TEL AVIV UNIVERSITY
The Iby and Aladar Fleischman Faculty of Engineering
The Zandman-Slaner School of Graduate Studies

POST-SILICON TEST OPTIMIZATION USING METHODS FROM BIOINFORMATICS

A thesis submitted toward the degree of
Master of Science in Electrical and Electronic Engineering

by

Ron Zeira

January 2013

TEL AVIV UNIVERSITY
The Iby and Aladar Fleischman Faculty of Engineering
The Zandman-Slaner School of Graduate Studies

POST-SILICON TEST OPTIMIZATION USING METHODS FROM BIOINFORMATICS

A thesis submitted toward the degree of
Master of Science in Electrical and Electronic Engineering

by

Ron Zeira

This research was carried out in the School of Electrical Engineering
Department of Electrical Engineering – Systems
under the supervision of Prof. Ron Shamir and Prof. Dana Ron

January 201

Acknowledgements

I would like to thank my advisor Prof. Ron Shamir and all the members of the Computational Genomics group for their support and guidance in every step of the way during my M.Sc studies. I would also like to thank Prof. Dana Ron for co-supervising my studies.

I had the privilege of collaborating with Mr. Dmitry Korchemny from Intel Corporation, who initiated this research, provided the industrial data, and supported me with his expert knowledge. I deeply thank the Intel Corporation for funding my fellowship over the past two years.

Abstract

This thesis studies problems in optimization of hardware testing using computational biology techniques. Mathematically, the hardware testing data can be presented as a matrix whose rows correspond to the tests performed on the hardware and columns correspond to meaningful events measured during each test. The matrix values are the number of times the event occurred in the test. It is analogous to gene expression matrix, where rows are genes and columns are conditions. Taking this analogy further, we can use methods developed in gene expression analysis. For example, clustering techniques can be used in order to partition the tests (or the events) into similarity groups. The identified groups can then be analyzed by hardware validation engineers in order to find redundant tests and replace them with representative tests. Gene expression analysis and visualization tools, such as EXPANDER, can assist in comprehension of the validation process. We also explore combinatorial methods for set cover problems in order to find small test sets with good event coverage.

We also define and study a new approach to clustering, based on finding cohesive subgraphs in an undirected weighted graph. The objective function of cohesion is a generalization of subgraph density, defined as the ratio between the number of edges and the number of nodes in the subgraph. Inspired by graph clustering, cohesion discourages the inclusion of inter-cluster edges and high degree nodes. We give a polynomial algorithm for finding a maximum cohesion subgraph in an undirected weighted graph, based on iterated flow computations. We then test our new approach on simulated clustering data generated using different models. We report improved performance using cohesion compared to density.

Contents

Figures

Tables

Algorithms

1. Introduction and summary

In today's world, electronic chips play a major role in almost every device we use. Chip designers and manufacturers invest vast resources in validation of their products to ensure their reliability and stability of operation. This validation process, referred to as silicon validation, is divided to two major parts: pre and post silicon validation. Pre-silicon validation is done using software tools before a real chip is ready. It usually focuses on smaller functional blocks and thorough testing. Post-silicon validation, on the other hand, is the validation of a whole system on a real prototype chip. It focuses on system level tests and has less debugging capabilities than pre-silicon validation.

In post-silicon validation, chip behavior is monitored by measuring internal signals, called events. Many random tests are run on the chip so that important events are eventually observed. Since the internal state of the chip is not completely observable, validation engineers manually design sophisticated tests in order to observe certain rare events. Such tests accumulate over time in chip design companies, resulting in cumbersome validation test suites.

In our research we try to study these validation test suites. We use the reports on events occurring during the tests to formalize the problem mathematically. We use results from a real industrial test suite. Our approach employs methods that have proven successful in computational biology.

We first investigate ways of reducing the number of tests used while keeping the same coverage of the events, using method developed for set cover problems. We substantial show reduction in the number of tests depending on the coverage goal selected.

Another question is choosing the order of tests. We compare several heuristics and several objective functions, and show that a very simple greedy heuristic outperforms the others in almost all aspects.

Then we turn to clustering similar tests, i.e., grouping together tests that cover similar sets of events. A partition of the tests into groups can be used by validation engineers to further understand test functions and reduce the number of tests, or design new ones to address less covered events. We investigate several alternative definitions for similarity between tests and use several algorithms for clustering. All solutions show a highly homogenous group of tests composed of more than 50% of the suite. This means that most of these tests have similar results, and hence there is redundancy among them. We use tree hierarchy to present similarity between tests. We then discuss how to interpret clustering results using additional information on test parameters. The analysis shows that a group of similar tests tends to use specific parameters that do not appear in other groups. This enables the validation team to identify important parameters for the design of new tests and to gain insight on the internal behavior of the chip.

The second part of this thesis is devoted to finding cohesive subgraphs in an undirected weighted graph and using this method to partition the graph. We define the cohesion of a subgraph by modifying the known definition of subgraph density. The density of a subgraph is defined as the ratio between the number of its edges and the number of its nodes. We propose a generalized definition, which is more suitable for graph clustering. The new definition, called cohesion, discourages the inclusion of inter-cluster edges and high degree nodes. Cohesion uses two parameters to weigh inter-cluster edges and node degrees. We give a polynomial algorithm to find a maximum cohesion subgraph. The algorithm, which generalizes Goldberg's density algorithm, performs iterated network flow computation. The algorithm requires $O(\log(n * w_{max}))$ max flow computations, where w_{max} is the maximum edge weight in the graph and n is the number of nodes.

To test the cohesion concept, we propose several random graphs models, using both weighted and unweighted graphs. We test the maximum density and maximum cohesion algorithms on data simulated using each model. All results show better clustering quality using cohesion under high levels of noise.

2. Preliminaries and background

In this chapter we give the required background for our work. We start with an overview of silicon validation, problem formulation and research goals. We then cover the computational, mathematical and statistical methods we base our work on. We also review previous work done on post-silicon validation.

2.1. Silicon validation

In this section we will review the terminology and methodology of silicon validation. We will discuss the differences between the validation stages and then focus on the post-silicon validation stage.

2.1.1. Pre- and post-silicon testing

The complexity of today's microprocessor's silicon designs is increasing rapidly and together with tight time-to-market product schedule many verification challenges arise. One of the key challenges is validation, namely, making sure (to the extent possible) that the produced chip meets its specifications and functions correctly without bugs. Chip manufacturers spend enormous efforts on validation to ensure reliability of their products. Intel corporation alone invests over $300 million annually in validation [1].

The validation process begins during the first stages of component design by defining the test plan, and continues throughout pre-silicon, post-silicon development and manufacturing. Each stage of validation differs in its scale (what is being tested), depth (how many and what tests are made), controllability (specific test generation), observability (ability to monitor internal behavior) and duration [2]. Bugs decline in numbers over the development and validation processes, but increase in cost. A summary of the validation stages domain and characteristics is presented in Figure 1.

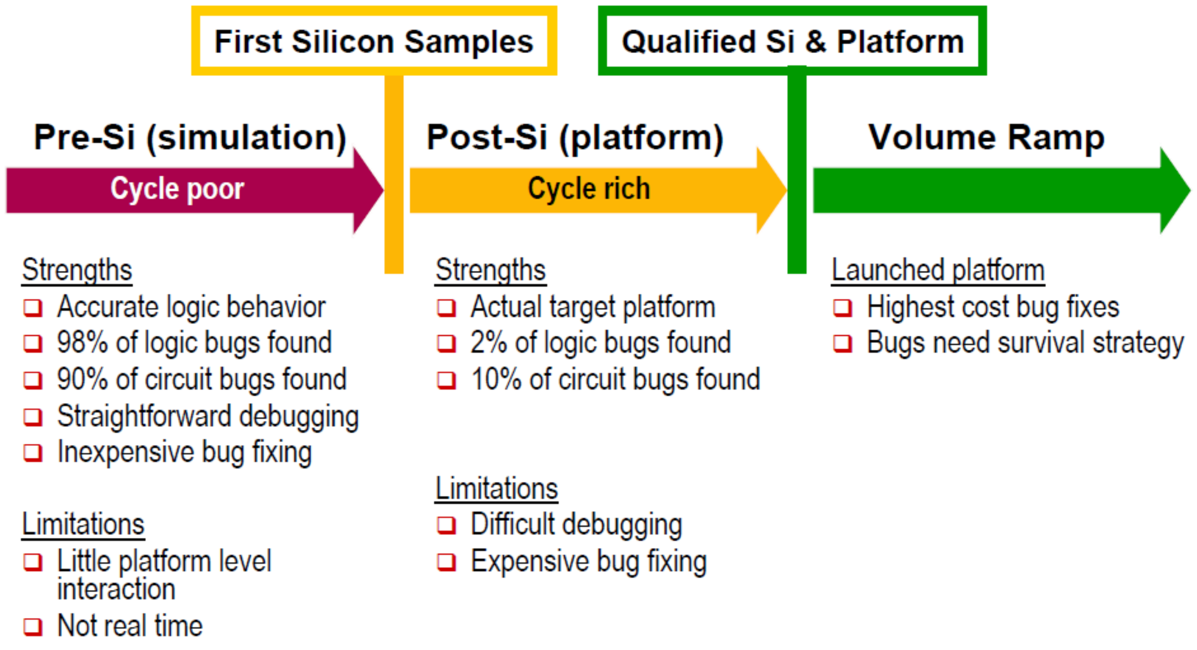


Figure 1: Validation domain and characteristics [3].

As the name suggests, pre-silicon verification refers to the chip verification process done before a real silicon chip exists. This verification stage is usually done using software simulation or emulation of the hardware's design. Pre-silicon validation is focused on exhaustive validation of smaller hardware blocks on shorter time scales. Often referred to as functional verification, it aims to validate the design's functionality before producing a silicon chip, a process called *tape-out*. The current practice for functional verification of complex designs starts with a definition of a test plan, comprised of a large set of *events* that the verification team would like to observe during the verification process. The test plan is usually implemented using random test generators that produce a large number of test-cases, and *coverage* tools that detect the occurrence of events in the test plan. Analysis of the coverage reports allows the verification team to modify the directives for the test generators and to better reach areas or specific events in the design that are not covered well.

In addition to stochastic testing, functional validation today broadly uses formal verification techniques [4]. Equivalence and model checking techniques are used to decide whether a system satisfies a set of properties, usually specified using temporal logic. The latter is now a key component of all industrial formal verification tools. Boolean reasoning models, such as Boolean Satisfiability (SAT) and Binary Decision Diagrams (BDDs), are also central to pre-silicon validation. Formal verification is usually used on small, well defined blocks rather than full systems due to their complexity.

Post-Silicon validation is the validation of the real chip on the board after a hardware prototype is produced. The validation process requires many resources, both machine and human. Compared to pre-silicon, post-silicon tests are much faster, but the internal observability (the ability to monitor many events simultaneously) is very poor. In pre-silicon validation, on the other hand, the situation is the opposite – unlimited observability, but slow testing rate. Hence, post-silicon validation concentrates on more complex system scenarios and protocols. Post-silicon validation aims to generate all possible pertinent scenarios randomly. As in pre-silicon, the test plan is comprised of a set of events that the verification team would like to observe during the verification process. Tests are usually implemented using random test generators that produce test-cases. Because each silicon tape-out is highly costly and the post-silicon validation must precede production, it is important to optimize this process while keeping high coverage of the events according to the test plan.

2.1.2. A formal description of the validation data

Suppose the system under test has a finite set of *configurations* or *parameters* $P = \{p_1 \dots p_k\}$.

A *test* on the system is defined by a set of configurations $T = \{t_1 \dots t_n\}$, such that $\forall t_i \in T \;\; t_i \subseteq P$. We assume that all tests are different, i.e., $\forall t_i, t_j \in T.\;\; i \neq j \rightarrow t_i \neq t_j$.

There is a finite set $E = \{e_1 \dots e_m\}$ of *events* that can occur and can be measured in the system.

Suppose a test t_i runs n_i times. A single *run* of a test t_i is a partial function $r_{ij}: E \rightarrow \mathbb{N}, 1 \leq j \leq n_i$, where $r_{ij}(e_l)$ is the number of times event e_l was observed in test t_i on the j'th run. We assume the function is partial since not all events are measured in a single run of a test. Hence, test t_i is repeated n_i times (with different random seeds) and in each run some of the events are recorded. Notice that if an event is not recorded during a run it does not necessarily mean it has not occurred.

