Defect Prediction Analysis Using Bayesian Networks

Boshen Cui under Prof. Mei Nagappan and Prof. Peter Van Beek

1. Introduction

In any software project, bugs are critical and serious problem, detrimental to performance, and lead to unintentional consequences. One important aspect of software development is the ability to find and catch bugs before the program in question is released for public or commercial use.

One key part of this, and a key area of software development research, is being able to predict and determine where bugs are likely to appear in a project's codebase based on other, measurable metrics. The granularity of these metrics can vary, for example, being measured on a per OOP class basis, vs per package, or per file basis. Some examples are measures such as total lines of code (TLOC), total number of people involved, churn (number of lines added/removed), and change burst (groups of changes occurring closely together, separated by gaps of no changes between bursts) related metrics – size, time of occurrence, number of bursts, etc.

Thus, the problem is as follows: **given a dataset of various metrics about various aspects of a software project, can we predict where software bugs are likely to appear?**

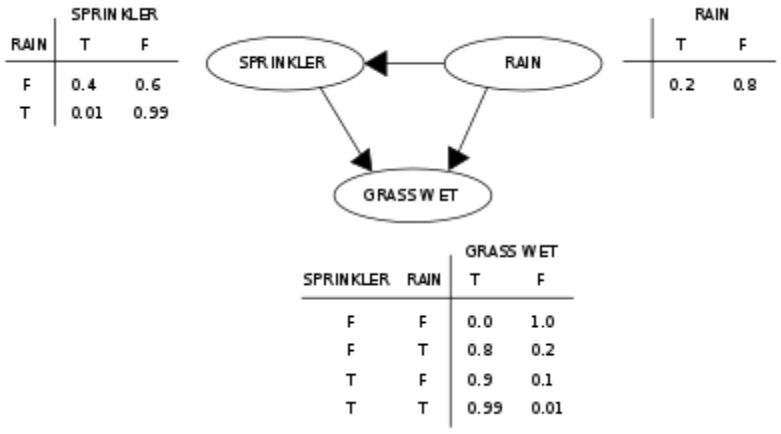
Currently, all prediction models are regression based – using techniques such as linear regression, or random forest classifiers. For example, a Saarland University study establishing the usefulness of change bursts as defect predictors used a stepwise regression model, resulting in a measured 67% precision and 51% recall against their ECLIPSE IDE dataset, and 91.1% precision and 92% recall against their Windows VISTA dataset.

This project explores using a new model – Bayesian Networks – as a predictor for defects, and as a verifier that the metrics being currently used **are** useful and correlated, and that current practices are correct.

1.1 Bayesian Networks

Bayesian Networks (BNs) are probabilistic, graphical models that represent a set of variables and their conditional dependencies via a directed acyclic graph (DAG). In this representation, the nodes of the graph represent the variables themselves and edges represent conditional dependencies. Each node has an associated probability function that, based on the possible values of its parents, outputs the probability of each of its possible values.

For example, in the graph below, the probability that grass is wet is modeled based on whether or not it is raining, and if the sprinkler is turned on.



(Wikipedia, 2018)

BNs are useful in that they can be used to infer and predict the value of unobserved variables based on the values of the observable variables that we know it is dependent on. In this case, the goal would be to build out a BN, based on a dataset, with the number of defects (NumberOfDefects) as a dependent variable, and various metrics as its ancestors. Then, from this, we will be able to see which metrics are most strongly related to NumberOfDefects, which may be weaker, and build out a probability function for NumberOfDefects, based on the values of the correlated variables and their probabilities, to predict which files, classes, or packages may be prone to bugs.

1.2 The Dataset

For this project, a subset of the ECLIPSE IDE dataset, used in the Saarland University study, was used to build out and explore BNs. The original dataset organized the ECLIPSE code, version history, and defect database in several ways – separating data into 4 different versions of ECLIPSE (2.0, 2.1, 3.0, 3.1), whether they were measured on a per class vs per package basis, whether bursts were measured on an hourly, daily, or weekly basis, and the different values used (ranging from 1 – 10) for burst size (minimum number required for a sequence of changes to be classified as a burst) and gap size (minimum distance between two changes for them to be classified as different bursts) to classify changes into bursts.

