches:

Uncontradicted in 1 graph...

1. b\_191547 --> churn2

2. commit --> org\_silo2

3. mis\_link --> org\_silo2

4. silence --> congruence2

Note that a new binarized variable appears, corresponding to CVE number 2019-1547 into churn2; as well as the three current time period into new time period directed edges, two of them from sociotechnical variables into the sociotechnical variables (mis\_link->org\_silo2 and silence->congruence2) and one from a work-rate variable (commit->org\_silo2). What are we to make of this? Well, first, that each of the two searches (10k vs. 2k) turns up the same 118 edges but 2 edges not present in the other. Here is an accounting for each:

1. b\_191547 --> churn2 (edge number 1) only appears in the NBS=10k search, but the No Edge type frequency is 0.4792 and the edge type “-->” frequency is 0.5194; so edge number 1 barely made it when NBS was set to 10k, which is what we’d expect.
2. commit --> org\_silo2 (edge number 2), like edge number 1, only appears in the NBS=10k search, but the No Edge type frequency is 0.4963 and the edge type “-->” frequency is 0.5037; so edge number 2 barely made it when NBS was set to 10k, which is what we’d expect.
3. mis\_link --> org\_silo2 (edge number 3) only appears in the NBS=2k search, but the No Edge type frequency is 0.4968 and the edge type “-->” frequency is 0.5032; so edge number 3 barely made it when NBS was set to 2k, which is what we’d expect.
4. silence --> congruence2 (edge number 4), like edge number 3, only appears in the NBS=2k search, but the No Edge type frequency is 0.4953 and the edge type “<--” frequency is 0.5047 (for the entry congruence2 – silence, so the edge type appears reversed); and so edge number 4 barely made it when NBS was set to 2k, which is what we’d expect.

Note that only statistics for edges that do show in the graph are available with the Ensemble Method 1 or 2 option set; and that’s why in the above four edges that only appear in one of the two graphs, we only focus on the statistics for the graph that the edge appeared in. So, when attempting replication, bear this in mind when looking for an explanation of a difference in the results of otherwise-very-similar searches.

In any case, we see that with NBS=2k, the results obtained are remarkably replicable. Summarizing: rerunning just on 39 variables No Tier for binarized variables, with NBS=10k and comparing the result with that obtained for NBS=2k: 118 of 120 edges found by either search was present in the other. A careful look at the four edges that were different found that all of them barely made the majority cut in the one search result in which they appeared.

Thus, we now focus on the remaining 82 time-period edges. As mentioned before the above common-to-all-three graphs list, we’ve underlined the 24 directed edges whose source (destination) is the current (next) time period. That leaves 58 edges whose source and destination are within the same time period. But our interest lies in the 24 edges that span different time periods as they identify which influences persist across time periods.

Fifteen of these 24 current-to-next time period edges have a source that relates to the volume of work done. We’ve been referring to these as “outcome” variables, but they really should be thought of as “work rates,” which include these six variables:

Six **work-rate variables**: churn, code\_dev, commit, file, mail\_dev, and thread

Here are the 15 edges whose source is a work rate in the current time period and whose destination is a variable in the next time period:

4. churn --> churn2

5. churn --> mail\_dev2

9. code\_dev --> code\_dev2

12. code\_dev --> file2

15. code\_dev --> silence2

17. code\_dev --> thread2

26. commit --> code\_dev2

27. commit --> commit2

49. file --> churn2

50. file --> commit2

51. file --> file2

52. file --> mis\_link2

55. mail\_dev --> mail\_dev2

58. mail\_dev --> thread2

82. thread --> mail\_dev2

Note that 5 of these 6 work-rate variables has a directed edge to its direct counterpart in the next time period (churn->churn2 and code\_dev->code\_dev2, etc.; the only exception is “thread”). Two of these 15 edges have a destination that is a socio-technical variable in the next time period (edge numbers 15 and 52; underlined above), indicating that (subject to there being no significant missing confounders):

* **code\_dev**, the number of coders who made commits in the **current** time period, has a direct causal relationship with **silence2**, the count of delays due to a lack of direct communication in the **next** time period (edge number 15)
* **file**, the number of files involved in a CVE’s remediation in the **current** time period, has a direct causal relationship with **mis\_link2**, the count of instances where two people were working on the same file with no evidence of direct communication between them in the **next** time period (edge number 52)

Of course, we’d expect both correlations, but it is interesting that the presence of next time-period variables code\_dev2 (first bullet) and file2 (second bullet), and their associated edges:

code\_dev->code\_dev2 and file->file2 (edge numbers 9 and 51)

are insufficient to mask the correlations that code\_dev has with silence2 (first bullet) and that file has with mis\_link2 (second bullet). What happens in the next time period is clearly not driven just by what is happening in that particular time period but by the previous as well.

During search, the Causal Discovery algorithm used (FGES in our case) takes into account just such relationships; as well as these other within-next-time-period relationships:

code\_dev2->silence2 (edge number 23 vs. first bullet)

code\_dev2->mis\_link2 (edge number 22 vs. second bullet)

communicate2->mis\_link2 (edge number 45 vs. second bullet).

We know that the Causal Discovery algorithm does indeed observe these correlations because it finds edges (9, 51; 22, 23, 45) but none of them, though pretty strong, are strong enough to mask out some remaining influence from the current time period. It’s as if something is going on in the current time period that transcends what the counterpart work rate variables and their interrelationships in the next time period are capturing with the result that what happened in the previous time period continues to affect (i.e., drive or cause) what plays out in the next.

Here are the other 9 edges (of 24), but these are from *sociotechnical* variables in the current time period into variables in the next (that again are common across all three No Tier/Tier1/Tier3 searches); and so regardless of whether/how the knowledge we specified constrains the causal edges that can form with the CVE-focused binarized variables):

41. communicate --> silence2

59. mis\_link --> code\_dev2

61. mis\_link --> mis\_link2

64. mis\_link --> silence2

69. org\_silo --> code\_dev2

71. org\_silo --> mis\_link2

72. org\_silo --> silence2

74. silence --> file2

75. silence --> silence2

Recall that there are five socio-technical variables:

Five **Sociotechnical variables**: communicate, congruence, mis\_link, org\_silo, and silence

Four of these five sociotechnical variables (i.e., all but congruence) are a source of an edge into the next time period. However, “congruence” has an extremely high correlation with “communicate” across the dataset (correlation is 0.99) and so observations we make here for edges from “communicate” likely also apply to “congruence.” Further, note that 2 of these 5 variables has an edge to its direct counterpart in the next time period (i.e., mis\_link->mislink2 and silence->silence2). But due to the very high correlation between them, what applies to mis\_link also likely applies to org\_silo, though the relationship will be partly masked; and so we possibly should include org\_silo->org\_silo2, as well; expanding the set of those that do have counterpart edges to 3.

Three of these 9 edges have a destination that is a work-rate variable in the next time period and these are edge numbers 59, 69, and 74 (underlined above), indicating that (subject to there being no significant missing confounders):

* **mis\_link**, the count of instances where two people were working on the same file with no evidence of direct communication between them in the **current** time period has a direct causal relationship with **code\_dev2**, the number of coders who made commits in the **next** time period.
* **org\_silo**, number of single collaborations between different developers in which at least one of them does not participate in the appropriate communication channel in the **current** time period has a direct causal relationship with **code\_dev2**, the number of coders who made commits in the **next** time period.
* **silence**, the count of delays due to a lack of direct communication in the **current** time period has a direct causal relationship with **file2**, the number of files involved in a CVE’s remediation in the **next** time period.

Note that we’re careful not to specify whether these relationships are good or bad, only that there is such a relationship. In general, one might suppose that these are likely to be bad, but determining if this is so is a different kind of evaluation, involving estimation, and the relationship might be nonlinear and more complicated to tease out. For example, bad, that is to say, a frequent occurrence of the three social smells (all three bullets above) may cause more coders to become involved in the next time period, involving more work and files, in order to deal with social smell-contributed errors. Or, on the other hand, perhaps in the next time period, fewer coders are involved due to frustration, resulting in fewer commits (but also drawing out the CVE timeline). We leave such an exercise to a future research study.

Once again, we are left wondering why the next-time-period-confined-edges (i.e., edges of the form X2->Y2) are not enough to explain the variation in these two next time period work-rate variables (code\_dev2 and file2) with the consequence that the edges found above (59, 69, and 74) must still score high.

The same wonderment applies to the six edges between sociotechnical variables, whose source is in the current time period and whose destination is in the next time period:

41. communicate --> silence2

61. mis\_link --> mis\_link2

64. mis\_link --> silence2

71. org\_silo --> mis\_link2

72. org\_silo --> silence2

75. silence --> silence2

(This box is of course simply the non-underlined edge subset of the previous nine-edge box.)