The *results of test* t_i are the set of its runs $R_i = \{r_{i1} \dots r_{in_i}\}$. The *results* of the entire test suite are the collection of all test results $\mathcal{R} = \{R_1 .. R_n\}$.

For each event $e_l \in E$ there is a *threshold* $\tau_l \in \mathbb{N}$. We say that run r_j of a test t_i *covers* or *hits* event e_l if and only if $r_{ij}(e_l) \geq \tau_l$. We say that a test t_i *covers/hits* event e_l if any of its runs covers it, i.e., for some $j, 1 \leq j \leq n_i \;\; r_{ij}(e_l) \geq \tau_l$.

A more compact representation of the test suite results uses a test × event matrix $M \in \mathbb{N}^{n\times m}$. The results of a test t_i are summarized in row $M_{i\cdot} = (e_{i,1} \dots e_{i,m})$, where $e_{i,j}$ is a function of the number of times event e_j occurs in test t_i. Such a matrix can be a derived from $\mathcal{R}$ using some aggregation on the test runs. Such aggregation can be, for example, taking the maximal value of hits in each event in all runs of the test, the number of times the threshold was exceeded, or the average number of hits per run. We may sometimes get the aggregated results matrix instead of $\mathcal{R}$. We refer to the matrix representation of the results M as the *hit matrix*.

2.2. Computational background

In this section we provide basic definitions and background for the computational problems we will discuss in the thesis. For further reading and more details see [5–24].

2.2.1. Basic concepts in graph theory

2.2.1.1. Undirected graphs

Let $G = (V, E)$ be an undirected graph with a finite set of nodes V and a set of unordered pairs of nodes, $E \subseteq V \times V$. We typically use e or (u, v) to denote an edge $e = (u, v)$, and we let $n = |V|$ and $m = |E|$. We assume G has no self loops or parallel edges. Edge weights may be specified by a weight function $W: E \to \mathbb{Q}^+$. When the graph is unweighted we assume all edge weights are 1. We denote the weight of edge $e = (u, v)$ as W_e or $W(u, v)$.

Let $S, T \subseteq V$ be subsets of nodes. We define the weights between S and T as the sum of edges with an end in each subset: $W(S,T) = \sum_{\substack{u\in S \\ v\in T \\ (u,v)\in E}} W(u,v)$. The weight of a subgraph with node set S is the sum of the edge weights inside it: $W(S) = W(S,S) = \sum_{\substack{u,v\in S \\ (u,v)\in E}} W(u,v)$.

The *degree* of a node v is defined as the sum of the weights on the edges incident to v: $d_v = \sum_{(u,v)\in E} w_{(u,v)}$. In case the graph is unweighted, the degree is just the number of edges touching node v. The *complement degree* of a node v is defined as $\overline{d_v} = |\{(u,v) | u \in V, u \neq v, (u,v) \notin E\}|$.

2.2.1.2. Directed graphs and flow networks

Let $G = (V, E)$ be a directed graph with a finite set of nodes V and a set E of ordered pairs of nodes, $E \subseteq V \times V$. We will use similar notation as for the undirected case. We may distinguish two nodes s and t in V as the source and sink, respectively.

A directed *s-t path* in G is a sequence of nodes and edges of the form $s, (s, v_1), v_1, (v_1, v_2), v_2, \dots, v_{k-1}, (v_{k-1}, t), t$. A minimal s-t cut in G is a minimal set of edges C whose removal disconnects s from t in G, i.e., breaks all directed s-t paths. If C is a proper superset of some s-t cut, it is an s-t cut but not a minimal one. The value $W(C) = \sum_{e \in C} W_e$ is the *weight of cut C*. A *minimum cut C_0* is an s-t cut whose weight, $W_0 = W(C_0)$, is minimum among all s-t cuts. All minimum cuts are minimal because edge weights are positive.

An *s-t flow f* in a directed graph G is a function $f: E \rightarrow \mathbb{Q}^+$ where $0 \le f(e) \le W(e)$ for all $e \in E$ and for all $v \in V \backslash \{s, t\}$, $\sum_{u \in V, (u,v) \in E} f(u, v) = \sum_{u \in V, (v,u) \in E} f(v, u)$. The *value* of the flow from s to t is $F = \sum_{u \in V, (s,u) \in E} f(s, u) - \sum_{u \in V, (u,s) \in E} f(u, s)$. In the *maximum-flow problem*, we wish to find a flow f^* that yields a maximum value for F, denoted as F^*. When referring to flow networks, we will use the term *capacity* instead of weight for edges and edge sets.

Given a flow f in graph G, we define the corresponding *residual network* denoted as $G(f)$ or G_f as follows. The residual network has the same nodes as the network G, but has edges with capacities $W_f(u, v) = W(u, v) - f(u, v)$. Only edges with non-zero capacity are included in G_f.

The max-flow min-cut theorem [5] states that the maximum value of the flow from s to t in a network equals the minimum capacity among all $\mathrm{s} - \mathrm{t}$ cuts, i.e., $F^* = W_0$.

The maximum flow problem is well studied and several polynomial time algorithms were developed for it. The first such algorithm due to Ford and Fulkerson, is based on augmenting paths and runs in $O(m\, max|F|)$. An improvement for the augmenting paths algorithm, due to Edmonds and Karp, runs in $O(nm^2)$. Using a different approach called *preflow-push-relabel*, Karzanov achieved a running time of $O(n^3)$. There are many more approaches and algorithms for the maximal flow problem. See [6] for references and much more information.

2.2.2. Covering and domination problems

2.2.2.1. Set cover problem

The *set covering problem* (SCP) [7] is a classical question in optimization. The problem study led to the development of fundamental techniques in the field of approximation algorithms. It was also one of Karp's 21 classical problems shown to be NP-complete [25].

Set cover problem: Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \dots, S_m\}$, and a cost function $c: \mathcal{S} \to \mathbb{Q}^+$, find a minimum cost subcollection of $\mathcal{S}$ that covers all the elements of U, i.e., $C = \operatorname{argmin}_{C` \subseteq \mathcal{S}} \sum_{S \in C`} c(S)$ s.t $\forall u \in U \ \exists S \in C, u \in S$.

The greedy algorithm:

The greedy strategy applies naturally to the set cover problem: iteratively pick the most cost-effective set and remove the covered elements, until all elements are covered. Let C be the set of elements already covered at the beginning of an iteration. During this iteration, define the *cost-effectiveness* of a set S to be the average cost at which it covers new elements, i.e, $\frac{c(S)}{|S \setminus C|}$.

```
Greedy set cover algorithm
$C \leftarrow \emptyset$
$\hat{S} \leftarrow \{\emptyset\}$
While $C \neq U$ do
        Find a set $S \in \mathcal{S} \setminus \hat{S}$ whose cost-effectiveness is smallest.
        $(S = \operatorname{argmin}_{S' \in \mathcal{S} \setminus \hat{S}} \frac{c(S')}{|S' \setminus C|})$
        $C \leftarrow C \cup S$.
        $\hat{S} \leftarrow \hat{S} \cup \{S\}$
Return $\hat{S}$.
```

Algorithm 1: Greedy set cover algorithm

It can be shown that the greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem, where $H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n}$ [7]. In particular $H_n = O(logn)$.

Surprisingly, the greedy algorithm above gives essentially the best approximation factor one can hope for, unless $P = NP$ [7].

2.2.2.2. Domination problems

The matrix domination problem can be thought of as a simple generalization of the set cover problem. To our knowledge, this problem is novel.

Given a matrix M in $\mathbb{N}^{m\times n}$, two rows $1 \le i \neq k \le m$ and a column $1 \le j \le n$, we say that row i *dominates* row k on column j iff $M_{kj} \le M_{ij}$. We say that row i *dominates* row k if it dominates it for all columns.

Matrix row domination: Given a matrix M in $\mathbb{N}^{m\times n}$ and a row cost function $c: \{1 \dots m\} \to \mathbb{Q}^+$, find a minimum cost subset C of the rows of M s.t. every other row is dominated on every column by some row in C, i.e., $C = \mathrm{argmin}_{C` \subseteq \{1\dots m\}} \sum_{i\in C`} c(M_{i\cdot})$ s.t. $\forall i \in \{1 \dots m\}\ \forall j \in \{1 \dots n\}\ \exists k \in C,$ s.t. $M_{kj} \ge M_{ij}$.

We show that the matrix row domination problem is NP-hard by a simple reduction from Set Cover.

Reduction: Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \dots, S_m\}$, and a cost function $c: \mathcal{S} \to \mathbb{Q}^+$ we reduce the problem to matrix domination problem by a reversible mapping between the collection of sets and the matrix rows. We define a matrix M in $\mathbb{N}^{m\times n}$ s.t $\forall i \in \{1 \dots m\}\ \forall j \in \{1 \dots n\},\ M_{ij} = \begin{cases} 1 & if\ j \in S_i \\ 0 & o.w \end{cases}$. Let $c`: \{1 \dots m\} \to \mathbb{Q}^+$ s.t. $\forall i \in \{1 \dots m\}\ \ c`(M_{i\cdot}) = c(S_i)$. Now, $\{i_1 \dots i_k\}$ is a minimum cost dominating set of rows in M if and only if $\{S_{i_1} \dots S_{i_k}\}$ is a minimum cost set cover of U. □

2.2.3. Clustering

Clustering is the challenge of finding and describing cohesive or homogeneous "chunks" in data, called the clusters [9]. The idea behind clustering is rather simple: introduce a measure of similarity between entities under consideration and combine similar entities into the same clusters while keeping dissimilar entities in different clusters.

Formally, let $U = \{e_1, \dots, e_n\}$ be a set of n elements, and let $\mathcal{C} = (C_1, \dots, C_K)$ be a partition of K into subsets. Each subset is called a *cluster*, and $\mathcal{C}$ is called a *clustering solution*, or simply *clustering*. Two elements e_i and e_j are called *mates with respect to* $\mathcal{C}$ if they are members of the same cluster in $\mathcal{C}$.

Given a set of elements U, a distance function between elements $d: U \times U \to \mathbb{R}$ and a number of clusters K, the clustering algorithm aims to partition U into K disjoint clusters such that mates in that clustering are more similar to each other than are non-mates. In some algorithms K is not provided. Some formulations use similarity or proximity metric between elements instead of distance. Clustering formulations vary in the way their objective functions balance between intra-cluster homogeneity and inter-cluster separation. For example, K-means (described in Section 2.2.3.1), minimizes the average distance within each cluster while fixing the number of clusters. The Click algorithm (described in Section 2.2.3.3) presents a probabilistic model and the algorithm tries to maximize the likelihood under the model. Most clustering formulations yield NP-hard problems. For additional clustering methods see, e.g., [10].

2.2.3.1. K-means

The K-means algorithm [9] is one of the earliest clustering heuristics. The algorithm seeks a partition of the entity set into K sets called clusters. Each cluster is represented by its mean vector. More formally, if $U = \{e_1 \dots e_n\}, e_i \in \mathbb{R}^m$ is the set of elements, the cluster structure is represented by a partition into subsets $S_i \subset U$ and m-dimensional *centroids* $c_i \in \mathbb{R}^m$, $i \in \{1, \dots, K\}$, satisfying $c_{ij} = \frac{1}{|S_i|} \sum_{k \in S_i} e_{kj}$.

Given K m-dimensional vectors c_i as cluster centroids, the algorithm updates cluster sets S_i according to the so-called *minimum distance rule*. The minimum distance rule assigns entities to the nearest centroid. Specifically, for each entity $e \in U$, its distances to all centroids are calculated, and the entity is assigned to the nearest centroid. When there are several nearest centroids, the assignment among them is arbitrary. In other words, S_i is made of all such $e \in U$ that $d(e, c_i)$ is minimum over all centroids $\{c_1, \dots, c_K\}$. d measures the distance in the m-dimensional space. For example, when using Euclidean distance $d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$.

```
K-Means
  1. Data preprocessing. Transform the data into a quantitative space and define a distance measure.
  2. Initial setting. Given the number of clusters $K$, choose initial centroids $c_1, \dots, c_K$.
  3. Iteration:
      i.   Cluster update. Given the centroids $c_1 \dots c_k$, determine clusters $S_1 \dots S_k$ using the minimum distance rule.
      ii.  Stop condition. If the cluster assignment did not change following step i, stop.
      iii. Centroid update. Given clusters $S_1 \dots S_k$, calculate $c_1 \dots c_k$ and go back to step i.
```

Algorithm 2: K-means

This algorithm usually converges fast to a local optimum, depending on the initial setting. The choice of the initial centroids may affect not only the speed of convergence but, more importantly, the final results as well.