For this project, we focused on changes measured weekly, on a per class basis. For each version of ECLIPSE, two datasets were created – one “merged” dataset that combined all the different burst and gap sizes into one file, and adding in variables for them to account for the differences, and one “optimal” dataset, which was the dataset with burst size = 3, gap size = 3 from the original study, which was found to give the best results. In total, the dataset had 29 different variables (including NumberOfDefects). See exhibit 6.1 for the total list of variables, their descriptions, and their buckets.

2. Methodology

Going from a dataset to a BN requires several steps. At the core of this process is a program called GOBNILP (Globally Optimal Bayesian Network learning using Integer Linear Programming), a C program which “learns” BNs from complete discrete data, using the SCIP framework for Constraint Integer Programming.

However, GOBNILP requires discrete data, whereas the variables in the ECLIPSE dataset took on a range of continuous values. The distributions of each variable were first plotted, and each variable was categorized into logical buckets based on those distributions, using pandas and matplotlib.

After feeding the discretized datasets to GOBNILP, we used graphviz to output visualizations of the generated BNs.

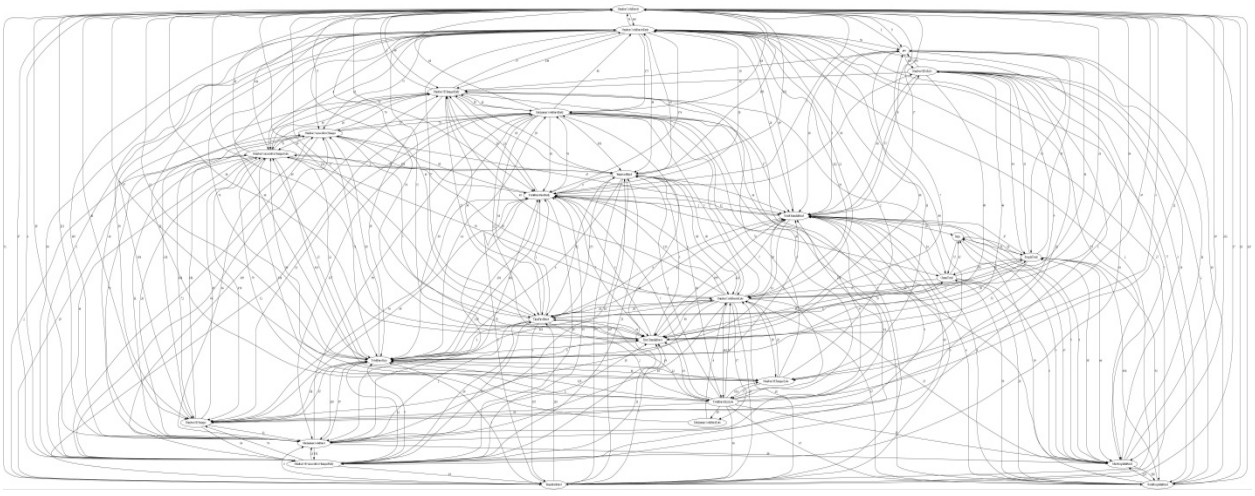
For simplicity, repeatability, and ease of use, python scripts were used to automate these tasks.

3. Experiments

From the discretized datasets, several experiments were performed, to explore the various shapes and similarities of the BNs generated.

3.1 Experiment 1

The first experiment compared the BNs between different datasets, and explored whether or not BNs between different datasets can be combined. For each dataset, we generated the top 10 BNs (i.e. the 10 BNs with the highest scores, 10 most optimal BNs). Then, we counted the number of time each edge appears in **any** of the graphs, and drew the resulting graph of all the edges, with the counts being the edge weights.



/experiment_1/results/aggregate_network.jpg

The result is an incredibly dense graph, with many edges with low weights. This suggested several things.