Looking at these 6 edges from sociotechnical variable (current time period) into sociotechnical variables (next time period) more carefully, we see these patterns:

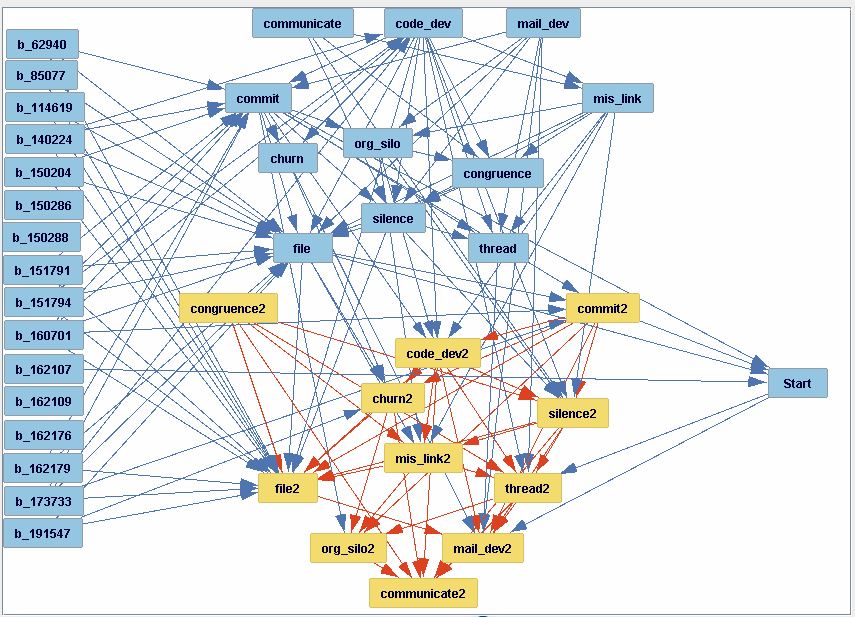
* For four of the five current time-period sociotechnical variables X, we have the direct causal relationship X->**silence2**, the count of delays due to a lack of direct communication in the **next** time period. Note that the sources for these four edges are four different sociotechnical variables in the current time period; but once again, we really should include “congruence” here given its very high correlation with “communicate.” In other words, there is strong evidence that multiple forms of sociotechnical behavior in the current time period directly influences (contribute to, cause) “silence2” in the next time period!
* And within the next time period, we find these direct causal connections as well, from “silence2” into: “comunicate2”, “mail\_dev2”, and “mis\_link2” in the next time period, and thus likely into “congruence2” as well.
* Thus, every sociotechnical variable in the current time period seems to have an almost direct causal relationship with a sociotechnical behavior in the next time period (via only one or two edges).

To better appreciate these findings: the variables in the next time period (i.e., those having a suffix “2” in their name) are insufficient to completely explain the work rates or sociotechnical behaviors experienced in that (next) time period; in fact, we find that the work rates and sociotechnical behaviors in the *current* time period influence not just what goes on in the current time period, but also what goes on in the *next*, often directly, strongly, and repeatedly for the almost 20-year spanned by the project dataset we employed; and largely regardless of which CVE was being remediated. (There are a few adjustments required for some of the causal relationships as shown by edges from binarized variables in the causal diagram below.)

Of course, if additional variables/measures were available, some of these edges could disappear, but the evidence seems quite strong that “What happens in Vegas stays in Vegas” with “Vegas” replaced by “previous time period” might not apply to the effects of sociotechnical behaviors.

With regard to the question of which tier is a more appropriate placement for the binarized variables, there is evidence from our comparison of all three (No Tier/Tier1/Tier3) search results that the answer is Tier 1 simply because that search result agrees much better with the result from searching the dataset in which the binarized variables were placed in no tier than do the results from placing them in Tier 3.

With all this in mind, let’s present the causal graph showing the 85 direct causal relationships with variables in current and/or next time periods and additional edges with the binarized variables and the Start variable:



We’ve colored the next time-period variables gold to make them stand out more clearly. In general, causal edges run downwards; however, the binarized variables are a bit of an exception because there’s no obvious place to put them in the graph, so they were moved to the left edge of the graphic. Note the fan-in from binarized variables onto “file,” “commit,” and “file2.” There’s something about the nature of some of these CVEs that perhaps encompasses more files than average, and perhaps more commits as well.

One apparent anomaly is “congruence2,” which has no incoming edges. Recall that the 11 current time-period variables are the same as the 11 next time-period variables, only they are shifted up one row (one time period). The causal subgraph for the next time-period (gold variables) should closely resemble the causal subgraph for the current time period (the 11 blue variables above the gold variables), but it doesn’t. (For example, “congruence” in the current time-period subgraph has four incoming edges from among the other 10 current time-period variables, whereas “congruence2” has none from its time period.) Why might that be?

A likely explanation is what has been noted before: these 11 variables are measuring events over the same time period and thus imputing/forcing a causal order to them might not be very sensible; and so the Causal Discovery algorithm does the best it can to identify/orient edges with the result that orientation in particular is driven by what best improves the score, and in the case of high-bandwidth connectivity between current and next time period variables, which seems to be the case with 24 edges from one set of 11 variables to the next, some orientations may indeed get flipped.

Another reason for apparent differences in the edges and their orientation between the two sets of 11 variables is the high correlation between certain pairs of variables, which can cause masking of causal effects. By high correlation here, we mean either that the correlation is > 0.9 or < -0.9. Looking at a correlation table, we find these four pairs of variables have high intercorrelations (correlation shown in parentheses, rounded to three significant digits):

* org\_silo and mis\_link (0.960)
* congruence and communicate (0.995)
* org\_silo2 and mis\_link2 (0.962)
* congruence2 and communicate2 (0.995)

Note that both members of each pair of variables is within the same time period.

What are the implications for causal search? It means that for each one of the above four pairs (A,B), a directed edge to/from X into/out of A will sometimes be preferred over a similar edge into/out of B. In particular, which edge is likely to appear in a search result will very likely be determined by small differences (e.g., the difference between two bootstrap samples). The consequence is that A and B will tend to share edges. One way to remedy this is to drop one of the two variables in a pair, for example, dropping either “congruence” or “communicate” (2nd and 4th bullets). Another is to construct a new variable that integrates the two variables (say, their principal component) and then drop them. At any rate, the existence of a pair of highly correlated variables (four such pairs in our case), will mean that a researcher is unlikely to replicate exactly the results that appear here, but if we treat A and B as “synonyms” for each other, most such differences should disappear.

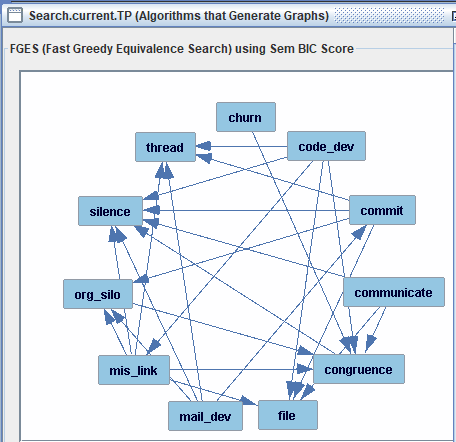
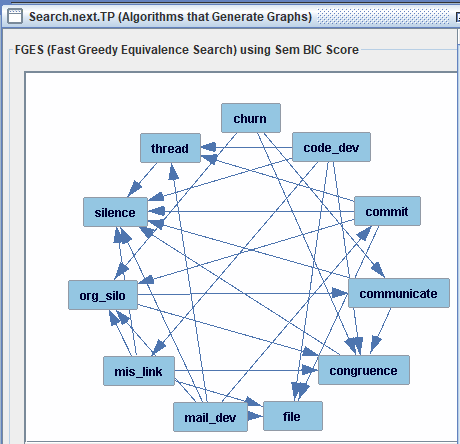
In other studies, we’ve adopted the first remedy, but in this case, due to the number of variables and high intercorrelations being manageable, and the desire to retain alignment with earlier sections of the article, we chose to retain rather than delete such variables to reduce confusion when comparing results with the other studies reported on in this article.

**Current Versus Next Time Period Variables**

What we’ve seen is that the orientations among the current time period variables are sometimes different than among the next time period variables (about 1/3 of the time). We’ve seen that there are multiple reasons for this. With this background, let’s revisit that congruence2 fan-in vs. communicate2 fan-in inconsistency: in the search result shown in the figure above, recall that “congruence2” had no edges coming into it. However, “communicate2” has six edges coming into it (from org\_silo2, congruence2, churn2, code\_dev2, thread2, silence2). (One may have to move nodes around in the Tetrad GUI to clarify what’s connected to what or consult the Edge and Edge Type Probabilities panel under the graph.) Is this really so different from the situation with the current time-period variables? Again, consulting the above search result, we find a reversal of sorts: it is “communicate” that has no edges coming into it. Recall communicate and congruence are highly correlated; and of course, so are congruence and its twin congruence2; and communicate with its twin communicate2, and therefore each pair of these are very highly correlated; and collectively, they should have very similar edges. Indeed, congruence has three edges coming into it (from org\_silo, code\_dev, and mis\_link). Two of these are the same, and the third, mis\_link is highly correlated with org\_silo. Also, note that the orientation between congruence and communicate changes.