2.2.3.2. Hierarchical clustering

In *hierarchical clustering* [9] we wish to present our data in the form of a hierarchy over the entity set. This hierarchy is represented by a rooted tree in which each node is the union of its children.

Figure 2 shows an example of a hierarchical tree. The vertical height axis represents the similarity between clusters. The tree assumes a constant distance between the root and the leaves. Such a tree is called a *dendrogram*.

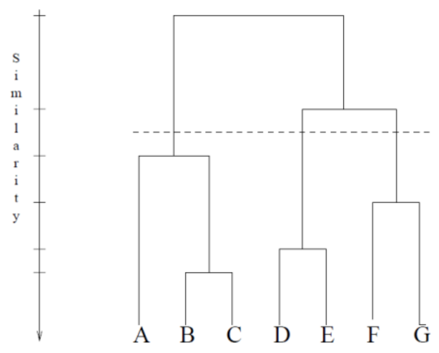


Figure 2: A dendrogram generated by a hierarchical clustering algorithm [10].

There are two approaches to building a cluster hierarchy:

- *Agglomerative* clustering methods build a hierarchy in a bottom-up fashion by starting from smaller clusters and sequentially merging them into 'parental' nodes.
- *Divisive* clustering methods build a hierarchy top-down by splitting greater clusters into smaller ones starting from the entire data set.

The agglomerative approach in clustering builds a cluster hierarchy by merging two clusters at a time, starting from singletons (one-entity cluster) or other pre-drawn clusters. Thus, each non-singleton cluster in the hierarchy is the union of two smaller clusters, and the whole hierarchy can be drawn as a binary tree. The singletons and their successive merges at every intermediate step form a cluster hierarchy, until the root is reached, at which point the full cluster hierarchy emerges.

In addition to the tree topology, hierarchical clustering also specifies edge lengths, reflecting distance (dissimilarity) between sets. Some algorithms assume all leaves have the same distance from the root and then parent nodes are equally distant from their children (Figure 2). Other algorithms do not assume that.

At each step of an agglomerative clustering algorithm, a set of already formed clusters S is considered along with a matrix of distances between the clusters in S. Then two closest clusters are merged and the newly formed cluster is assigned distances from the other clusters. A clustering (partition) solution can be derived from a hierarchy in different ways, e.g., by trimming the tree at a certain height and taking the top cluster nodes.

Agglomerative algorithms differ depending on between-cluster distance measures used in them and on the rule for identifying closest clusters.

UPGMA algorithm

UPGMA [11] is an algorithm for constructing a dendrogram clustering solution (see Figure 2). Let d be the distance function between two elements. We define the distance $D_{i,j}$ between two clusters C_i ,C_j with sizes n_i ,n_j respectively, as follows:

$$D_{i,j} = \frac{1}{n_i + n_j} \sum_{p \in C_i} \sum_{q \in C_j} d(p,q)$$

The distance from a new cluster $C_{(ij)}$ formed by joining C_i and C_j, to all other clusters can be computed as a weighted average of the distances from its components:

$$D_{(ij),k} = \left(\frac{n_i}{n_i + n_j}\right) D_{i,k} + \left(\frac{n_j}{n_i + n_j}\right) D_{j,k}$$

The new node formed by merging clusters i and j is connected to both i and j by branches of length $D_{i,j}/2$. UPGMA picks C_i, C_j with the smallest distance D_{ij} in each iteration.

Neighbor Joining Algorithm

Neighbor joining [12] aims to produce a rooted tree with branch lengths without assuming equal root-leaf distances. The input is the distance matrix between elements. Initially each element is a cluster. At each iteration, the algorithm identifies two nodes that are guaranteed to be neighbors in the current tree (i.e., nodes with a common parent node), and merges them to form a new cluster. It then computes the distances from the new cluster. When the algorithm finishes, we represent the results as an edge weighted tree. The tree is not required to be a dendrogram. If there exists a tree such that distances between leaves match the matrix distances, the algorithm is guaranteed to find it.

Figure 3 shows an example of a tree generated by the neighbor joining algorithm. The elements are marked 1-8 and internal nodes A-F. Numbers on the edges represent branch lengths.

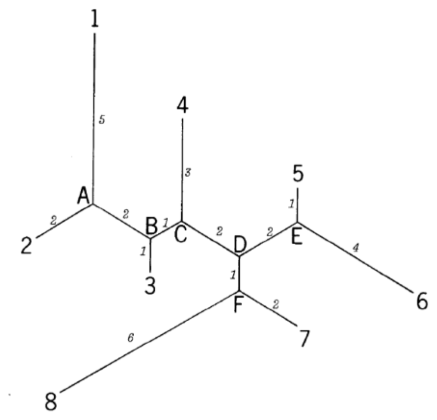


Figure 3: Dendrogram the neighbor joining algorithm [12].

2.2.3.3. Click

Click (CLuster Identification via Connectivity Kernels) [13] is a graph-based algorithm for clustering. The input for Click is a similarity matrix between elements. The Click algorithm attempts to find a partition of the set of elements into clusters, so that two criteria are satisfied: *homogeneity* - pairs of elements from the same cluster are highly similar to each other; and *separation* - pairs of elements from different clusters have low similarity to each other. Unlike conventional clustering algorithms, Click allows some elements to remain un-clustered. Un-clustered elements, referred as *singletons,* should be dissimilar to any of the clusters found.

Click initially identifies highly homogeneous and well-separated sets of elements called *connectivity kernels,* which are subsets of very similar elements. The remaining elements are subsequently added to the kernels by the similarity to kernel centroids.

Probabilistic Model

The Click algorithm makes the following assumptions:

- Similarity values between mates are normally distributed with parameters μ_T, σ_T.
- Similarity values between non-mates are normally distributed with parameters μ_F, σ_F.
- Similarity values are mutually independent.

For clusters to be identifiable, these parameters must also satisfy $\mu_T > \mu_F$, and σ_T, σ_F should be small enough compared to $\mu_T - \mu_F$.

Basic Click Algorithm

The Click algorithm represents the input data as a weighted *similarity graph* $G = (V, E)$. In this graph, nodes correspond to elements and edge weights are derived from the similarity values. The weight w_{ij} of an edge (i, j) reflects the probability that i and j are mates, and is set to be:

$$w_{ij} = ln\frac{pf^M(S_{ij})}{(1-p)f^N(S_{ij})}$$

where p is the probability of two genes to be mates, and $f^M(S_{ij})$ $(f^N(S_{ij}))$ is the value of the probability density function for mates (non-mates) for S_{ij}. According to our assumptions $f^M \sim N(\mu_T, \sigma_T)$, $f^N \sim N(\mu_F, \sigma_F)$.

The main idea of the algorithm is as follows: given a connected graph G, we would like to decide whether $V(G)$ is a subset of some true cluster, or $V(G)$ contains elements from at least two true clusters. In the first case we say that G is *pure*. In order to make this decision, we would like to test the following two hypotheses for every possible cut C in G:

H_0^C: C contains only edges between non-mates.

H_1^C: C contains only edges between mates

G is declared a *kernel* if H_1 is more probable for all cuts. The decision whether G is a kernel relies on the following theorem:

Theorem [13]: *G is a kernel iff the weight of $MinCut(G) > 0$.*

Following the theorem, the basic algorithm splits G recursively using min-cut computations until a kernel or a singleton is reached. This is done heuristically since edge weights can be negative, and the MIN-CUT problem for a weighted graph with both positive and negative edges is NP-Complete.

Click refinements

The Basic-Click algorithm divides the graph into kernels and singletons. These kernels are expanded to the full clustering, using several heuristic refinements:

- *Adoption Step*: kernels "adopt" singletons to create larger clusters.

- *Cleaning Step*: removing from clusters nodes having a low degree.
- *Merge Step*: merging clusters whose centroids are similar.

2.2.3.4. Performance measures

When a correct solution for a clustering problem is known, we can evaluate an algorithm's performance by measuring how close the true and the algorithm's solution are. Let S, T be two clustering solutions. We mark by n_{11} the number of pairs of elements that are mates in both S and T, n_{01} is the number of pairs that are mates only in S, and n_{10} is the number of pairs that are mates only in T. The Jaccard coefficient is defined by:

$$J = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$$

The value is 1 iff the two solutions are identical. The similarity of the solutions improves as the value of the coefficient increases.

Unfortunately, in most cases the "correct" solution for the clustering problems is unknown. In these cases, a clustering solution will be considered good if it provides tight clusters that are well separated from each other.

The input to the clustering problem is U a set of N elements and a *fingerprint* function mapping elements in U to vectors in $\mathbb{R}^m$. We define the *fingerprint* of a set of elements to be its centroid, i.e., the coordinate-wise mean vector of the fingerprints set members. Let $C_1, \ldots, C_K$ be clusters, $C(u)$ be the cluster of element u, $F(C)$ be the fingerprint of a cluster C, and let $S(F(u), F(v))$ denote the similarity between two fingerprints.

The average homogeneity of a clustering solution $\mathcal{C} = (C_1, \ldots, C_K)$ is defined as:

$$H_{Ave}(\mathcal{C}) = \frac{1}{|U|} \sum_{u \in U} S(F(u), F(C(u)))$$

The minimum homogeneity of $\mathcal{C}$ is defined as:

$$\mathrm{H_{Min}}(\mathcal{C}) = \min_{\mathrm{u} \in \mathrm{U}} \mathrm{S}\left(\mathrm{F(u)}, \mathrm{F(C(u))}\right)$$

The average separation of $\mathcal{C}$ is defined as:

$$S_{Ave}(\mathcal{C}) = \frac{1}{\sum_{i \neq j} |C_i||C_j|} \sum_{i \neq j} |C_i||C_j| S(F(C_i), F(C_j))$$

The maximum separation of $\mathcal{C}$ is defined as:

$$S_{Max}(\mathcal{C}) = \max_{i \neq j} S(F(C_i), F(C_j))$$

A clustering improves when H_{Ave} and $\mathrm{H_{Min}}$ increase, and when S_{Ave} and S_{Max} decrease.

Another way to evaluate a clustering solution is the silhouette method [14], which is based on comparison of the cluster tightness and separation. The method can evaluate clustering validity and could also be used to determine the number of clusters. The *silhouette* of each element $u \in U$ is defined as:

$$S(u) = \frac{b(u) - a(u)}{\max\{a(u), b(u)\}}$$

Where $a(u)$ is average dissimilarity of u to other elements in the same cluster; $b(u)$ is the average dissimilarity of v to elements in the closest cluster. The silhouette index ranges between -1 and 1. The silhouette of a clustering solution is the average silhouette over all elements. The clustering solution is better when its silhouette is close to 1. A negative silhouette index means that an element is more similar to a different cluster than to the cluster it is in.

2.2.4. Statistical scores

In this section we describe some statistical tests that will be used in our study.

2.2.4.1. Hypergeometric score

The *hypergeometric* distribution describes the probability of drawing k white balls in n draws without replacement from a finite population N containing m white balls and $N - m$ black balls. The probability mass function is defined as:

$$P(X = k | N, m, n) = \frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$$

The statistical significance (or *P-value*) of a draw of k white balls is the probability to get at least k white balls under the hypergeometric distribution. More formally: $P-value = \sum_{i\geq k} P(X=i)$. The lower this probability is, the less likely the model is given the data.

2.2.4.2. FDR correction

When performing many statistical tests, correction for multiple testing is required. Bonferroni correction [26] may be too harsh if we are willing to accept few false positive results. The standard approach is called FDR (*False Discovery Rate*) [15], and it ensures that the expected fraction of false positives, out of all accepted tests, would remain low.

Let $H_1, \dots, H_N$ be the null hypotheses and $p_1, \dots, p_N$ their corresponding p-values ordered in increasing order. For a given threshold α we reject the hypotheses with the lowest p-values such that $p_i \leq i\frac{\alpha}{N}$. Under some mild assumptions, the procedure guarantees that the false positives fraction will not exceed α.

2.2.4.3. Mantel test

The Mantel test [16], [17] is a technique to estimate the resemblance between two proximity matrices computed for the same elements. The matrices must therefore be of the same dimensions, but not necessarily symmetric though this is often the case.

The test is based on the correlation between the two similarity matrices. To enable correlation calculation each matrix is transformed into a vector row by row. For symmetric matrices, such as similarity matrices, only the upper (or lower) triangle is needed.