1. There was a lot of variation between different datasets (different versions, gap sizes, burst sizes)
2. There was a lot of variation between suboptimal graphs generated by the same dataset – even if graphs had similar GOBNILP score (a heuristic value that GOBNILP outputs based on how accurate it thinks its BN models the actual dataset), they may still look different. This leads to:
3. There may be multiple BN interpretations of a single dataset, which may all be equally as valid

3.2 Experiment 2

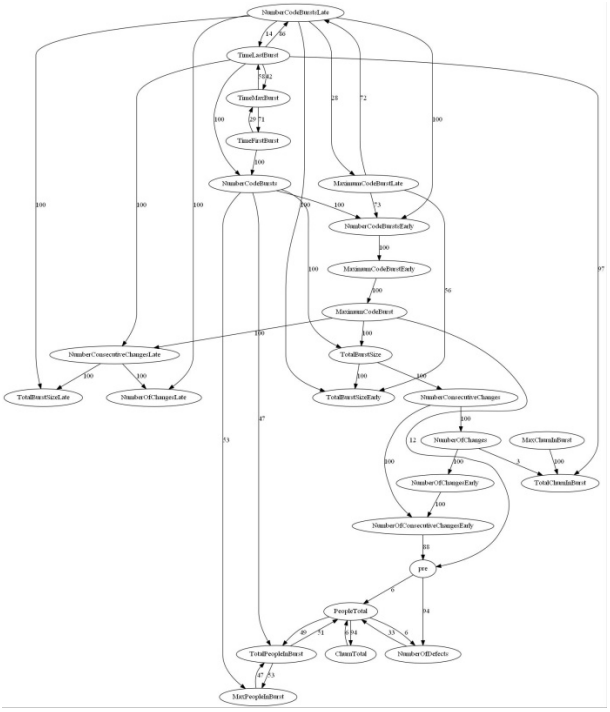
Next, we were interested in seeing how the structure of graphs and relationships between variables may vary between suboptimal BNs generated from a single dataset. For each dataset/file, we:

1. Generated the top 100 BNs for that file
2. Counted how many times each edge appeared in all of the BNs
3. Drew the resulting graph of all the edges, with the counts being the edge weights

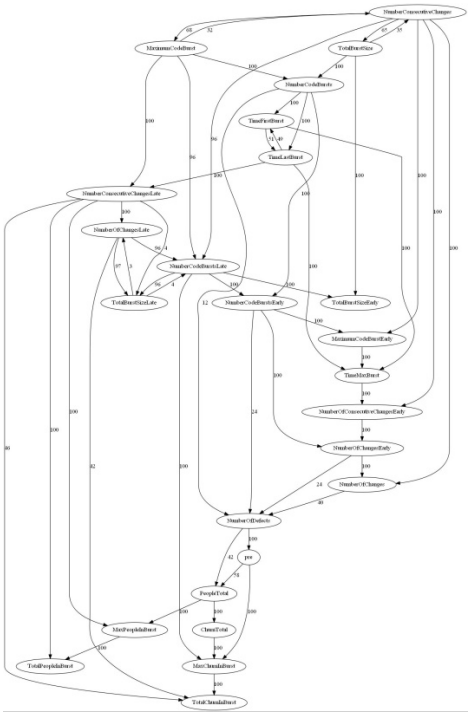
When doing this, it was important to ensure that all suboptimal BNs had a similar GOBNILP score. If the scores varied wildly, it may imply that some suboptimal BNs are not very accurate, and should not be included in the set.

The resulting graphs were much cleaner than the one generated in experiment 1, however, they were still very large and complex. From this, two observations were made:

1. Some edges appeared in most, if not all of the 100 suboptimal BNs, while others appeared less frequently (e.g. in only 50 of the 100 BNs generated, or even 5). This suggested that some relationships between variables (i.e. the ones that appear more frequently) are stronger than others (i.e. the ones that appear less frequently), and we can be more certain that there are relationships between those variables
2. The shapes of the graphs still looked very different between different datasets/files. This suggested that the BNs generated from one version/gap size/burst size of ECLIPSE can’t necessarily be used as a predictor for another.