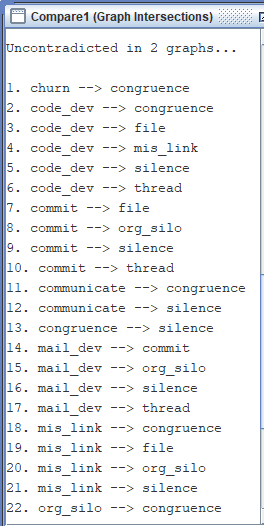
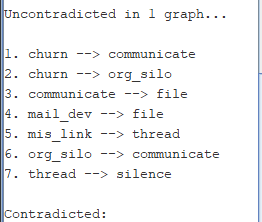
Why such reversals of orientation for the edges found among variables in the current vs. next time period? It may be that many variables in the current time period influence the next and the Causal Discovery algorithm (FGES) when presented with this situation, tends to find similar adjacencies (undirected edges) in the two time periods (reflecting similarly high correlations that survive conditioning on other variables) but then tends to orient many of these adjacencies in opposite directions (orientation happens after identifying the adjacency in FGES) because what maximizes the improvement in score tends to also be that which allows more variables in the current time period to reach more variables in the next by a directed causal path. This is a conjecture and not necessary to the exposition, but we offer it as an additional explanation for some of the differences we’re seeing in the orientations between the two time periods, when we include both sets of 11 variables in the same search. We mention this in part because when the two sets of variables are searched separately, many of these reversals disappear. In any case, it’s no more than about 1/3 of the edges that show such reversals when comparing within-same-period edge orientations, current vs. next; whereas about 2/3 of such edges have identical orientations.

To see if it is indeed the case that considered separately, the current time period causal subgraph is more consistent with the next time period causal subgraph, we subset the dataset into the 11 current time-period variables and separately the 11 next time-period variables and search each sub-dataset separately with the same search settings we’ve been using all along, with no knowledge box, but with 50000 bootstrap samples. Here’s the result (current TP on the left; next on the right):



The variables above at the left and right were purposely renamed to match and so just look at the general pattern of edges between the two graphs. They seem very similar.

Comparing these two graphs, formally, we find this list of directed edges in common between the two graphs (list on left) and list of directed edges found in only one of the two graphs (list on right):



And there are no contradicted edges (A-->B vs. A<--B).

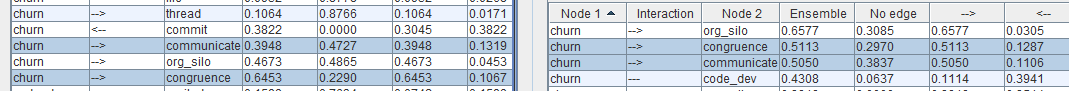
The current.TP graph has 24 edges (22 of them also found in next.TP graph); while the next.TP graph has 27 edges (22 found in previous.TP graph). So, 7 differences. In what follows, we attribute these 7 differences to five sources:

1. Bootstrapping produces edges of different orientations and so when we compare two graphs (results of bootstrapped search), and the first has an edge A --> B and the second has no edge for A and B, the first could be due to a more lopsided split than the second. For example, if in the first case, the edge type frequencies are: No edge: 0.1000, -->: 0.5100, <--: 0.3900 and in the second case: No edge: 0.1000, -->: 0.4900, <--: 0.4100; then an edge will only appear in the first case under the Majority Ensemble Method rules, even though the frequencies of each edge type are barely different from each other.
2. The masking situation already described: that is if we have A and B very highly correlated, then the conditions under which X --> A appears instead of X --> B is sensitive to very small changes in the data, say, due to small differences in bootstrap samples.
3. Sampling error, which is about differences in the bootstrap samples generated for bootstrapped search leading to slightly different frequencies in edge orientations. This error can be estimated to some degree, empirically, but not so precisely due to the other 4 reasons given here for differences in bootstrapped search results.
4. Cascading edge orientation effects can occur once the search algorithm commits to a different orientation for one edge while building its graph, it’s more likely to commit to different orientations for any connecting edges due to conditional independence tests. This cascading effect tends to be more pronounced for denser graphs (more edges per node).
5. Ambiguous edge orientation. Sometimes, the search algorithm is unable to attribute a particular edge orientation when building the graph because the difference in score between “-->” and “<--” is a tie. (This happens particularly in node neighborhoods where the graph is more sparse.) So, another reason why bootstrapped searches of the previous.TP dataset and next.TP dataset might differ is due to having insufficient density of edges, or oriented edges joining a particular edge to differentiate the improvement in score between these two orientations. We note those cases where the probability of edge type “---”, that is the undirected edge, is particular high (say over 0.1000) in what follows because unfortunately tool screenshots display that edge frequency in the 16th column of the Edge and Edge Type Probabilities (EETP) table, whereas in our screenshots, we like to show the first seven columns of the previous.TP and next.TP EETP tables side-by-side. That probability can also be inferred by simply subtracting the sum of the fifth through seventh columns, which display the probabilities for edge types “No Edge”, “-->” and “<--”, respectively, from 1.000.

We call the source for these differences: (a) “Near 0.5;” (b) “Masking;” (c) “Sampling Error;” (d) “Cascading Edge Orientation;” and (e) “Ambiguous Edge Orientation.” As it is more difficult to obtain visibility into Cascading Edge Orientation, we acknowledge the source but won’t attribute any differences to it.

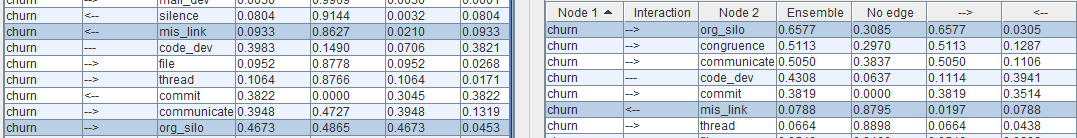
Considering these 7 differences (previous.TP graph vs. next.TP graph), one at a time:

1. churn --> communicate. Here is a screenshot of the relevant part of the Edge and Edge Type Probabilities Table (EETP), with previous.TP on the left and next.TP on the right:



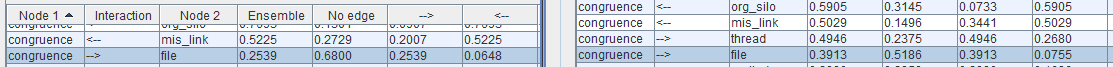
Note that the probability of (frequency for) No edge for “churn --> communicate” is 0.4727 and 0.3837, respectively, which makes “Near 0.5” likely to be the cause (note the probability of “churn --> communicate” is 0.5050 in next.TP) for the difference. Masking (see “churn --> congruence”) and/or Sampling Error might also be contributing: if in previous.TP, more “churn --> congruence” edges had been regarded as “churn --> communicate” instead, this difference might not have happened.

1. churn --> org\_silo. “Near 0.5” seems likely to be the explanation here. Note how close we are to 0.5 for the probability of “-->” for previous.TP (0.4673, third probability cell). But note that next.TP shows 0.6577 probability for the same edge, and this seems somewhat far apart (i.e., 0.4673 vs. 0.6577). Masking seems unlikely here given the high probability of No Edge for churn and mis\_link in both graphs (0.86-0.88). So, perhaps this farness is due to Sampling Error. It is not, however, due to Ambiguous Edge Orientation.



1. communicate --> file. Probability is 0.5557 in current.TP graph and 0.4108 in next.TP graph so this could be a case of “Near 0.5” perhaps combined with Masking because “congruence --> file” has similar probabilities but with a reversal in which graph has the larger probability. Here are screenshots of the relevant probabilities:





In other words, if more “congruence --> file” edges in the latter were regarded as “communicate --> file” instead, this difference would go away.

1. mail\_dev --> file. (Note the table entries for this edge appear as “file <-- mail\_dev” instead.) Per the screenshot below, the edge probability (fourth probability cell) is 0.4539 in current.TP graph vs. 0.5104 in next.TP graph. So, this difference seems to be adequately explained by “Near 0.5.”



1. mis\_link --> thread. Per the screenshot below, the edge probability (third probability cell) is 0.5699 in current.TP graph vs. 0.4494 in next.TP graph. So, this difference seems to be adequately explained by “Near 0.5.”