When applied to similarity matrices, the test assumes that only the rank order of elements in the two matrices is important for clustering, i.e., clustering algorithm are likely to be more sensitive to the order of similarities than to their actual values. In order to measure correlation between two proximity matrices, the nonparametric Spearman rank correlation statistic, denoted ρ_M, is used. Hence, the test statistic is the rank correlation between corresponding similarity values of the row vectors of matrices X and Y.

Given two similarity matrices, first, an observed Mantel statistic, ρ_M, is calculated. Second, to estimate the p-value, the elements (rows and columns) within one of the matrices are subjected to random permutations, and ρ_M is recalculated for each random permutation. The significance is empirically estimated as the proportion of permutations that lead to a value of ρ_M that is equal to or higher than the original observed correlation.

2.2.5. Graph clustering and density

Graph clustering is the task of grouping the nodes of the graph into clusters taking into consideration the edge structure of the graph in such a way that there should be many edges *within* each cluster and relatively few *between* the clusters. In this section we focus on a specific clustering objective function named density.

2.2.5.1. Densest subgraph problem

The *density* of a graph is defined as the ratio of number of edges to the number of nodes in the graph. The definition can also be generalized to handle weighted edges. Formally, given an undirected graph $G = (V, E)$ and a weight function over the edges $W: E \rightarrow \mathbb{Q}^+$, the *density* of a subgraph on a node set S is defined as $D(S) = \frac{W(E(S))}{|S|}$, where is $E(S)$ is the set of edges in the subgraph induced by S and the weight of an edge set is the sum of the weights all edges in it.

The *densest subgraph problem* receives an input a weighted graph and seeks a subset of nodes $\hat{S}$ of maximum density. The optimum density is denoted as $D^* = \max_{S \subseteq V} D(S)$.

The densest subgraph problem can be solved optimally in polynomial time based on an elegant network flow formulation proposed by Goldberg [18]. Charikar [19] gave a linear programming formulation to this problem. He also showed that we can find a 2-approximation to the densest subgraph problem in linear time using a very simple greedy algorithm. Saha et al. [20] used a different flow construction to solve the problem.

When a size constraint is specified, namely, when the goal is to find a densest subgraph of exactly/at least/at most k nodes, the problems become NP-hard [27]. There is a very simple, linear time, greedy 3-approximation algorithm for the densest subgraph with at least k nodes [27]. A 2-approximation can be achieved for the same problem using flow computation or linear programming [22]. For the densest subgraph with exactly k nodes there is a polynomial approximation factor [23]. It was shown that the problem does not have any PTAS under reasonable complexity assumptions [24]. For the densest subgraph with at most k nodes approximation complexity is not yet known, but an approximation bound follows from the problem of densest subgraph with exactly k nodes [22].

2.2.5.1.1. Goldberg's algorithm [18]

Goldberg's algorithm reduces the problem of finding a maximum density subgraph to a series of minimum cut problems, which in turn can be solved using network flow techniques. The algorithm requires $log(n)$ min-cut computations on networks with $n + 2$ nodes. We present here the unweighted version of the algorithm but it can easily be generalized to the weighted case as we show in Chapter 7.

The algorithm works as follows: Let D be the density of the desired subgraph. At each stage of the algorithm there is a guess $g > 0$ for D. Then a network is constructed and a minimum cut computation enables decision on whether $D \leq g$ or $g \leq D$. Using binary search on the interval of possible values of D, within $\log(n)$ iterations the algorithm finds a maximum density subgraph.

Let d_i be the degree of node i of G. Given a guess g, we convert G into a network $N = (V_N, E_N)$ as follows: We add a source node s and a sink node t to G; replace each (undirected) edge of G by two directed edges of capacity 1 each; connect the source s to every node i of G by an edge of capacity m; and connect every node i of G to the sink t by an edge of capacity $(m + 2g - d_i)$. Figure 4 illustrates the construction.

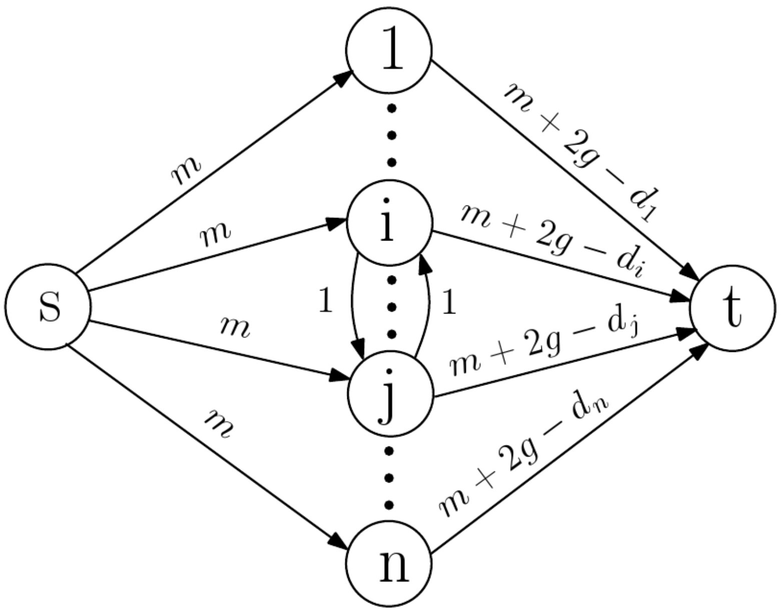


Figure 4: Goldberg's flow network for detecting densest subgraph. Nodes $1, \dots, n$ are the original graph nodes and s, t are added as a source and a sink respectively.

Notice that all capacities are non negative because for any i, $d_i \leq m$ and our guess g will always be non negative.

A partition of V_N into two sets, S and T, such that $s \in S$ and $t \in T$, determines an s-t cut. Let $V_1 = S \setminus \{s\}$, and $V_2 = T \setminus \{t\}$. If $|V_1| = 0$, then the capacity of the cut $c(S,T) = mn$; otherwise, the capacity of the cut is given by (see Figure 5):

$$c(S,T) = \sum_{i \in S, j \in T} c_{ij} = \sum_{j \in V_2} c_{sj} + \sum_{i \in V_1} c_{it} + \sum_{i \in V_1, j \in V_2} c_{ij} =$$

$$= m|V_2| + \left(m|V_1| + 2g|V_1| - \sum_{i \in V_1} d_i \right) + \sum_{\substack{i \in V_1, j \in V_2 \\ (i,j) \in E}} c_{ij} =$$

$$= m|V| + 2|V_1| \left(g - \left(\frac{\left(\sum_{i \in V_1} d_i - \sum_{i \in V_1, j \in V_2} 1\right)/2}{|V_1|} \right) \right)$$

Notice that $\left(\sum_{i \in V_1} d_i - \sum_{i \in V_1, j \in V_2} 1\right)/2$ is the number of edges in the subgraph of G induced by V_1, so

$$D_1 = \frac{\left(\sum_{i \in V_1} d_i - \sum_{i \in V_1, j \in V_2} 1\right)/2}{|V_1|}$$

is the density of the subgraph of G generated by V_1. Therefore, $c(S,T) = m|V| + 2|V_1|(g - D_1)$. The following theorem gives a way to tell whether g is too large or too small.

Theorem (Goldberg,84) [18]. Assume that S and T give a minimum cut. If $|V_1| \neq 0$, then $g \leq D$; *If* $|V_1| = 0$ (i.e. S={s}) then $g \geq D$. (Proof omitted)

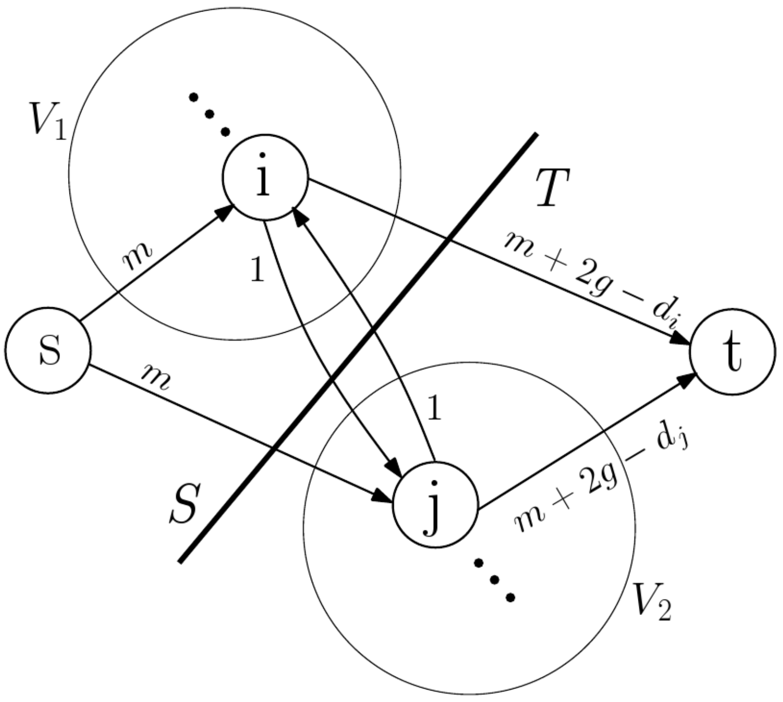


Figure 5: An s-t cut in the flow network.

The maximum density D^* lies between 0 and m; furthermore, the smallest distance between two different possible values of D^* is at most $\frac{1}{n^2}$. Hence, if G' is a subgraph of G with density D', and no subgraph of G has a density greater or equal to $D' + \frac{1}{n^2}$, then G' is a maximum density subgraph.

Now we can describe an algorithm to find the maximum density subgraph.

```
$l \leftarrow 0; u \leftarrow m; V_1 \leftarrow \emptyset;$
while $u - l \geq \frac{1}{n^2}$ do
  begin
        $g \leftarrow \frac{u-l}{2};$
        Construct N=($V_N$,$E_N$);
        Find min-cut (S,T);
        If $S = \{s\}$ then $u \leftarrow g$
          else
            begin
                $l \leftarrow g;$
                $V_1 \leftarrow S \setminus \{s\};$
            end;
  end;
return (*subgraph of G induced by $V_1$*)
```

Algorithm 3: Goldberg's algorithm for finding a maximum density subgraph

Running time: Let $M(\lambda, \varphi)$ be the time required to find a minimum capacity cut in a network with λ nodes and φ edges. The only loop is executed $\lceil \log((m+1)n^2) \rceil = O(logn)$ times. The flow network contains $n+2$ nodes and $2m+2n$ edges. Hence, the algorithm runs in time $O(M(n, n+m)logn)$.

2.2.5.1.2. The algorithm of Saha et al. [20]

The algorithm of Saha et al. [20], like Goldberg's, finds a densest subgraph using a series of min-cut computations. The algorithm guesses the density of the maximum density subgraph and then refines the guess by a network flow computation. The binary search and stop criterion are exactly the same as Goldberg's. The only difference is in the flow network construction, as described below.

Create a flow network G' with a source node s and sink node t. In G' we have a node corresponding to each edge in G (call this set E') and a node corresponding to each node in G (call this set V'). Add edges from s to $e \in E'$ of capacity $W(e)$ and an edge from $v \in V'$ to t

with capacity α, where α is a guess for the maximum density. Add edges from $e = (x, y) \in E'$ to both $x \in V'$ and $y \in V'$ with capacity ∞.

The general idea of this method is that the nodes that correspond to edges between nodes on the same side of the cut must also be on that side. If the guess is lower than the optimal density there will be a minimum cut other than the trivial one separating s from the rest of the G'. Otherwise the minimum cut weight would be the same as the trivial. We omit the full proof.

Saha et al. use their network construction to calculate the densest subgraph with annotation based distance restrictions. We note that they could have used Goldberg's construction for the same purpose. An apparently unique feature of their approach is that it allows to solve the problem subject to the additional constraint that the desired subgraph must contain a given node set. In a different effort, they give an algorithm to compute all ϵ-close to densest subgraphs. We note that Goldberg's construction with a different enumeration algorithm can also be used to compute all ϵ-close to densest subgraphs.

2.3. Previous studies on post-silicon test optimization

Post-silicon validation is in broad practical use by all chip developers. It can be credited with finding of many functional bugs that escaped pre-silicon verification. However, in general, functional verification methodology for post-silicon is still less varied and mature than for pre-silicon platforms. Very little is published on post-silicon verification methodologies (e.g., [28]), and most research in post-silicon validation has centered on on-line checking and debugging capabilities of the silicon platforms (e.g., [29]).