/experiment_2/results/Eclipse20_GAP3_BURST3/aggregated_network.jpg



/experiment_2/results/Eclipse21_GAP3_BURST3/aggregated_network.jpg

3.3 Experiment 3

The next few experiments focused on comparing the differences between the “aggregated” and “optimal” datasets, to explore how adding the gap/burst size as variables could affect the structure of the graph, and if it was possible to combine data sets with different gap/burst sizes. For each version of ECLIPSE, we:

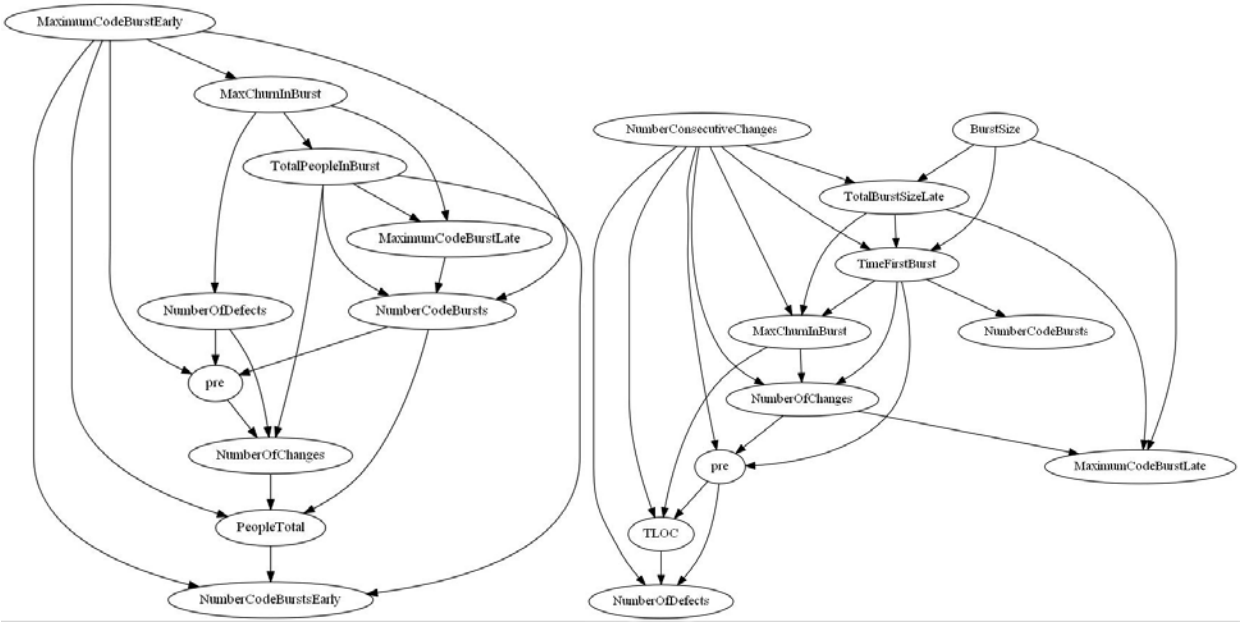
1. Generated the top 10 BNs for both datasets
2. For each dataset, counted how many times each edge appeared in all of the BNs and drew out the resulting graph, similar to what had been done before.
    a. One major difference this time, in an effort to simplify the graphs, was that edges that appeared in less than 50% of the BNs were “pruned”, and not included in the aggregated networks
3. Compared the best (i.e. top) BNs between the aggregated and optimal dataset
4. Compared the aggregated graphs between the aggregated and optimal dataset
5. For each dataset, compared the best BN vs the aggregated network – to see if the suboptimal BNs captured any relationships that weren’t present in the top network

However, these graphs still came out very large and messy. This was exacerbated by the fact that multiple variables were capturing the same, or very similar information – for example, out of the 29 variables, 3 were NumberOfChanges, NumberOfChangesEarly, NumberOfChangesLate. Furthermore, we noticed that from the graphs that were generated, few variables were direct

parents of NumberOfDefects, which was what we were interested in. Some were grandparents, or even higher up in the DAG tree, and some were not even in the same branch as NumberOfDefects – implying that they were independent from each other.