1. org\_silo --> communicate. Per the screenshot below, the edge probability (fourth probability cell) is 0.4623 in current.TP graph vs. 0.5260 in next.TP graph. So, this difference seems to be adequately explained by “Near 0.5.” In particular, the table on the left suffers from Ambiguous Edge Orientation. If there had been less ambiguity, the 0.4623 might have been high enough to exceed the 0.5 threshold required to place an edge.



1. thread --> silence. Per the screenshot below, the edge probability (fourth probability cell) is 0.4775 in current.TP graph vs. 0.6187 in next.TP graph. (Note the table entries for this edge appear as “silence <-- thread” instead.) So, this difference seems to be adequately explained by “Near 0.5.” While 0.4775 and 0.6187 may seem somewhat far apart, note that the orientation “-->” (third probability cell) is about just as far apart but in the opposite direction, so perhaps this farness is due to Sampling Error.



So, in summary, the two datasets, previous.TP and next.TP share 22 directed edges in common under Majority Ensemble Method but differ on 7 edges. These differences seem almost fully attributable to the graph that lacks one of these 7 edges having a probability for that particular edge type that is just a bit under 0.5 (and/or the other being a just bit over 0.5).

Another contributor to differences: the Majority Ensemble Method rule does not display an edge between a pair of variables in the not-so-infrequent situation where:

* The edge type probability for every edge type (-->, <--, ---) is under 0.5 while the probability of *adjacency* (i.e., some kind of edge type other than No Edge) being well over 0.5

This has sometimes made reporting or comparing the results of bootstrapped search using the Majority Ensemble Method frustrating, because one can argue that there should be an edge even if we don’t know how it should be oriented (perhaps orient as “---”, which we do when the orientation rules fail to apply to/reach that particular edge). For this reason, the authors have developed some code to interpret the results of applying the Preserved Ensemble Method to a bootstrapped search. In an earlier version of our data analysis, we employed just such code, but in the interest of significantly improved understandability for this article, perhaps at some slight cost to replicability, we chose to use the Majority Ensemble Method throughout. But we want the reader to be aware that this is an alternative, should they choose to employ similar analytic modeling methods on their own. The results from this more painstaking analysis and interpretation largely agree with those obtained from the simpler Majority Ensemble Method. (There were differences in the variables selected so we can’t say the results completely agree, but they certainly very consistent with each other, specifically with respect to there being significant direct causal relationships form what happens in the current time period into what happens in the next time period.)

This concludes a detailed explanation of how we applied Causal Discovery (bootstrapped FGES search) to the OpenSSL dataset assembled by the scraping tool. But the reader might be unfamiliar with Causal Discovery and Inference and might want to know why the authors chose to employ such an approach.

**Why do Causal Discovery and Causal Inference?**

Causal inference has entered the research methodology discourse in fields as diverse as Econometrics and Epidemiology; and indeed, almost all leading and significant research in these two fields is now conducted in this more rigorous setting.

A more technical discussion of how Causal Discovery and Causal Inference work can be found in the references; however, they are entering more widespread use in:

* Epidemiological studies, where how one establishes medical facts, guidance, and policy can be, literally, a matter of life or death
* Economics science, where major decisions are made regarding American and International economics policy.

For just two recent examples, we cite a paper by Miguel Hernán that has been described as one of the most influential papers in 100 years of the *American Journal of Epidemiology* [33, 34]; and the awarding of the 2021 Nobel Prize for Economics to Angrist and Imbens [35]

However, for our purposes, this may prove a sufficient explanation:

Regarding the use of Causal Discovery algorithms (FGES): when evaluating candidate edges, causal discovery *conditions* on (this is the verb “conditions” not the “noun”) other variables when searching for which new edges best improve the score for an intermediate-stage graph during search, evaluating whether a given pair of variables remain correlated even when taking conditioning on a subset of the other variables into account. If the partial (conditional) correlation even fails once, no edge will be credited to the variable pair. Thus, each edge in the resulting graph (a Directed Acyclic Graph or DAG [34]), has had to survive a “gauntlet” set of conditional independence tests (or the score equivalent), with a significant partial correlation found each time (or the edge would be removed). Note that each such conditioning constitutes an alternative explanation for how causality plays out. Though it takes more computation time, Causal Discovery is superior to ordinary (marginal) correlation between variable pairs in a research study as it eliminates many other alternative causal hypotheses for how the values attained by one variable affects/drives what happens with a second variable.

Continuing on our answer to “why causal discovery/inference:”

Likewise, causal discovery leads to superior covariate handling in linear/logistic regression [36, 37]. Key to estimating the direct casual effect of a treatment on a response by linear regression is determining which covariates to use. Presumably, the data analyst has experimented with different combinations of variables acting as covariates, but conditioning on the wrong set can create nonsense associations/edges between variable pairs.

And for these reasons, we employed Causal Discovery in our analysis of the OpenSSL dataset.

**Additional references:**

32. Muthén, L. K., & Muthén, B. O. (1998-2017). Mplus User's Guide. Eighth Edition. Los Angeles, CA: Muthén & Muthén.))

33. American Journal of Epidemiology, 2021. “100 Years of the American Journal of Epidemiology.” [100 years | American Journal of Epidemiology | Oxford Academic (oup.com)](https://academic.oup.com/aje/pages/100-years)

34. Miguel A. Hernán, Sonia Hernández-Díaz, Martha M. Werler, Allen A. Mitchell, Causal Knowledge as a Prerequisite for Confounding Evaluation: An Application to Birth Defects Epidemiology, American Journal of Epidemiology, Volume 155, Issue 2, 15 January 2002, Pages 176–184, <https://doi.org/10.1093/aje/155.2.176> (Selected as one of the most influential papers in the history of the journal.)

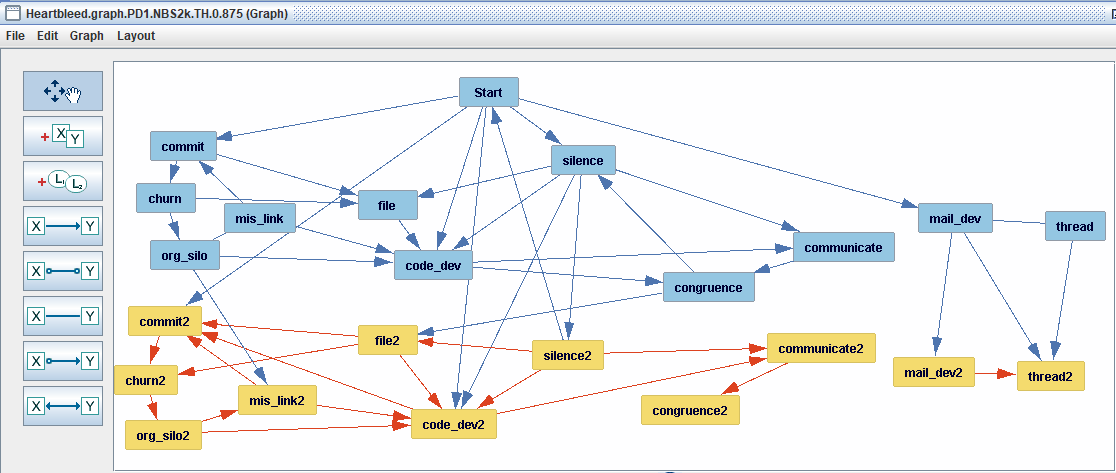
35. Nobel Prize Organization. “Press Release: The Prize in Economic Sciences.” Awarded to Joshua D. Angrist and Guido W. Imbens “for their methodological contributions to the analysis of causal relationships.” [The Prize in Economic Sciences 2021 - Press release - NobelPrize.org](https://www.nobelprize.org/prizes/economic-sciences/2021/press-release/)

36. Wikimedia Foundation. “Simpson’s Paradox.” <https://en.wikipedia.org/wiki/Simpson%27s_paradox>

37. Stanford Encyclopedia of Philosophy. “Simpson’s Paradox.” <https://plato.stanford.edu/entries/paradox-simpson/>

**LOG of other analyses performed and remaining open issues**

Four additional analyses:

1. Reran the causal discovery analysis (EM2) but with a larger NBS=10k. Ran bootstrapped FGES on 39 variables (Start, two times 11 time-period variables, 16 binarized variables that showing up earlier as having any edges) three times with the 16 binarized variables in either No Tier, Tier 1, or Tier 3, and with Number of Bootstrap Samples (NBS) parameter set to 2k (2000). Then reran just on 39 variables No Tier, with NBS=10k and compared the result with that obtained for NBS=2k and 118 of 120 edges found for each (same number) was present in the other. A careful look at the four edges that were different found that all of them barely made the majority cut in the search result in which they appeared; and so NBS=2k sounds like a perfectly adequate parameter setting for NBS in this domain.
2. Run the causal discovery analysis on Heartbleed only (CVE 2014-0160).
   1. Started with file nv-bin-OSSTRVZLM..AppendNextQtr..DeleteShortCVEs..Drop99nvs.csv
   2. Made copy, deleting all rows not associated with CVE 2014-0160 and all binarized variables and then all nvs derived from binarized variables that were constant 0, calling the resulting file: Heartbleed.CVE-2014-0160.csv.
   3. I then re-opened the Tetrad session Down.to.39vars.search.w.b\_vars.in.different.tiers.nbs10k.tet and inserted a note that simply said “Below--Heartbleed” and below it, placed a data box, where I loaded the data in. And then specified knowledge in a knowledge box and performed a search (Algorithm box named NBS2k.EM0.PD1); then saved the graph (file Heartbleed.PD1.NBS2k.txt) and then edited processGraphFilesv054.1.py with the correct filename and ran it to apply different percentile thresholds 1 through 5, noting that the last of these was about the right size 50-80 edges and with a cutoff threshold for PNE of 0.8750. (Goal was to find a percentile threshold that produces a number of edges roughly equal to that in the “common” (dataset-wide-based) graph. Fifth percentile is still a bit low, producing 47 edges for 23 variables, but went with that.)
   4. The result (deleting variable b\_140160 that I had accidentally left in, but of course it was not connected to anything because it was a constant) looks like this (colored next-time-period variables gold to help differentiate them from the other variables):
   5. 
   6. We can see from this figure that:
      1. The four variables addressing the amount of mail activity (mail\_dev and thread) in current and next time periods are closely related to each other but to anything else (though the arrow from Start into mail\_dev suggests either a general increase or general decrease over time).
      2. Likewise, the arrows from Start (and from “silence2” into it) signify those behaviors and CVE remediation work rates and behaviors that are generally increasing/decreasing with time: commit[2], code\_dev[2], silence[2] (the first two are work rates but the last is a social smell).
      3. Within each time period, we see a common pattern for Heartbleed: ignoring the “2” suffix in the variable name, both the current and next time period have this common structure, for example:

commit -> churn -> org\_silo -> code\_dev -> communicate -> congruence

mis\_link connects the same three variables: commit, org\_silo, code\_dev

file connects commit, churn, code\_dev, and silence

silence -> file -> code\_dev (also <- silence -> communicate)

* + 1. There are cycles, of course, due to the way this graph was obtained, nevertheless the internal consistency is pretty high even if some of the adjacencies are not consistently oriented (e.g., commit -> file <- churn vs. commit2 <- file2 -> churn2).
    2. Finally, as we did in the more general causal graph, we do see that sociotechnical behaviors in one time period impact the next. The particular variable here that does this most clearly is “silence,” which goes into silence2 and code\_dev2. But also: “org\_silo” into “mis\_link2;” and “congruence” into “file2.”
  1. Comparing Heartbleed against the more generally-based causal graph (that includes Heartbleed CVE timeline data as well as timeline data from another 113 CVEs): the above Heartbleed graph has 47 edges of which 45 are directed and 2 undirected (the generally-based causal graph includes the same two adjacencies but directs them thusly: mail\_dev ->thread and mis\_link-> org\_silo). Of these 45, 15 are identical between the general graph and Heartbleed; 15 are identical ignoring orientation, but are oriented oppositely to each other; and that leaves 15 directed edges appearing in the Heartbleed graph but not in the general causal graph. Now the general casual graph did not have the Heartbleed CVE as one of the idiosyncratic CVEs whose binary indicator included an edge into one or more of the time period specific variables, suggesting it should be largely the same or at least not significantly different/inconsistent. This finer analysis seems to suggest that indeed Heartbleed, with some exceptions (e.g., the mail activity was largely inconsequential from a causal perspective) had a similar set of causal influences as seen in the great majority of other CVEs.

1. Rerun on entire dataset but omit one from each pair of highly-intercorrelated variables: (org\_silo, mis\_link) and (congruence, communicate). While the great majority of edges were very similar [graphs resulting from searching the original dataset vs. reduced dataset], even running with Preserved ensemble method was insufficient to explain away a few of the differences: an edge involving a deleted variable too frequently had no counterpart in the dataset without that variable even though there was a very highly correlated variable in the reduced dataset, it sometimes would not engage in an edge with anything like the frequency in the original dataset. My conclusion is that masking effects can make accurate interpretation quite difficult if there’s a fair bit of it going on.
   1. Since “missing link” is explained in the text of the article, I (otherwise arbitrarily) chose to delete org\_silo (and org\_silo2) from the dataset.
   2. Likewise, somewhat arbitrarily, but because I thought “congruence” as a name was better differentiated than “communicability” as a name from the other sociotechnical variables, I chose to delete communicability (and communicability2) from the dataset.
   3. Given the very high similarity between NoTier for binarized variables with NBS=10k and the same but with NBS=2k, I just ran with NBS=5k.
   4. Result is, after searching with No Tier in both cases, one with the full set of 39 variables and the other with just the 35, is 95 directed edges in common (remember the search with 39 variables and NBS=10k had 120 directed edges, so 25 are still to be accounted for. In the other direction, there are 103 edges (one of these undirected); and so 8 still to be accounted for. And 32 uncontradicted in one graph, which I take to mean the one undirected edge was not counted as a contradiction. And so how many of these 32 edges uncontradicted in one graph involve deleted variables? 21, leaving 11 edges that have variables common to both searches. That work is shown at the bottom of this file.
   5. Most of these 11 edges are unsurprising (such as edge number 1); but some are very surprising, such as edge numbers 4 and 17 below). However, for the 39-variable (“39v”) dataset, upon a closer look, using the Ensemble method on the graph for which no corresponding edge appears, the mystery vanishes, there are in fact two edges whose edge type frequencies almost match, but just barely fail to meet the majority threshold (see edge number 17 below for an explanation).
   6. See “Work on analysis 3” far below for details.
2. Compare the result between *current* and *next* time periods for those particular 11-variable subgraphs within the larger 39-variable search. Are edges really often reversed? (Not explored but one could also ask: does the effect go away when doing #3?) Bottom-line, this is a more difficult analysis than one might think because of the two pairs of very highly intercorrelated variables and how to account for differences due to only such correlations, but bottom line is that working with exact pairs (i.e., sticking strictly to the variable names we’re given and not confusing things by saying this edge here is equivalent to that edge there when you allow highly-intercorrelated variables to substitute for their counterpart), we find that the same node-pairs in the two subgraphs have matching edge types (i.e., orientations) about twice as often as they are reversed, so the effect isn’t as severe as I first expected, though still more frequent than when the subgraphs are searched separately. In the 3 pages that follow, we show our work towards reaching this informal conclusion (somewhat based more on impressions than careful counts).

Work on analysis 3:

Uncontradicted in 1 graph... ((i.e., 32 edges appearing in one or the other of 39v.nbs10k or 35v.nbs5k but not both. 21 of the edges are in 39v and involve a variable that does not appear in 35v [i.e., communicate[2] or org\_silo[2]] but some of these do seem problematic, for example churn2->org\_silo2 is a majority edge type in 39v; but churn2->mis\_link2 is extremely weak in 35v. And there probably are other examples.

My overall conclusion from performing analysis 3 is this:

* I think it’s easy to underestimate the subtleties of masking and it is truly better to get rid of very highly intercorrelated variables before doing search; and this will also help with replicability as the masking introduces all kinds of variations on interpretation and explaining.

Nevertheless, for reasons of maintaining consistency with other sections of our article, I retain all the variables used and deal with each masking situation that arises as best I can.