Pre-silicon validation, on the other hand, received a lot of research attention. Pre-silicon validation is based on a well-established methodology of coverage-driven verification (CDV [30]). A verification plan comprises a large set of features in the Design Under Verification (DUV) that need to be verified; random stimuli generators directed towards the verification goals using test-templates [31] (i.e., general specifications of the desired test structure and properties); and coverage analysis tools [32] that detect the occurrence of events in the verification plan and provide feedback regarding the state and progress of the verification process.

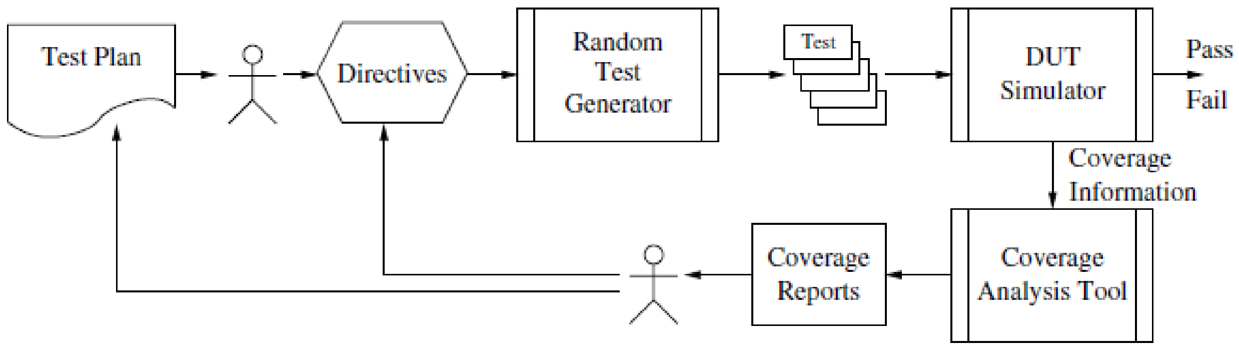


Figure 6: Functional verification process with automatic test generation [33].

Coverage directed test generation (CDG) has been previously studied [33–36]. Although it mainly focuses on functional (pre-silicon) verification, the CDG methodologies can be employed in post-silicon verification as well. The current practice for functional verification of complex designs (Figure 6) starts with a definition of a test plan, comprised of a large set of events that the verification team would like to observe during the verification process. The test plan is usually implemented using random test generators that produce a large number of test cases, and coverage tools that detect the occurrence of events in the test plan, and provide information related to the progress of the test plan. Analysis of the coverage reports allows the verification team to modify the directives for the test generators and to better "hit" areas or specific tasks in the design that are not covered well. The analysis of coverage reports, and their translation to a set of test generator directives to guide and enhance the implementation of the test plan, result in major manual bottlenecks in the otherwise highly automated verification process. Considerable effort is invested in finding ways to close the loop of coverage analysis and test generation. CDG is a technique to automate the feedback from coverage analysis to test generation. The main goals of CDG are to improve the coverage progress rate, to help reaching uncovered tasks, and to provide many different ways to help reach a given coverage task.

Fine and Ziv [33] suggested the use of *Bayesian networks* to generate many different test-cases, each leading to different coverage tasks. Bayesian networks offer an efficient modeling scheme by providing a compact representation of the complex (possibly stochastic) relationships among the CDG ingredients, together with the possibility to encode essential domain knowledge. The CDG process begins with the (manual) construction of a Bayesian network model that describes the relations between the test directives and the coverage space with possible hidden nodes that affect the inner design knowledge. After the Bayesian network structure is specified, it is trained using a sample of directives and the respective coverage tasks. Learning algorithms are used to estimate the Bayesian network's parameters. In the evaluation phase, the trained Bayesian network can be used to determine directives for a desired coverage task, via posterior probabilities queries. A later work [35] tries to automatically construct a data-driven CDG engine based on Bayesian networks, aimed at providing coverage boosting with minimal human effort.

Clustering is not a new idea in the verification studies. Following their work on CDG, Fine and Ziv [36] tried to enhance the verification process using clustering techniques. They use clustering to enhance the efficiency of the CDG process by focusing on sets of non-covered events, instead of one event at a time. They also try to find the correct number of clusters. In a different work [37], they attempt to improve the coverage process efficiency by clustering together related events and generating one set of directives that attempts to cover all events in the same cluster.

3. The tested data

In this section we describe the industrial data example that we analyzed in this thesis.

3.1. The analyzed data sets

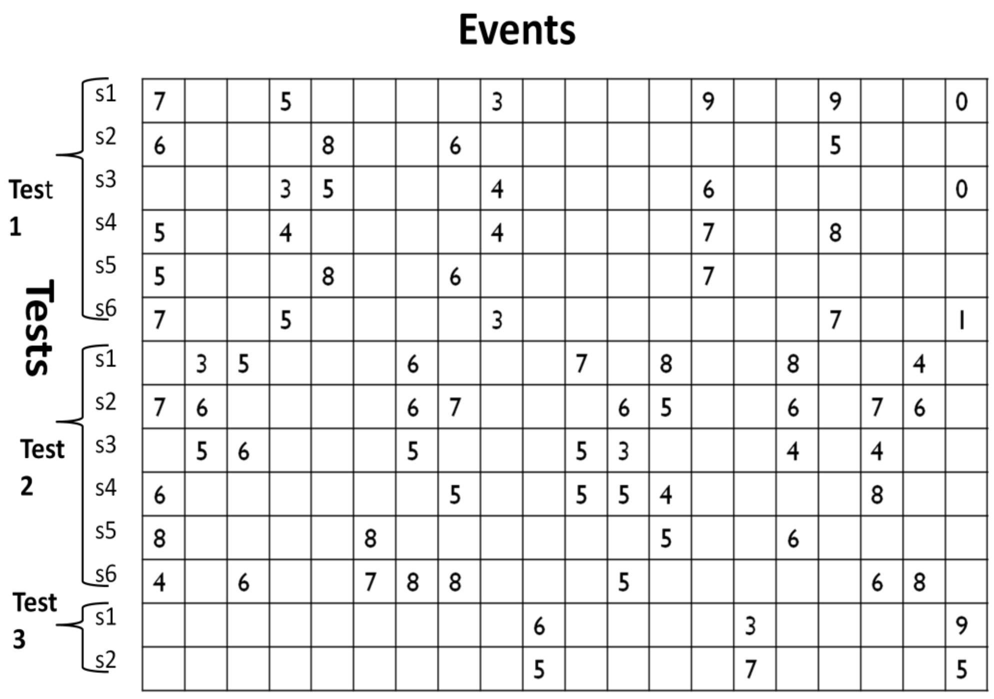
Recall the formulation in Section 2.1.2. The industrial data example is available as a set of tests (T), each having a set of parameters (P) defining how it was performed and what its results were:

- Configs: for each test, a list of config files it runs with. There are ~250 configs. On average each test has ~70 configs.
- System-elements: for each test, a list of the system elements it runs on. There are ~2700 system elements. On average each test has ~600 system elements.
- Modifications: for each test, a list of mod (modification) files it runs with.

In this industrial example, the parameter set (P) is actually comprised of three sets. Together, the three sets of configs, system-elements and mods uniquely define a test.

Additional two data bases provide the events and results:

- Events (E, τ): each event e is accompanied by a threshold τ_e. An event is considered hit or covered by a test run if it occurred at least τ_e times in that run.
- Results ($\mathcal{R}$): The results are summarized by a table with one dimension corresponding to (test, seed) combinations and the other corresponding to events (see Figure 7). For some combinations of test, event and sample (seed run), the table holds the value of the event's counter. There are 874 distinct tests and 302 events. Some tests have a lot of seeds (thousands) and some only a few (or 1). Notably, for any specific run (seed) of the test, not all the counters were measured. There are 7 million counter reads in the table. The counter's value depends on the amount of time that the test ran, which may be different from test to test (and currently is not available).



Tests		Events																			
Test 1	s1	7			5					3					9			9			0
	s2	6				8			6									5			
	s3				3	5				4					6						0
	s4	5			4					4					7			8			
	s5	5				8			6						7						
	s6	7			5					3								7			I
Test 2	s1		3	5				6				7		8			8			4	
	s2	7	6					6	7				6	5			6		7	6	
	s3		5	6				5				5	3				4		4		
	s4	6							5			5	5	4					8		
	s5	8					8							5			6				
	s6	4		6			7	8	8				5						6	8	
Test 3	s1										6					3					9
	s2										5					7					5

Figure 7: An example of the matrix summarizing the test results.

3.2. Summarizing the raw test data

When taking the average event hit count for each test over all its seeds and filtering counter values below the threshold, we obtain a matrix M with 718 tests, 105 events and 8930 counter values. This means that only $\frac{8930}{718*105} = 11.8\%$ of the values in the hit matrix are non zero.

When taking the maximum (instead of average) over all seeds for each test, we obtain a matrix M' with 722 tests, 108 events and 9059 counter values (11.6%).

This suggests that the hit distribution over the different seeds is relatively small. Therefore there is high redundancy in the test seeds and perhaps time can be saved by using less seeds.

We can transform M' to a binary coverage matrix, in which $M'_{ij} = 1$ if test i covers event j, and otherwise $M'_{ij} = 0$. In the binary coverage matrix, each event is covered on average by 12.5 tests, with a maximum of 44 and a minimum of 1. Each test covers on average 84 events with a maximum of 388 and a minimum of 1.

Our notion of coverage is that a test covers an event if any of its runs covers it. Hence, M' was used as the data matrix for further analysis. Values that are below their threshold are discarded and treated as 0 hit count or uncovered.

4. Analysis using covering and domination techniques

Recall that we say that a test *covers* an event if the hit counter value exceeds the threshold. We seek to find small subsets of the tests that achieve similar event coverage as the whole set of tests. In the simplest type of cover we require each event separately to be covered by the selected subset. On the other hand, one might be interested in tests where two events happen together. In that case the subset selected should cover every pair of events that is covered together by some test in the whole set of tests. A generalization of the covering objective beyond looking at pairs of events is to find a subset of the tests that will cover all possible event combinations that are hit together by some test.

4.1. Single event cover and domination

4.1.1. Set cover

In the set cover problem we seek a minimum subset of the tests that hits all events (see Section 2.2.2.1). In this formulation there is no use of the actual hit count. When applying the greedy algorithm (Algorithm 1) to the sample data, it gave a subset of 18 tests that cover all events. Although the greedy algorithm guarantees only an approximation (Section 2.2.2.1) it happens to give on these data the same results as solving the full integer linear programming problem.

4.1.2. Domination

Recall the problem definition in Section 2.2.2.2. We say that test A *dominates* test B on event e if the event's counter value in test A is at least as high as B's (see Figure 8). The goal is to find a minimum subset S of the tests, such that for each event in any test there is a test in S that dominates it. On the sample data, the greedy approach gave a subset of 45 tests. The greedy algorithm, in each iteration, adds a test that covers the maximal number of uncovered test-event entries. A similar analysis to the set cover greedy algorithm (see Section 2.2.2.1) gives this algorithm an approximation ratio of $OPT * H_m$, where OPT is the minimum number of tests required to dominate all events and m is the number of events. When looking only on binary entries instead of hit count, the problem is identical to set cover.

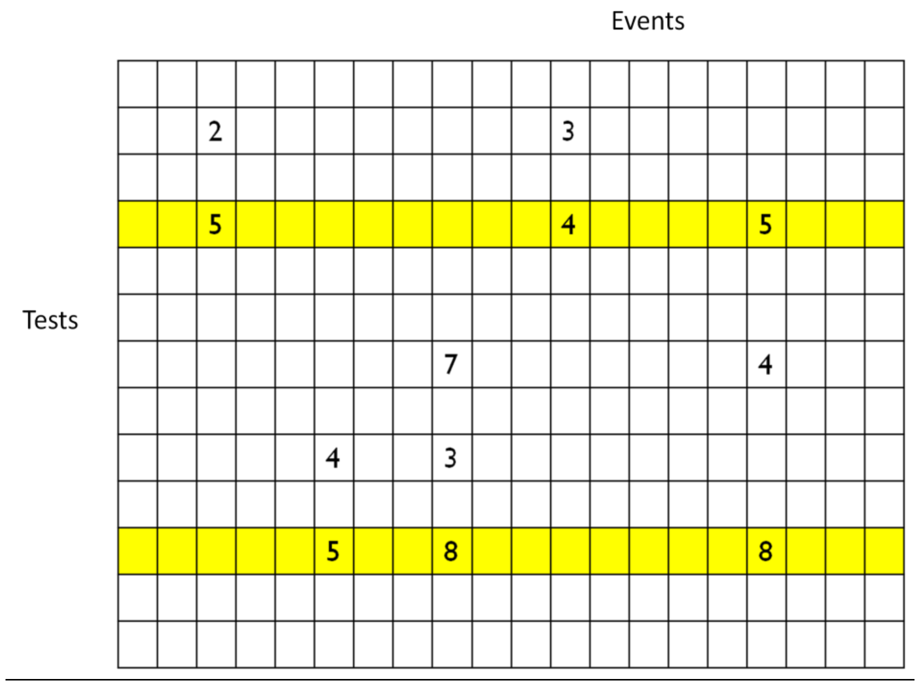


Figure 8: An example of a dominating set. Each white entry has a yellow entry in its column that is equal or higher. Therefore the yellow rows form a dominating set.