3.4 Experiment 4

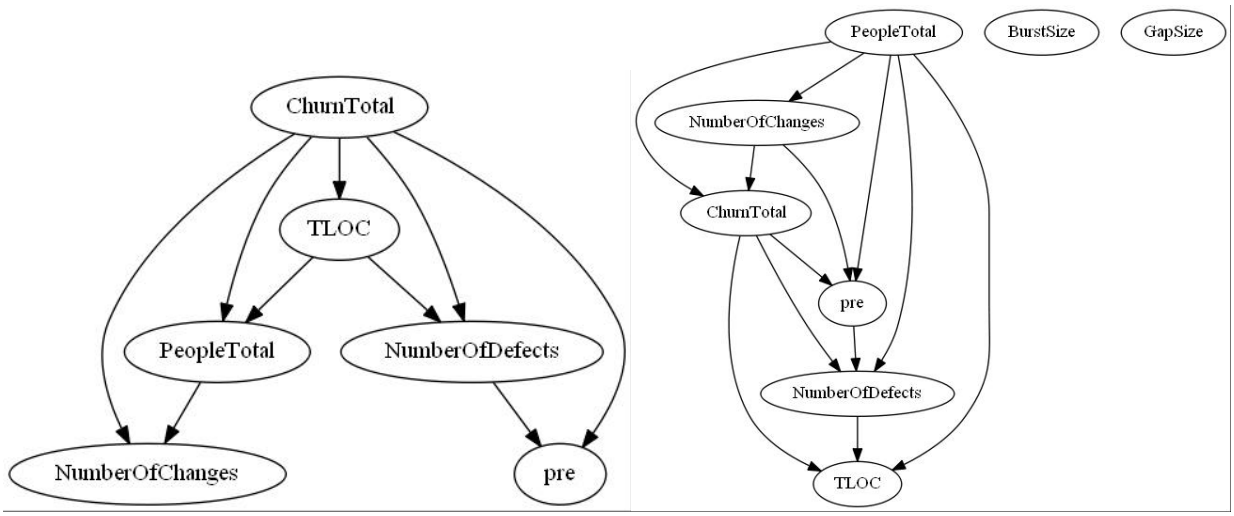
Based on the results of experiment 3, we decided to simplify the datasets by removing some variables, in order to pull more concrete, meaningful, and generalizable conclusions out of the generated BNs. First, we dropped any variables that were not either completely **dependent** or completely **independent** (i.e. nodes that had edges only going in, or only going out), based on the top BNs in experiment 3. Using this "pruned" dataset, we reran the procedure in experiment 3. The resulting graphs were much cleaner and readable, however, graphs between different ECLIPSE versions, and even between the optimal and aggregated datasets in same version still looked noticeably different. Furthermore, since the variables being kept were based on the results of experiment 3, they were not standardized, and varied from graph to graph:



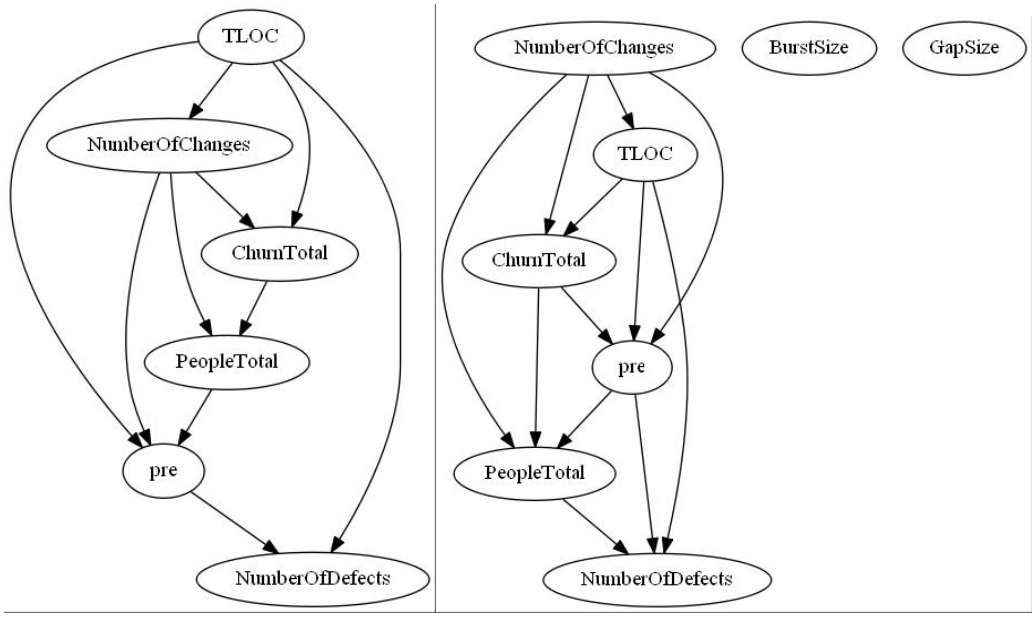
Comparing the top BN generated between the optimal vs aggregated datasets for ECLIPSE20.
Left: optimal (GAP3_BURST3) Right: aggregated (ALLGAPS_ALLBURSTS)

3.5 Experiment 5

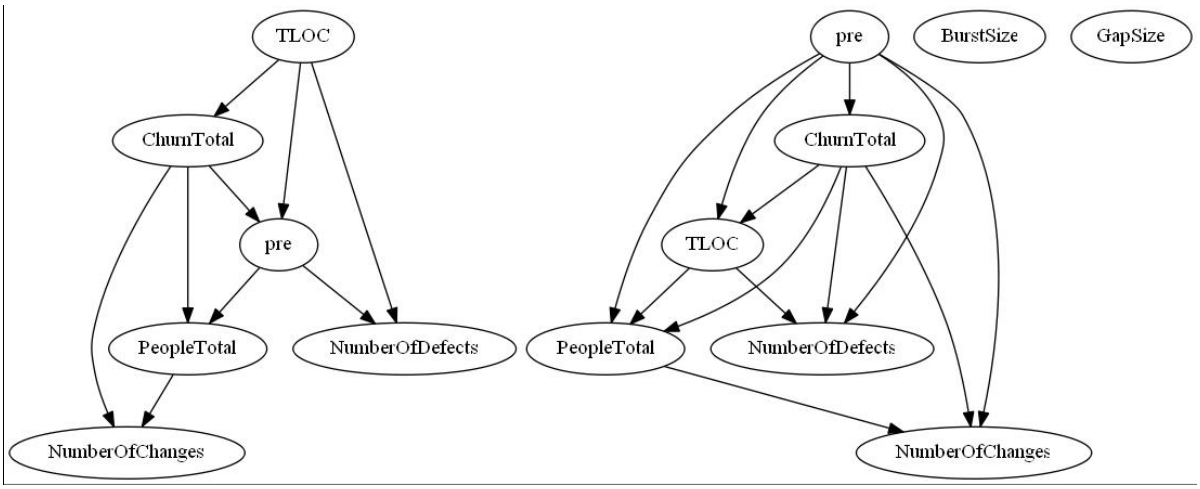
Finally, one last attempt at simplifying the dataset was made – this time by keeping only a standardized subset of variables: TLOC, BurstSize, GapSize, pre, PeopleTotal, NumberOfChanges, NumberOfDefects, and ChurnTotal, and rerunning the steps in experiment 3 on this "simplified" dataset. This resulted in much more consistent and sensible BNs across all versions:



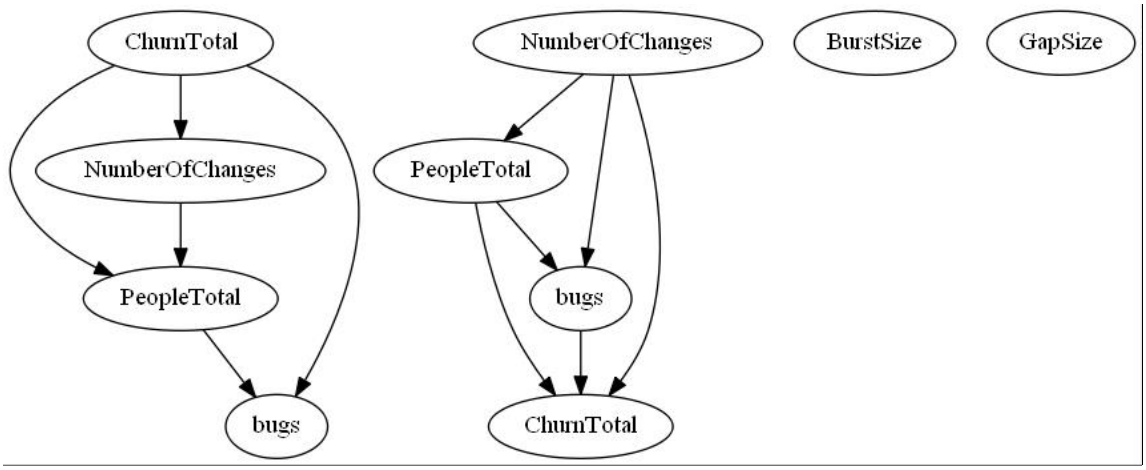
Comparing the top BN generated between the optimal vs aggregated datasets for ECLIPSE20.
Left: optimal (GAP3_BURST3) Right: aggregated (ALLGAPS_ALLBURSTS)



Comparing the top BN generated between the optimal vs aggregated datasets for ECLIPSE21.
Left: optimal (GAP3_BURST3) Right: aggregated (ALLGAPS_ALLBURSTS)



Comparing the top BN generated between the optimal vs aggregated datasets for ECLIPSE30.
Left: optimal (GAP3_BURST3) Right: aggregated (ALLGAPS_ALLBURSTS)



Comparing the top BN generated between the optimal vs aggregated datasets for ECLIPSE31.
Left: optimal (GAP3_BURST3) Right: aggregated (ALLGAPS_ALLBURSTS)

However, there are still some issues that need to be investigated: notably, why BurstSize and GapSize appear to be completely independent from the rest of the variables.

4. Conclusions

Based on the experiments so far, there are a few conclusions that we can pull out.

First, BN structure seems to be heavily dependent on the dataset that is fed into GOBNILP itself. Even if two datasets (for example, ECLIPSE20 and ECLIPSE21) are very similar, slight differences can lead to very different BN structures.

NumberOfDefects seems to only be strongly dependent on a few variables, having at most 1 - 3 as direct parents/connected directly by edges. It is still indirectly dependent on others (grandparents, >2 degrees of separation), while being completely independent from even others

(different branches of the DAG). In contrast to this, current regression models have been including all of these variables.

5. Next Steps

There are still many areas to take this model, and many more things to try and investigate. Notably:

1. Investigate why burst/gap size appear to be independent from other variables in experiment 5.
2. Further analysis on which variables seem to be closely related to NumberOfDefects. While the different BNs generated from different datasets seem to show different variables connected to NumberOfDefects, it is worth investigating if some are consistently connected to it, or if there is no pattern here.
3. Look into different discretization techniques. While some variables (e.g. ChurnTotal) span thousands of values, others are more contained, ranging from 1-100, or even 1-20. It may be worth looking into leaving some of these values be, instead of bucketing them down into 1-4 values, and seeing how this would affect the BN structure.
4. Letting GOBNILP run longer: on datasets with more variables to account for (i.e. experiment 3), a timeout often had to be imposed, otherwise GOBNILP would simply continue running, solving, trying to find a better network, without terminating. In these experiments, this timeout was 10 minutes, which allowed GOBNILP to get within 1-3% of the optimal score. However, it may be worth repeating these experiments with a longer timeout (i.e. hours, or even overnight), and seeing how this would affect the structure of the resulting BNs.
5. Building out a queryable program, based on the structure of some of the BNs. This would involve building out the probability functions described in section 1.1 for NumberOfDefects, based on the variables the BN state to directly affect it. The probabilities would be calculated directly from the dataset – for example, to calculate P(NumberOfDefects > 1 | TLOC < 10) (i.e. the probability that NumberOfDefects > 1 given that TLOC < 10), we would find the number of rows in the dataset where this is true, divided by the total number of rows in the dataset.
   After building out these probability functions, we can then test and evaluate how accurate it is on our datasets, compared to traditional regression models.