1. churn --> congruence

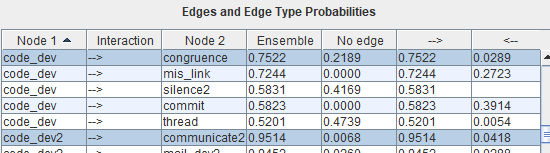
* ((the majority edge type for 39v.NBS10k is an edge that has frequency of only 0.5008; I’m not surprised this doesn’t appear in the graph 35v.NBS5k))

2. churn2 --> communicate2 (involves deleted variable so ignored)

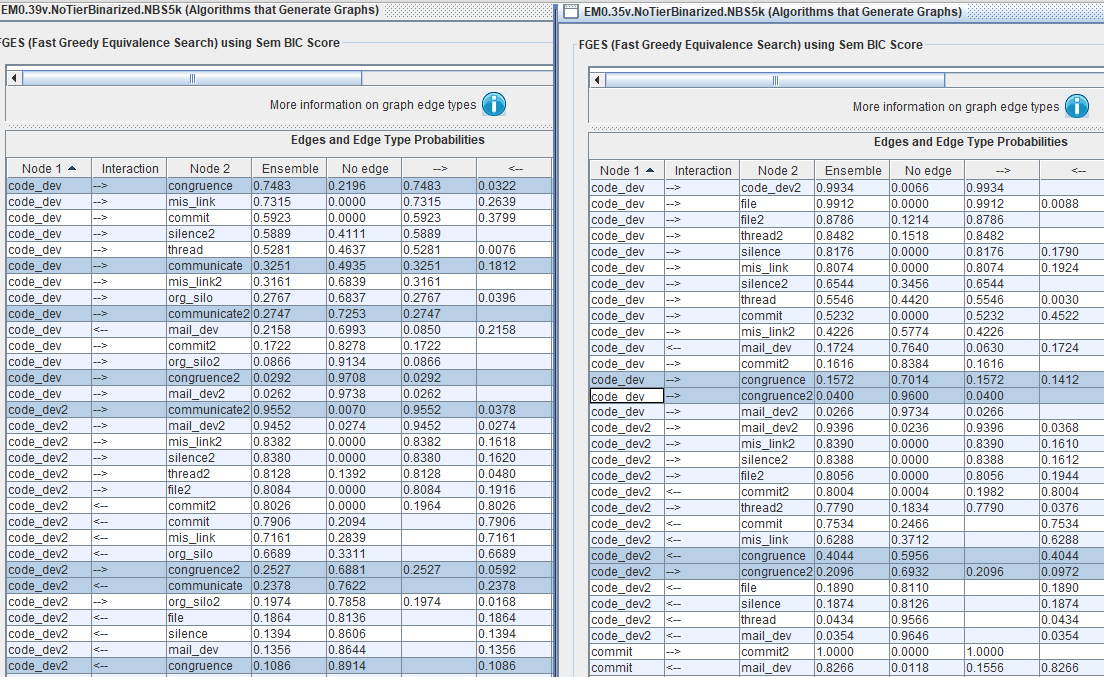
3. churn2 --> org\_silo2 (involves deleted variable so ignored)

4. code\_dev --> congruence

* Now, the absence of this edge is much more surprising. Looking at both code\_dev[2] with congruence[2] and communicate[2], we find no such edge in 35v.NBS5k (of course, there won’t be one with communicate[2]), but here’s what 39.NBS10k looks like, with the two relevant edges selected:

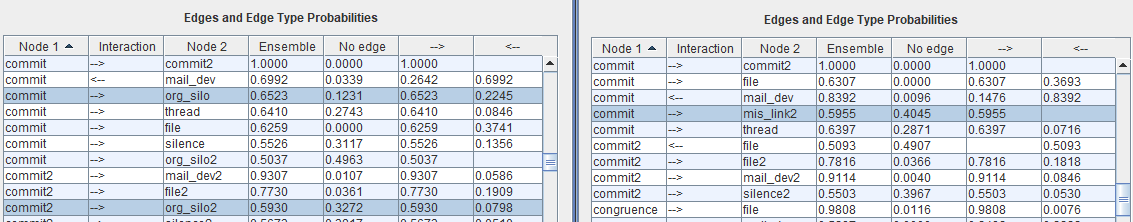


**And I have to say this presence on the left and absence on the right bothers me considerably. Especially with code\_dev2->communicate2 appearing 0.95 of the time in 39.nbs10k; but not at all in 35.nbs5k. I don’t know what to make of this except that sometimes masking effects can be severe.**

* I repeated the search (No Tier for Binarized variables, same settings as before for everything except: NBS=5k, EM=Preserved(0)) on both 39v and 35v datasets so I could see what the edge type frequencies were even when No edge was the majority/highest edge type.
* In the EETP table, the members of each Node 1 and Node 2 pair appear in alphabetical order (Node 1 is ordered before Node 2 in what I think of as ASCII order). Thus to find all code\_dev—congruence-relevant rows, and I will look in both time periods, I need to click on Node 1 column to sort by Node 1 in alphabetical order (what I call the order) and then look for the rows with “code\_dev” and “code\_dev2” in the Node 1 position. I arranged the EETP tables side-by-side. I then highlighted all combinations of:  
  code\_dev[2]—congruence[2] and code\_dev[2]—communicate[2]. Here’s the single screenshot that has all combinations (8 on the left and just 4 on the right because of course the communicate[2] variables were deleted from the dataset prior to search):
* 
* Cross-time-period edges:
  + Comparing (a) code\_dev2<-congruence in 35v vs. (b) code\_dev2<-congruence in 39v plus (c) code\_dev2<-communicate. Because of knowledge, -> is N/A for all three. Note that for <-, we have: (a) 0.4044, (b) 0.1086, (c) 0.2378; and that (a) almost is the same as (b)+(c).
  + Comparing (a) code\_dev->congruence2 in 35v vs. (b) code\_dev->congruence2 in 39v plus (c) code\_dev->communicate2. Because of knowledge, <- is N/A for all three. Note that for ->, we have: (a) 0.0400, (b) 0.0292, (c) 0.2747; and that frequency (a) almost is the same as (b) and they are an edge with the same names just different datasets; but (c) is quite different from (a) and (b), in spite of the fact that congruence2 and communicate2 are very highly correlated.
* Same time-period edges:
  + Of course, edge number 4 is a same time period-edge, and so these two cases are of greater interest. We looked at “Cross-time-period edges” just to evaluate search result consistency in a distantly-related case (I say “distantly-related” because although congruence and communicate are very highly intercorrelated (and as are the corresponding pair of variables for the next time period—both pairs are 0.994 or higher intercorrelation) and thus statements about one could be converted into statements for the other by variable substitution, in general, anyway; however, code\_dev and code\_dev2 are not so very highly correlated (correlation 0.634), and thus it’s very possible that a statement that is true about code\_dev and congruence[2] can’t be made for code\_dev2 and congruence[2] (where the square bracket 2 is intended to say either remove the square brackets and 2 throughout the statement or keep the 2 (and not the square brackets)).
  + Comparing (a) code\_dev2—congruence2 in 35v vs. (b) code\_dev2—congruence2 in 39v plus (c) code\_dev2—communicate2. Knowledge imposes no constraint here and thus we will list frequencies for both directions:  
    (a) -> 0.2096, <-0.0972; (b) -> 0.2527, <-0.0592; (c) -> 0.9552, <-0.0378. Once again (see two sub-bullets up): frequency (a) almost is the same as (b) and they are an edge with the same names just different datasets; but (c) is quite different from (a) and (b), in spite of the fact that congruence2 and communicate2 are very highly correlated. (Note that is specifically the direction (c) code\_dev2->communicate2 that is so different.
  + Comparing (a) code\_dev—congruence in 35v vs. (b) code\_dev—congruence in 39v plus (c) code\_dev—communicate. Knowledge imposes no constraint here and thus we will list frequencies for both directions:  
    (a) -> 0.1572, <-0.1412; (b) -> 0.7483, <-0.0322; (c) -> 0.3251, <-0.1812; and here, all three edges are quite different from the other two, especially edge (b), even though the identical edge type (->) in 35v has very low frequencies; in 39v, it has a very high frequency.

1. code\_dev2 --> communicate2 (involves deleted variable so ignored)

6. commit --> mis\_link2

* Now looking at commit->mis\_link2, and the near-equivalent commit->org\_silo2, in both graphs, we see this:
* 
* 35v.nbs5k on the right of course doesn’t have org\_silo, and we find the majority edge type for commit->mis\_link2 to be 0.5955 (No edge of 0.4045). On the other hand, 39v.nbs10k has commit->org\_silo of around 0.65 for the majority edge type, but commit->org\_silo2 of 0.5930 for the majority edge type (No edge of 0.3272), the latter being not so different from commit->org\_silo2, recalling the correlations between mis\_link and org\_silo being fairly high. Overall, not surprising, though a bit disappointed that there’s no commit->mis\_link in 35v.nbs5k. Still, not a huge surprise.

7. commit --> org\_silo (involves deleted variable so ignored)

8. commit --> org\_silo2 (involves deleted variable so ignored)

9. commit --> silence

* Looking for a moment at commit2->silence2 instead, we see their commonality in both search results:



Note how similar these are!

* Now looking at commit->silence:



* Note that the majority edge type is to 0.55, which is pretty close to 0.5, so not so anomalous to be missing from 35.nbs5k.