4.2. Event pair cover and domination

In this problem we seek to cover or dominate all possible event pairs. Out of all possible pairs $\binom{108}{2} = 5778$, only 3559 pairs are covered by one or more set in the whole set of tests.

4.2.1. Set cover

This problem can be formulated as a set cover problem where the universe to be covered is the set of event pairs. On the sample data, a greedy approach gives a set of 77 tests that covers all 3559 event pairs.

4.2.2. Domination

In this problem version we want to find a test set dominating all event pairs. In other words, we seek a subset of the tests such that each event pair will be dominated together by a test in the subset (see Figure 9). On the sample data set, a greedy approach found a set of 240 tests that dominates all event pairs.

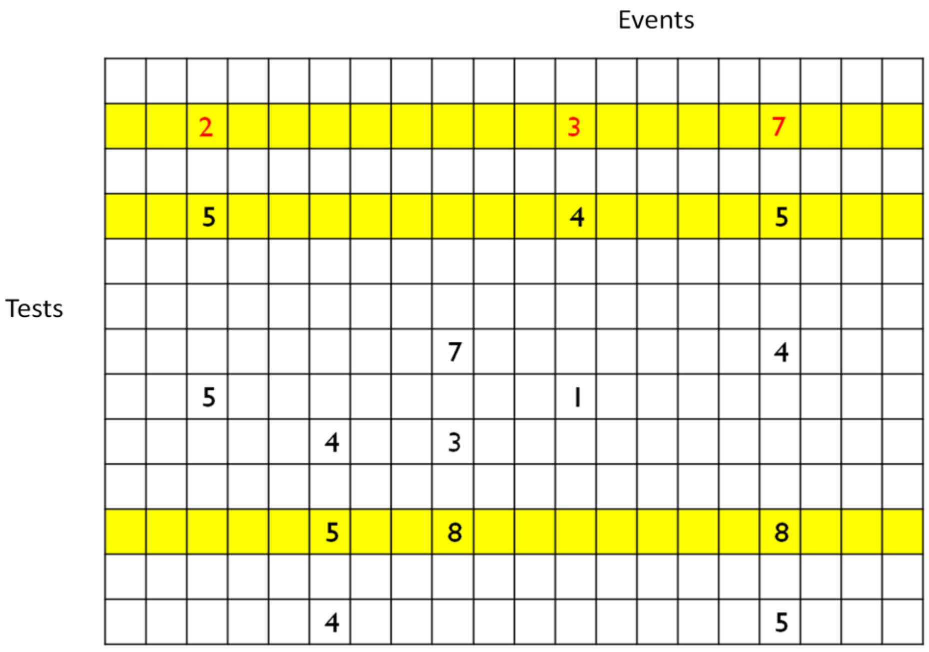


Figure 9: An example of an event pair dominating set. The yellow set of rows 4 and 11 is not a pair dominating set since the pair of events with values 3, 7 in row 2 is not dominated by any other single row. Rows 2, 4, 11 constitute a pair dominating set.

4.3. All subset cover

We say that a test A *dominates* test B if for every event that B hits k times, A hits the same event at least k times. Notice that if test A is dominated by test B, A cannot contribute to any type of cover better than B can. Thus, by removing all dominated tests we remain with a subset that has the same coverage as the whole set on all possible event combinations. This algorithm can be implemented straightforward in $O(n^2m)$ time. Notably, finding the subset that covers all possible event combinations is the only polynomial problem out of all coverage techniques we discussed.

On the sample data set, this strategy gives a subset of 473 tests that are un-dominated. When discarding the actual hit count and looking on a binary cover matrix, domination becomes containment ($\subseteq$) and gives a subset of 291 tests.

5. Prioritizing tests

One of our main goals is to reduce the complexity of the post silicon validation effort. One strategy is to prioritize the tests by ranking them according to their importance. Validation engineers can then opt not to check the whole test suite and focus initially on the important tests.

5.1. Performance criteria

Evaluation of a subset of tests can be defined using several criteria. Maximizing coverage percentage of the subset, as described in Section 4, is one criterion. Another goal is maximizing the average (or minimum) number of times each event is covered by the subset. A desired property of a good subset is heterogeneity, so that the tests would be different from one another.

5.2. Greedy approaches

A simple test ranking can be done in a greedy fashion, by either adding or removing the best/worst test at each stage. The utility of the considered test can be evaluated according to its similarity to other tests or its relative added coverage. A clustering solution can also be used to this end.

A simple incremental greedy algorithm that aims to maximize the coverage selects a test that covers an event that is the least covered at that stage. If the allowed number of tests is equal or larger than the number of events then clearly we get a set cover problem.

The *k-center problem* seeks to find a node subset S of size k in an edge-weighted graph such that the maximal distance from any node to S would be minimized. A 2-approximation algorithm is achieved by selecting the farthest node from those already chosen at each stage [38]. Inspired by this method, a similar approach is suggested here. Start with an arbitrary or a known subset S (a cover, for example). Add the test least similar to S. A different version can start with all the tests and remove the one most similar to another. Dissimilarity to a group can be defined as the average dissimilarity to its elements, or as the minimal dissimilarity to an element. This method aims to minimize the homogeneity.

One-class SVM is a variation of SVM classification due to Chen et al. [39]. The strategy is to map the data into the feature space and then use a hyper-sphere to describe the data in feature space so that most of the data points fall in the hypersphere. This can be formulated as an optimization problem. We want the ball (hypersphere) to be as small as possible while at the same time including most of the training data. The tradeoff between the radius of the ball and the number of training samples that it can hold is set by the parameter $\nu \in [0,1]$. When ν is small, we put more data points into the ball. When ν is larger, we reduce the size of the ball.

Our one-class SVM ranking algorithm works as follows: In each stage a one-class SVM is built for the remaining tests using a dot-product similarity kernel. The outliers of the SVM solution are added to the test set, since they represent different patterns from the main set. By selecting the SVM error ν we control the number of tests added in each stage. In our implementation, we add 50 tests each time until a total of n tests are selected.

5.3. Evaluations

In the following experiment, for each value of n we selected n tests out of the entire test set (718). The methods mentioned above were used in addition to selecting a random subset. A summary of the methods and graph legend is presented in Table 1. All methods ran 100 times and results were averaged. To allow some degree of randomness, instead of taking the best test in each iteration, a random test was selected among the top 3 tests.

Label	**Method**	**Color**
Add farthest (avg)	Add the least similar test to the current set. Similarity to a group is the average similarity to its members.	🟥
Add farthest (min)	Add the least similar test to the current set. Similarity to a group is the minimal similarity to its members.	🟩
Remove farthest (avg)	Remove the least similar test from the current set. Similarity to a group is the average similarity to its members.	🟪
Remove farthest	Remove the least similar test from the current set. Similarity to a group is the minimal similarity to its members.	🟦
Add min event	Add a test covering the least covered event.	🟧
One-class SVM	Build one class SVM models iteratively and add 50 outliers to the set in each iteration.	🟦
Random	Randomly choose n tests.	🟪

Table 1: Greedy test ranking methods.

In Figure 10 and Figure 11 we report the coverage percentage of single events and event pairs for each method. As expected, the greedy algorithm that adds a test covering the minimally covered event, is the first to reach full coverage. Both incremental and decremental similarity based methods show the same trend. Measuring dissimilarity to an existing set gives better results when taking the minimal dissimilarity than when taking the average. Most methods perform better than random, at least for large n.

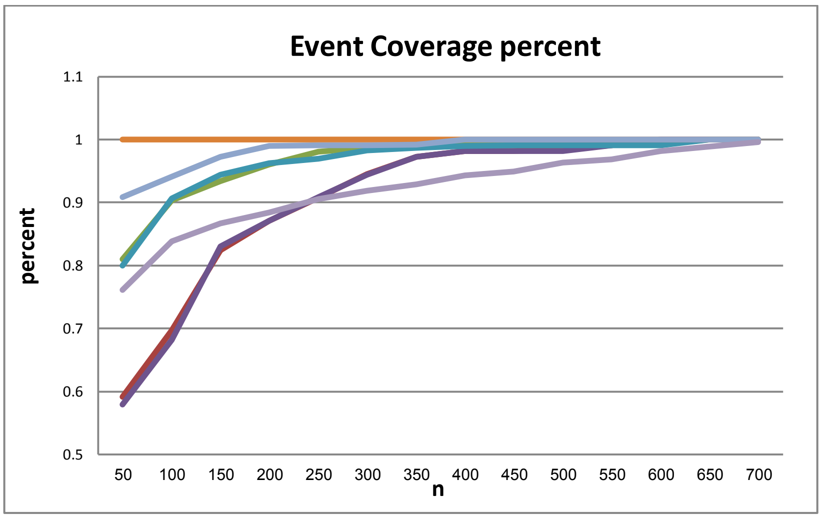


Figure 10: Event coverage percent as a function of the number of tests (n). Legend is presented in Table 1.

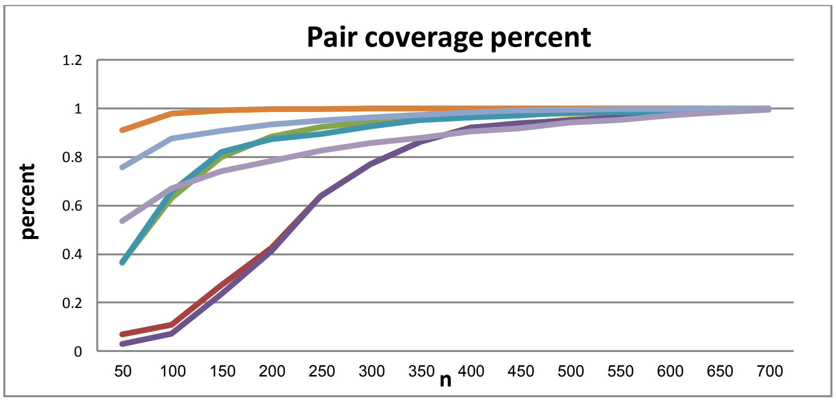


Figure 11: Event pair coverage percent as a function of the number of tests (n). Legend is presented in Table 1.

A more general objective is to cover all possible subsets of events. As showed in section 4.3 we only need to look for tests that are not contained in any other test. A minimal set of tests that covers all subsets contains 291 tests. Figure 12 shows the number of tests not dominated (contained) by any other test selected as a function of n. Best performance is obtained by the method that adds the test with minimally covered event, reaching full coverage at 600 tests. Other methods do not perform consistently better than random selection.

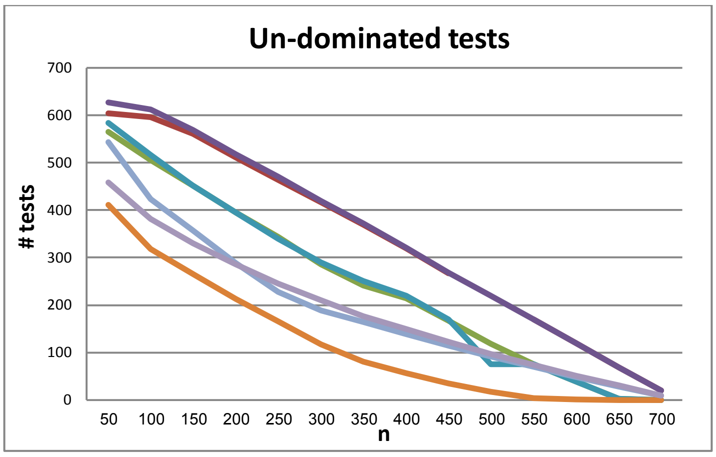


Figure 12: The number of tests not dominated by any of the tests selected. Legend is presented in Table 1.

Another possible objective is to increase the number of times each event is covered. Figure 13 and Figure 14 show the average and minimum number of times events are covered for each method. Note that the 'min event' objective considers only events that have not reached their maximal possible number of covering tests. Not surprisingly, the 'add min event' method is best in terms of the average and minimal event cover. Other methods are worse than random when it comes to the average event cover. On the other hand, for minimal event cover, all methods are better than random.