6. Exhibits

Full experiment results and graphs can be found at
https://github.com/Broshen/defect_prediction_analysis

6.1 Variables used, descriptions, and their discretized segments:

Variable Name	Description	Segments
bugs	Synonymous to NumberOfDefects, used only in ECLIPSE31 datasets. Number of bugs found in a class, after a version has been released	0, >0
BurstSize	Minimum number of consecutive changes made required for a sequence of changes to be classified as a burst	1,2,3,4,5,6,7,8,9,10
ChurnTotal	Total number of lines in the class that have been added, modified, or deleted	0, 1-10, 10-100, 100-1000, 1000+
GapSize	Minimum distance between two consecutive changes for them to be grouped into different bursts	1,2,3,4,5,6,7,8,9,10
MaxChurnInBurst	Maximum churn value across all bursts, in a class	0, 1-10, 10-100, 100-1000, 1000+
MaximumCodeBurst	Size of the largest burst that occurred within the classes' lifetime	0, >0
MaximumCodeBurstEarly	Number that occurred within the first 80% of the project's lifetime	0, >0
MaximumCodeBurstLate	Number that occurred within the last 20% of the project's lifetime	0, >0
MaxPeopleInBurst	Maximum number of people involved in the class, across all bursts	0, >0
NumberCodeBursts	Number of code bursts that occurred within the classes' lifetime	0, >0
NumberCodeBurstsEarly	Number that occurred within the first 80% of the project's lifetime	0, >0
NumberCodeBurstsLate	Number that occurred within the last 20% of the project's lifetime	0, >0
NumberConsecutiveChanges	Number of consecutive builds for a given gap size	0, 1, 2-10, 11+
NumberConsecutiveChangesLate	Number that occurred within the last 20% of the project's lifetime	0, 1, 2-10, 11+
NumberOfChanges	Number of builds in which the class had changed	0, 1, 2, 3+
NumberOfChangesEarly	Number that occurred within the first 80% of the project's lifetime	0, 1, 2, 3+
NumberOfChangesLate	Number that occurred within the last 20% of the project's lifetime	0, 1, 2, 3+
NumberOfConsecutiveChangesEarly	Number that occurred within the first 80% of the project's lifetime	0, 1, 2-10, 11+
NumberOfDefects	Number of defects observed for that particular class	0, >0
PeopleTotal	Number of people who ever committed a change to the class	1, 2, 3+
pre	Number of pre-release defects in a class, defects found before a version was released	0, >0
TimeFirstBurst	When the first burst occurred	0, >0
TimeLastBurst	When the last burst occurred	0, >0
TimeMaxBurst	When the biggest burst occurred	0, >0
TLOC	Total lines of code in a class	0-10, 10-100, 100+
TotalBurstSize	Number of changed builds in all change bursts	0, >0
TotalBurstSizeEarly	Number that occurred within the first 80% of the project's lifetime	0, >0
TotalBurstSizeLate	Number that occurred within the last 20% of the project's lifetime	0, >0
TotalChurnInBurst	Total churn in all changes bursts	0, 1-10, 10-100, 100-1000, 1000+
TotalPeopleInBurst	The number of people involved across all bursts	0, >0

7. Links

Github Repo: https://github.com/Broshen/defect_prediction_analysis

Original Study: https://www.st.cs.uni-saarland.de/publications/files/nagappan-issre-2010.pdf

Original Datasets: https://github.com/kimherzig/change_burst_data

GOBNILP: https://www.cs.york.ac.uk/aig/sw/gobnilp/

SCIP: http://scip.zib.de/

GraphViz: http://www.graphviz.org/