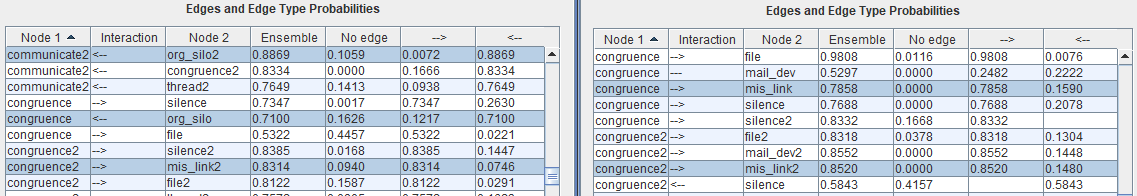
10. commit2 --> org\_silo2 (involves deleted variable so ignored)

11. communicate --> congruence (involves deleted variable so ignored)

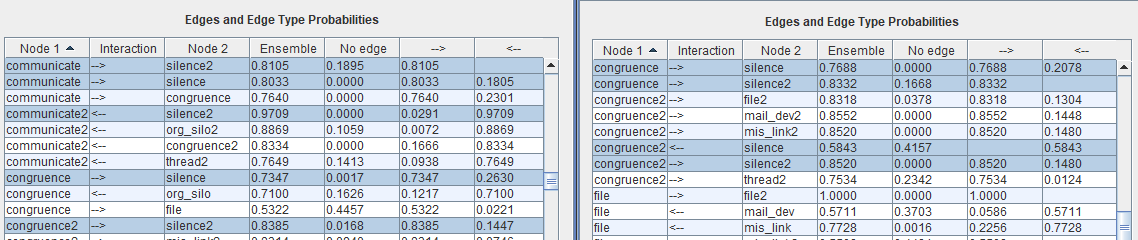
12. communicate --> silence (involves deleted variable so ignored)

13. communicate --> silence2 (involves deleted variable so ignored)

14. congruence --> mis\_link

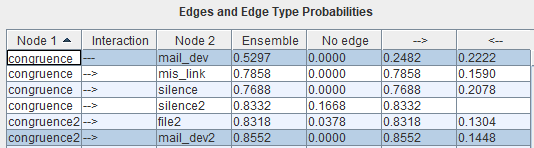
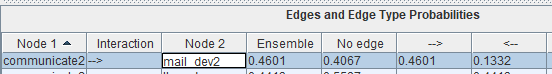
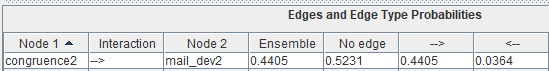
* Of course, here, congruence[2] might be replaced by communicate[2] *and* mis\_link[2] might be replaced by org\_silo[2]. Here’s all that appears (39v.nbs10k on the left; 35v.nbs5k on the right):
* 
* An edge congruence->mis\_link appears with very high certainty on the right (No edge = 0.0000) with majority type -> (0.7858). Corresponding to this on the left we find congruence <- org\_silo with low but not extremely low No edge (0.1626) and <- being the majority edge type (0.7100). Though with a different edge orientation, I do consider this edge analogous to the first one on the right. The bottom right highlighted edge, congruence2->mis\_link2, is the same edge but transported to the next time period, and again No edge is 0.0000, while -> is the majority edge type, 0.8520. The situation is far less muddled on the right than the left. Looking at the two highlighted edges that we haven’t yet addressed on the left, we see communicate2<-org\_silo2 and congruence2->mis\_link2, both with low but not terribly low No edge probabilities but opposite orientation. However, the latter of these (congruence2->mis\_link2) easily corresponds in frequencies to the same edge in the graph (i.e., Edges and Edge Type Probabilities highlighted rows) at right, which is what we’d expect to see. There’s clearly masking going on here, but I do consider this a reasonable difference between the two graphs, but it makes me think that there would have been value from deleting communicate and org\_silo (and their next time period counterparts) and only trying to deal with those search results. (Bottom-line: the situation at left is quite muddled, but much clearer on the right, and the one on the right is also much more defensible as the current-time period and next-time period versions of the same edge (congruence-mis\_link) have very similar frequencies, as we’d expect to see given the 5413 pairs of values they have in common; but again at the left this current-vs.-next commonality is very hard to see in what’s present.)

15. congruence --> silence2

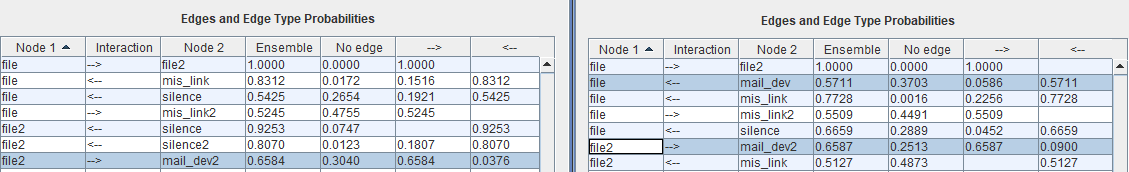
* Again, we expand to consider near-equivalent edges to find that 5 edges at left become 4 at right:
* 
* Focusing first on same-period edges above, congruence->silence and communicate->silence appear with nearly the same frequencies at left No edge very near 0, majority edge type -> 0.73 and 0.80, respectively. At the right, these are replaced by a single edge congruence->silence, also with very near 0 No edge frequency and -> majority edge type with frequency 0.77. These are pretty much indistinguishable from each other, but not the specific edge of interest.
* Likewise, first on next-period edges above congruence2->silence2 and communicate2 <- silence2 are near equivalents with extremely high certainty of their respective, but reversed edge orientations. A very muddled result. At the right, both are replaced by the single congruence2->silence2 near-certainty edge majority ->; and again near 0 No edge frequency. Can’t be too surprising, but interesting how muddled the situation is at left in comparison.
* On the other hand, the left hand graph has just one current time period->next time period type edge: communicate->silence2 with near-certainty edge majority -> (0.81) and No edge (0.19); in contrast at right we have two such edges: congruence->silence2 (the precise edge specification we were looking for) with-> 0.83 and 0.17 No edge frequency, which is very similar to the communicate->silence2 at left. But then we have the unexpected congurnce2<-silence with a high No edge frequency (0.42) and only a 0.58 frequency for the edge majority, whose frequencies do not look at all similar, but given this edge is not from a congruence-type edge into a silence type edge current time period into next, but rather the other way around, this extra edge (with no edge frequency 0.42 that is near 0.5) should not be considered surprising.
* So, I’m going to mark this as not so surprising.

16. congruence2 --> communicate2 (involves deleted variable so ignored)

17. congruence2 --> mail\_dev2

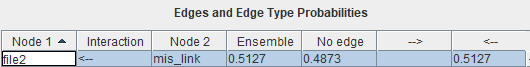
* There is no near-equivalent edge spec in the graph at left (i.e., nothing like communicate[2]->mail\_dev[2] or congruence[2]->mail\_dev[2]); so the question is, for the graph at right, is the same-time-period edge appearing in both time periods and at roughly the same No Edge frequency with a majority edge type near 0.5? Here’s the associated part of the graph:
* 
* And we immediately see that the edges have a rather different frequency (though 0.0000 of No Edge for both) and the former has an undirected edge type as the majority of around 0.53 frequency, so yes, that’s very near 0.5 (this is that one edge of 103 from 35v.nbs5k that was undirected!) and so we shouldn’t consider this edge popping up as surprising. But the second highlighted edge is the one numbered 17: congruence2->mail\_dev2, and it has the orientation “->”. Given how high the -> frequency is (0.8552), with nothing equivalent appearing in the graph at left, I’m going to mark this one as quite different. I’d have to run Ensemble Method as Preserved to evaluate this edge on the left more carefully. Perhaps the high 0.8552 is split between a pair of edges and I can only see this in Preserved for the graph at the left.
* Searching 39v.nbs2k with Preserved Ensemble Method (i.e., NBS=2k) (other settings the same): produces these respective congruence2->mail\_dev2 (communicate2->mail\_dev2) edge type frequencies (two screenshots):
* 
* 
* Remembering the very high correlation between communicate2 and congruence2 and that each is conditioning on the edge that the other has with mail\_dev2, we’d expect, after deleting communicate2 from the dataset, these ensemble frequencies for “->” and “<-“ to be added up into a single edge: “->” 0.4601 plus 0.4405 and “<-“ 0.1332 plus 0.0364, or for “->” around 0.90 and for “<-“ around 0.17. Whereas, as we can see from the screenshot for “congruence2->mail\_dev2” we have 0.8552 and 0.1448 for searching 35v.nbs5k, which is indeed very, very close. Thus, I upgrade what was red to yellow.

18. mail\_dev --> file

* In this case there are no near-equivalent edges, but this is a same-time period edge and so there is a corresponding one in the next time period. What do the two graphs show?
* 
* Note that on the left we have one edge but for the next time period and oppositely oriented; file2->mail\_dev2 with relatively high No edge (0.30) and with the orientation -> having frequency 0.66; in contrast, at the right, we have not so much contrast relative to file2 -> mail\_dev2 both appear and somewhat relatively similar frequencies; however, in the current time period, we find the exact edge specification sought. Even though file and mail\_dev have 5413 (of 5414) pairs in common with file2 and mail\_dev2 (both on the left and right), only one of these appear at the left, which could be a masking effect but a bit weird. Instead, the picture at right is more what we’d expect to see but with opposite orientation. Again, I assume context here (of having both sets of 11 variables and constraining edges interfering with the score obtained after either file, file2, mail\_dev, or mail\_dev2 on the left side get an edge into them not found at the same stage or earlier at right). At any rate, the specific edge type specified has a majority edge type of <- with frequency 0.5711, which is not so far from 0.5, so I’m going to guess that this is probably okay.