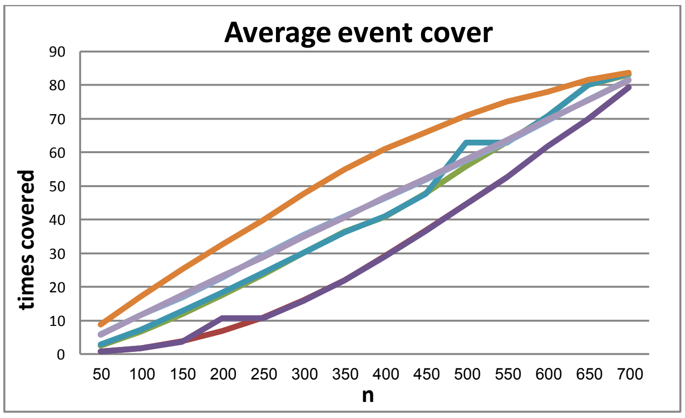


Figure 13: Average number of times events are covered. Legend is presented in Table 1.

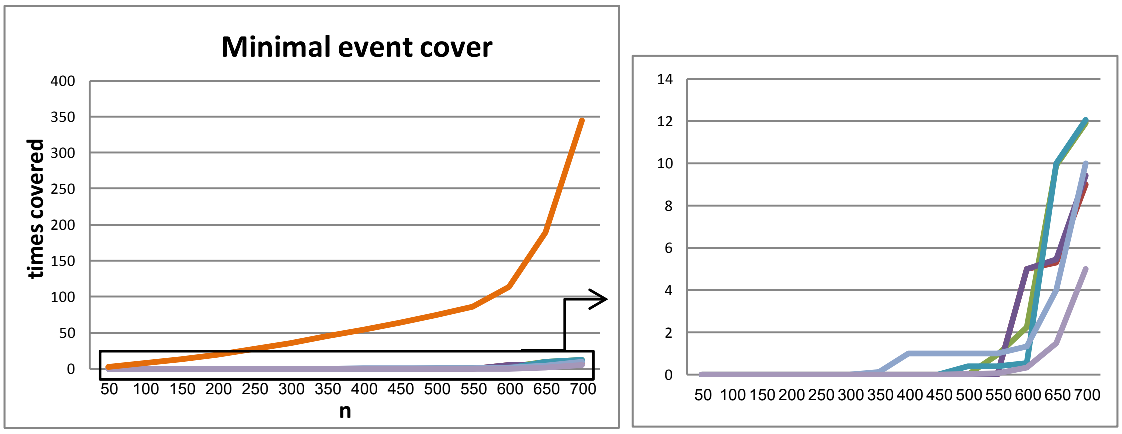


Figure 14: Minimum number of times events are covered. The left plot presents all methods while the right one shows all methods except 'min event'. Legend is presented in Table 1.

We would like the selected subset to be heterogeneous. Figure 15 shows the homogeneity (as defined in Section 2.2.3.4) of the solution produced by each method as a function of n. The methods that are based on average dissimilarity, 'add farthest' and 'remove farthest', perform best.

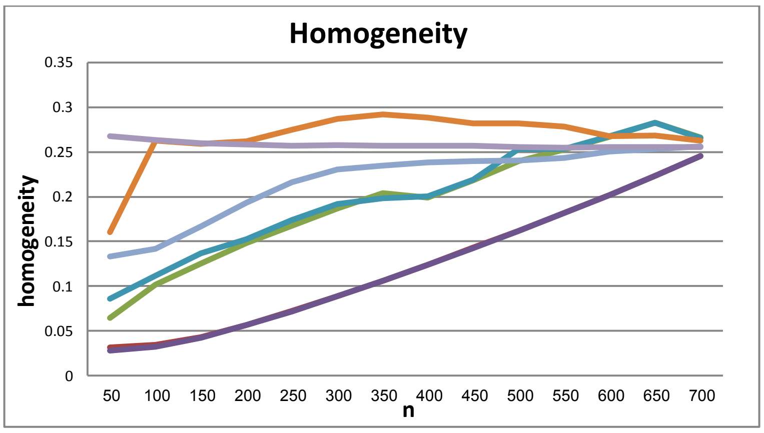


Figure 15: Subset homogeneity of each method as a function of n. Legend is presented in Table 1.

In a different experiment, we repeated the tests of Figure 15 starting with the basic subset of 291 tests that covers all possible subsets. Figure 16 shows the homogeneity of each method as a function of n in this case. Most methods provide similar homogeneity to random selection of tests. Only the two methods based on average similarity have consistently lower homogeneity.

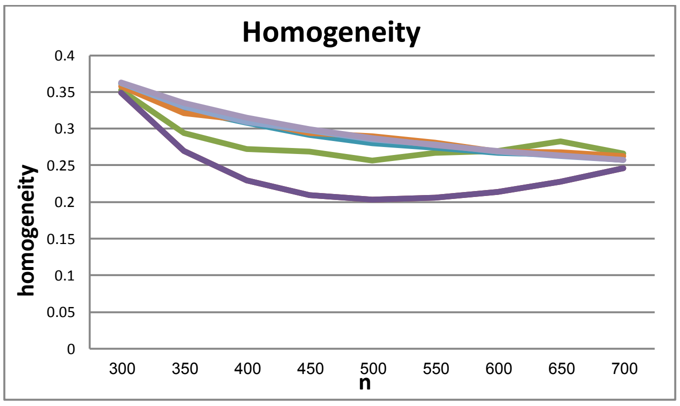


Figure 16: Subset homogeneity of each method as a function of n. The subset selection starts from the known set of 291 tests that dominates all tests. Legend is presented in Table 1.

Figure 17 and Figure 18 present the average and minimum times events are covered when the selection start from the dominating set. The 'add min event' method gives the best results for both average and minimum event cover. Other methods have worse average event cover than random. On the other hand, with the minimum event cover objective, all methods are better than random.

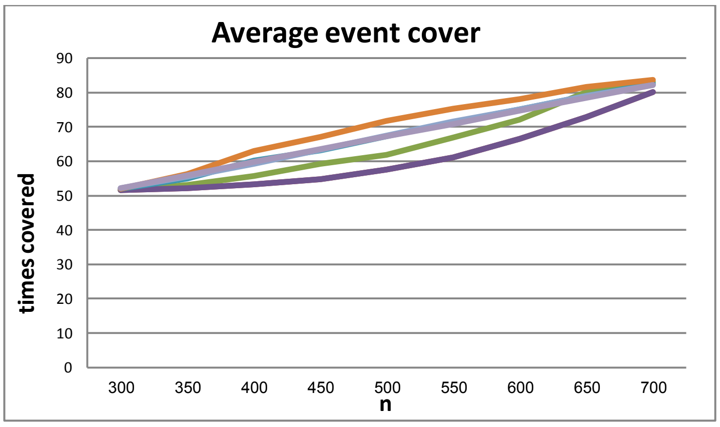


Figure 17: Average number of times events are covered. The subset selection starts from the known set of 291 tests that dominates all tests. Legend is presented in Table 1.

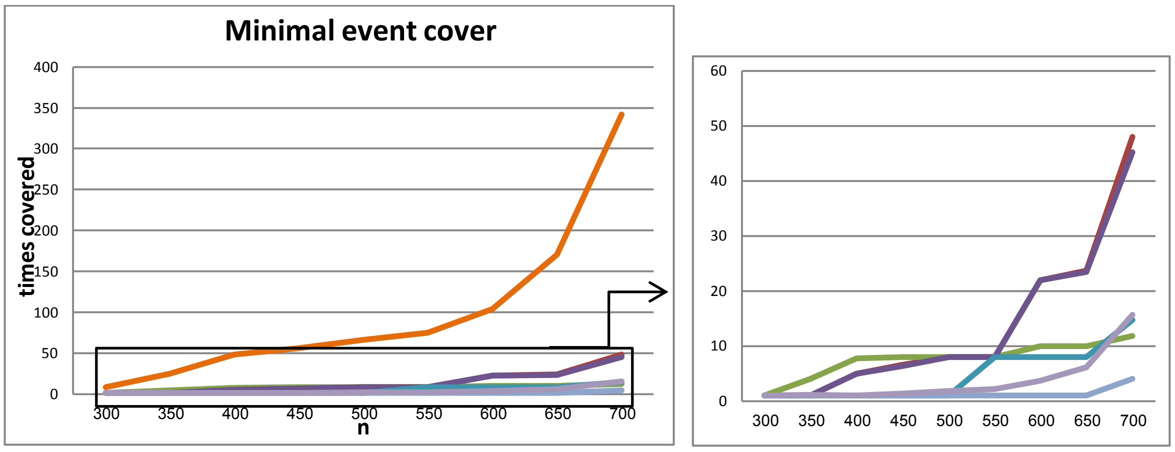


Figure 18: Minimum number of times events are covered. The subset selection starts from the known set of 291 tests that dominate all tests. The left plot presents all methods while the right one shows all methods except 'min event'. Legend is presented in Table 1.

In conclusion, we see an advantage of the 'add min event' approach for most objective functions except those aiming to maximize heterogeneity. Interestingly, the SVM method does not perform well. It is noteworthy that randomly selecting tests sometimes outperforms more directed algorithms.

6. Analysis using clustering techniques

We now describe a different approach to test optimization. We will try to partition the tests into clusters that achieve similar event coverage. Such a partition of the tests can help discover redundant or similar tests and replace them with a smaller number of representative tests. Together with a visualization tool it may also assist in finding low cover areas.

Clustering expression profile of genes is a common method in bioinformatics. There are several gene expression analysis and visualization tools, such as EXPANDER [40]. Our idea is to use the test-events hit matrix as if it was a gene expression data and apply these tools.

6.1. Ad-hoc similarity measures for post-silicon test data

6.1.1. Hit matrix similarity

The hit count matrix can formally be treated as a gene expression matrix and then one can use some clustering methods such as K-means, Click or SOM. There are several problems in this approach:

- The hit counts depend on the time each test was run. In order to achieve a reliable comparison between the hit counts of different tests we should have used the rate of events and not the hit count. The very high variance in the hit count between the tests and between the events may also suggest that the actual numbers are misleading.
- Clustering is dependent on the data normalization methods. In our study, several normalization methods were tried: normalizing each test or event to mean 0 and variance 1, quantizing hit numbers or even binarizing them. Each of these methods gives a different ranking of the similarities between tests or events.
- Another problem is the sparseness of the matrix. Similarity measures like Pearson correlation (used in Click) or Euclidian distance (used in K-means) will consider sparse test vectors to be very similar. Using Euclidian distance for example, a test that hit an event a thousand times would be considered more similar to a test that does not hit the event than to one that hits it a million times. A better similarity measure should give high score to tests that hit common events.

6.1.2. Binary test vector similarity

Validation engineers are mostly concerned with whether the event was covered than with the actual count, assuming that it reached the threshold. Together with the problems described above this leads us to concentrate on a binary coverage matrix. The matrix holds 1 for each event that is covered by a test. Alternatively, one could think of each test as the set of events that it covers.

We would like to use a similarity measure that would prefer matches in hits over misses (or 1's over 0's). One such similarity measure is the Jaccard coefficient, defined as the size of the intersection of two sets divided by the size of their union. Let v_i be the set of events covered by test i.

$$S_{Jaccard}(v_i, v_j) = \frac{|v_i \cap v_j|}{|v_i \cup v_j|}$$

This measure scores high two tests that share many common events. A drawback of the Jaccard coefficient is its tendency to underestimate the similarity as depicted in Table 2. Take for example the following scenarios:

- When the sets are about the same size and overlap on 1/2 of the elements, Jaccard will give a similarity of only 1/3. To get a similarity of 1/2, the sets need to overlap on 2/3 of the elements.
- When one set is contained in the other, Jaccard similarity will be equal to the proportion of containment. For example, if the proportion is 1/2, the similarity will be only 1/2.