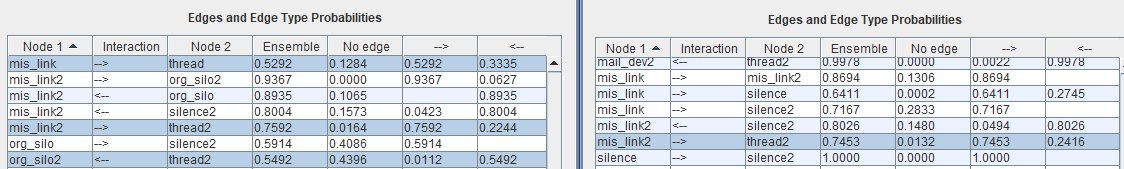
19. mail\_dev --> org\_silo (involves deleted variable so ignored)

20. mis\_link --> file2

* Another mixed time period edge, but with a near-equivalent org\_silo->file2 that can only appear on the left. Also, given our experience with analyzing a different edge above (edge number 15), we know not to treat this edge as somehow equivalent to file->mis\_link2.
* So, what do we have? Nothing equivalent to mis\_link->file2 on the left-hand side. And on the right:
* 
* Note the majority edge type has frequency only 0.51, so this is a near miss, and not surprising!

21. mis\_link --> org\_silo (involves deleted variable so ignored)

22. mis\_link --> thread

* The near-equivalent org\_silo->thread that can only appear on the left, but this is a same-time-period-type edge and so has a counterpart in the other time period. What do the graphs look like for this edge?
* 
* Three edges on the left, with the matching one on top and note that the majority type is only 0.5292; and on the right, only a next time period-version of the same edge with somewhat similarly-distributed frequencies. And note that these frequencies almost exactly match the next time period-version of the same edge on the left (mis\_link2->thread2). And so, this looks like another case of a near miss, and can’t be considered surprising. The third highlighted edge on the left with org\_silo2 as Node 1 might somehow be the result of masking and in any case has a majority edge type frequency of only about 0.55. So, I rule this as not surprising.

23. mis\_link2 --> org\_silo2 (involves deleted variable so ignored)

24. org\_silo --> code\_dev2 (involves deleted variable so ignored)

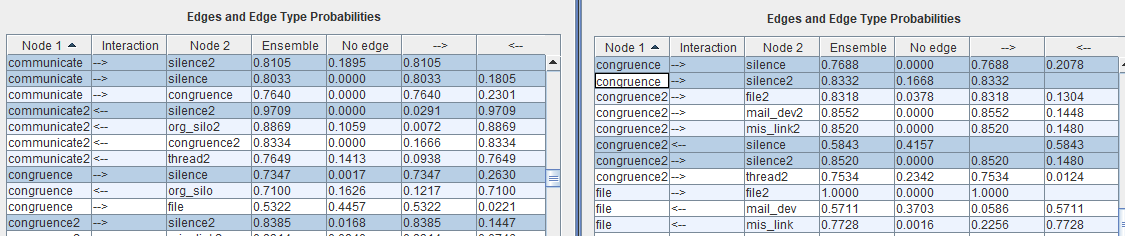
25. org\_silo --> congruence (involves deleted variable so ignored)

26. org\_silo --> mis\_link2 (involves deleted variable so ignored)

27. org\_silo --> silence2 (involves deleted variable so ignored)

28. org\_silo2 --> communicate2 (involves deleted variable so ignored)

29. silence --> congruence2

* Another current-to-next-time period type edge, with the near-equivalent silence->communicate2 that can only appear in the graph on the left. Here’s a screenshot of edges having one of these two specifications or the within-same-time-period versions of this cross-time-period edge (which really shouldn’t be expected to be similar, but we show for completeness):
* 
* So, there’s really only one the match here: congruence2<-silence, which is the third highlighted edge on the right. All of the others shown are either versions of the edge residing within the same time period (all but the top edge on the left; and all but the second and third edges on the right) or an edge also crossing from current into next but with nodes reversed (second edge on the right). So is the majority edge type near 0.5? It is 0.5843, which is pretty close, and no surprise.

30. silence2 --> communicate2 (involves deleted variable so ignored)

31. thread2 --> communicate2 (involves deleted variable so ignored)

32. thread2 --> org\_silo2 (involves deleted variable so ignored)

Work on analysis 4:

Uncontradicted in 3 graphs...(((removing all current->next edges and the six edges involving Start, leaves us with 85-30=55 edges. Then matching current and next pairs and identifying when orientations match vs. differ, etc.)))

6. churn2 --> communicate2

7. churn2 --> file2

8. churn2 --> org\_silo2

10. code\_dev --> congruence

11. code\_dev --> file

13. code\_dev --> mis\_link

14. code\_dev --> silence

16. code\_dev --> thread

18. code\_dev2 --> churn2

19. code\_dev2 --> communicate2

20. code\_dev2 --> file2

21. code\_dev2 --> mail\_dev2

22. code\_dev2 --> mis\_link2

23. code\_dev2 --> silence2

24. code\_dev2 --> thread2

28. commit --> file

29. commit --> org\_silo

30. commit --> silence

31. commit --> thread

33. commit2 --> churn2

34. commit2 --> code\_dev2

35. commit2 --> file2

36. commit2 --> mail\_dev2

37. commit2 --> org\_silo2

38. commit2 --> silence2

39. communicate --> congruence

40. communicate --> silence

42. congruence --> silence

43. congruence2 --> communicate2

44. congruence2 --> file2

45. congruence2 --> mis\_link2

46. congruence2 --> silence2

47. congruence2 --> thread2

53. file2 --> mail\_dev2

54. mail\_dev --> commit

56. mail\_dev --> silence

57. mail\_dev --> thread

60. mis\_link --> file

62. mis\_link --> org\_silo

63. mis\_link --> silence

65. mis\_link --> thread

66. mis\_link2 --> file2

67. mis\_link2 --> org\_silo2

68. mis\_link2 --> thread2

70. org\_silo --> congruence

73. org\_silo2 --> communicate2

76. silence --> thread

77. silence2 --> communicate2

78. silence2 --> file2

79. silence2 --> mail\_dev2

80. silence2 --> mis\_link2

81. silence2 --> thread2

83. thread2 --> communicate2

84. thread2 --> mail\_dev2

85. thread2 --> org\_silo2

Looking at orientations among A-B vs. A2-B2 more carefully and trying to match, I see that only about 1/3 of these are reversed, which is still kind of high, but less frequent than I thought:

6. churn2 --> communicate2

7. churn2 --> file2

8. churn2 --> org\_silo2

10. code\_dev --> congruence

11. code\_dev --> file

13. code\_dev --> mis\_link

14. code\_dev --> silence

16. code\_dev --> thread

18. code\_dev2 --> churn2

19. code\_dev2 --> communicate2

20. code\_dev2 --> file2

21. code\_dev2 --> mail\_dev2

22. code\_dev2 --> mis\_link2

23. code\_dev2 --> silence2

24. code\_dev2 --> thread2

28. commit --> file

29. commit --> org\_silo

30. commit --> silence

31. commit --> thread

33. commit2 --> churn2

34. commit2 --> code\_dev2

35. commit2 --> file2

36. commit2 --> mail\_dev2

37. commit2 --> org\_silo2

38. commit2 --> silence2

39. communicate --> congruence

40. communicate --> silence

42. congruence --> silence

43. congruence2 --> communicate2

44. congruence2 --> file2

45. congruence2 --> mis\_link2

46. congruence2 --> silence2

47. congruence2 --> thread2

53. file2 --> mail\_dev2

54. mail\_dev --> commit

56. mail\_dev --> silence

57. mail\_dev --> thread

60. mis\_link --> file

62. mis\_link --> org\_silo

63. mis\_link --> silence

65. mis\_link --> thread

66. mis\_link2 --> file2

67. mis\_link2 --> org\_silo2

68. mis\_link2 --> thread2

70. org\_silo --> congruence

73. org\_silo2 --> communicate2

76. silence --> thread

77. silence2 --> communicate2

78. silence2 --> file2

79. silence2 --> mail\_dev2

80. silence2 --> mis\_link2

81. silence2 --> thread2

83. thread2 --> communicate2

84. thread2 --> mail\_dev2

85. thread2 --> org\_silo2

There are 24 yellow-green (12 pairs) of items above and 12 red (6 pairs); or twice as many.

-

This ends the replication package.