Possible solutions [9] could be replacing the denominator by $\min(|v_i|, |v_j|)$ or by $\max(|v_i|, |v_j|)$ for example. Another way is computing the geometric mean or the arithmetic mean of the fractions $\frac{|v_i \cap v_j|}{|v_i|}$ and $\frac{|v_i \cap v_j|}{|v_i|}$. When looking on the sets as binary vectors, the geometric mean can be interpreted as the normalized dot product.

$$S_{geometric}(v_i, v_j) = \frac{|v_i \cap v_j|}{\sqrt{|v_i| \cdot |v_j|}} = \frac{\langle v_i, v_j \rangle}{\|v_i\| \cdot \|v_j\|}$$

Another possible measure is:

$$S_{arithmetic}(v_i, v_j) = \frac{\frac{|v_i \cap v_j|}{|v_i|} + \frac{|v_i \cap v_j|}{|v_j|}}{2} = \frac{|v_i \cap v_j|}{2} \cdot \left(\frac{1}{|v_i|} + \frac{1}{|v_j|}\right)$$

Scenario	Jaccard	Geometric	Arithmetic
$\lvert V_1\rvert = \lvert V_2\rvert = k$ $\lvert V_1 \cap V_2\rvert = \alpha k$	$\frac{\alpha}{2-\alpha}$	α	α
$V_1 \subseteq V_2$ $\lvert V_1 \cap V_2\rvert = \lvert V_1\rvert = \alpha\lvert V_2\rvert$	α	$\sqrt{\alpha}$	$\frac{1+\alpha}{2}$

Table 2: Comparison of three similarity measures for sets (or binary vectors). The table exemplifies Jaccard's tendency to underestimate similarity.

For our data set, the similarity between every pair of tests obey: $S_{Jaccard} \leq S_{geometric} \leq S_{arithmetic}$. The second inequality holds for every data set because of the arithmetic-geometric means inequality.

How can we estimate how similar are the two similarity measures? We used the Mantel [16], [17] test described in Section 2.2.4.3 to evaluate the correlation of two similarity matrices.

When scoring similarity of all pairs of tests in our data, the correlation between the Jaccard similarity and the geometric similarity is 0.9873 using Spearman correlation with p-value$< 10^{-4}$. This means that these measures are quite similar.

6.1.3. Between cluster similarity

Given some partition of the tests into clusters, how does one calculate the similarity between clusters or between a test and a cluster? There are several common alternatives:

- Calculate the binary cluster centroid. This is a set/vector that holds events that appear in the majority of cluster members. Similarity of two clusters is then defined as their centroids' similarity. This method is used in the binary matrix decomposition algorithm [41], a version of a binary K-means.
- Calculate the average similarity between all pairs of tests, one from each cluster.

6.2. Clustering with K-means

K-means was used to cluster the sample data because of its simplicity and ability to adapt to different similarity measures. The K-means code was inspired by Expander's implementation.

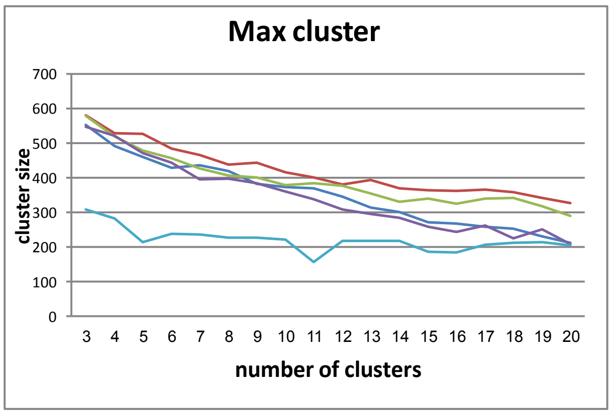
The choice of the initial partition in K-means affects the clustering solution. The choice of the representatives was done in two ways:

- Choose disjoint tests as representatives. We will refer to this method as the 'non-overlapping seeds' method.
- Choose tests that have some overlap as representatives. Randomize 100 pairs of tests and take the 10th highest dissimilarity as a threshold. Then choose tests that are at least as dissimilar as the threshold. We will refer to this method as the 'overlapping seeds' method.

Using a binary cluster centroid for the cluster-to-single-test similarity did not work well in K-means. Due to the sparsity of data, the centroids were very sparse and non-informative. We also tried defining the set of events of a cluster by the set of all events that are hit by a fraction p of the cluster tests, for different values of $p < \frac{1}{2}$, with poor results. Therefore, to compute similarity between clusters, all pair-wise similarity values for tests in the two clusters were computed and the average was used as the inter-cluster similarity.

K-means was run for $k = 3$ to 20 and results were averaged over 25 runs for each k. Jaccard and geometric mean similarity measures were used. In addition, the effect of the initial solution was tested, by choosing initial seed tests with and without common events.

Figure 19 shows the distribution of the cluster sizes as a function of the number of clusters. The graphs plot the maximal size cluster and the cluster sizes standard deviation. We noticed that in all clustering solutions there is always one big cluster consisting of at least 50% of the tests. It can be seen that the choice of the initial solution has a greater effect on the distribution than the similarity measure. When the initial solution for K-means has tests with overlapping events, the final clustering tends towards smaller clusters in both similarity measures. It also shows results when using Euclidian distance based similarity, using K-means in Expander. K-means with Euclidian distance (see Section 2.2.3.1) tends to divide clusters more uniformly than binary measures.



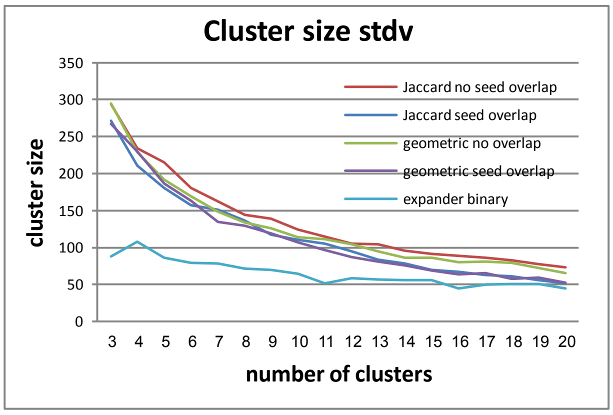
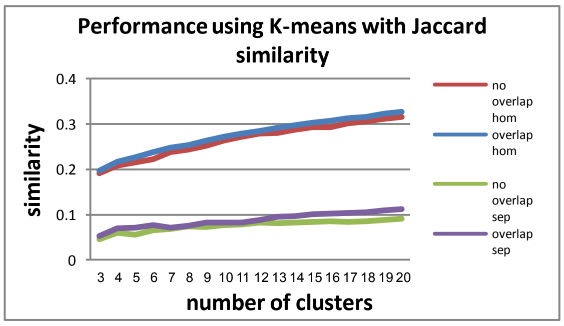


Figure 19: Performance of different variants of K-means as a function of the value k. Left: maximum cluster size. Right: cluster size standard deviation. K-means was run with two similarities measure: Jaccard and geometric mean. Different choices of the initial solution were tested: tests that have coverage overlap and those that do not. For comparison, Expander's K-means solution with Euclidian distance is presented too.

Figure 20 shows the average homogeneity and separation for all clusters with two similarity measures. Notice that different similarity values are a function of the similarity measure. Separation is presented here using similarity and not dissimilarity so lower values means better separation. When splitting the data to more clusters we get more homogeneous clusters, as expected. On the other hand, when there are more clusters the separation between them decreases (similarity increases). There is a tradeoff between homogeneity and separation when choosing the initial solution – taking orthogonal cluster representatives gives better separation but worse homogeneity. The difference is noticeable in the Jaccard similarity.



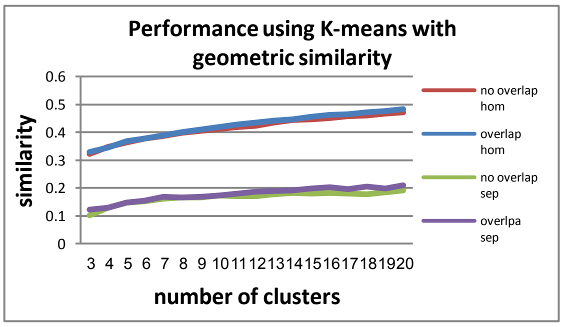


Figure 20: Performance of different variants of K-means as a function of the value k. The graphs show different similarity measures and different choices of the initial solution. Left: homogeneity and separation using Jaccard similarity. Right: homogeneity and separation using geometric mean similarity

Figure 21 shows the average silhouette. The silhouette index takes into consideration both homogeneity and separation. Finding a maximum point for the silhouette can help choose the "correct" number of clusters. Unfortunately, there is no clear maximum indicating a "correct" number of clusters. We can see that for large number of clusters, starting with orthogonal representatives has a clear advantage.

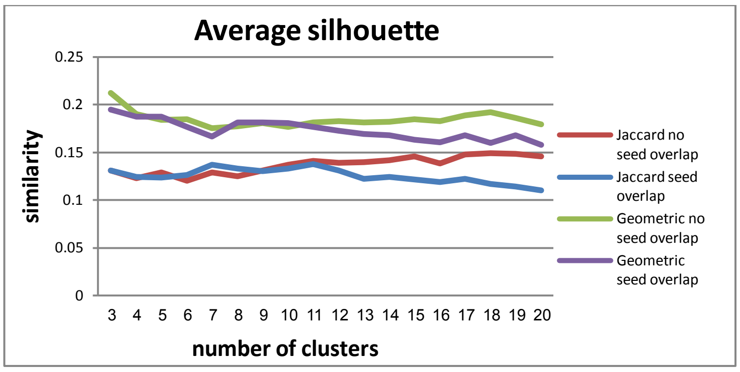


Figure 21: The average silhouette of solutions obtained using different variants of K-means, as a function of the value k. Variants differ by the similarity measure and the choice of the initial solution.

6.3. Hierarchical clustering

6.3.1. Dendrogram solution

We present application of hierarchical clustering to the sample post-silicon data. Expander supports hierarchical clustering on both matrix dimensions but it uses Pearson correlation similarity measure. In order to use the binary similarity measures the R statistical software was used.

Figure 22 shows hierarchical clustering using Jaccard similarity. The tree's leaves are the test names. The tree is cut into 8 clusters at height 5.

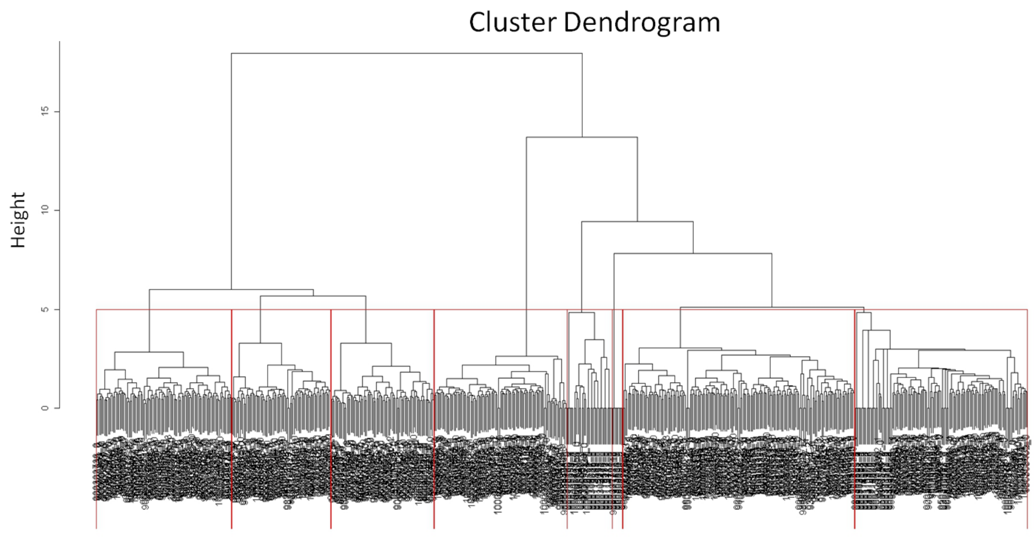


Figure 22: A hierarchical clustering solution presented as a dendrogram. The Jaccard similarity measure was used here. Each leaf corresponds to a test. Red lines represent trimming the tree to 8 clusters.

Such a dendrogram can also be used to produce clusters by cutting the tree at a given height or branch length. Another approach is to focus on deep branches that may be singled out as highly similar tests.

Comparing clusters produced from hierarchical clustering vs. clusters produced by K-means with the same number of clusters, shows the following results: the homogeneity is better in the hierarchical clustering solutions but the separation is worse, for all cluster sizes. The differences are extreme when Jaccard similarity is used. These results are not surprising since each new cluster added by cutting the hierarchical clustering tree is a sibling of an existing cluster. Therefore the clusters could be highly similar causing the separation to deteriorate.

The hierarchical method has an advantage of explicitly presenting the similarity relationship between clusters. It may also suggest the number of clusters to use when running K-means, for example.

6.3.2. Neighbor joining

Figure 23 shows an example of a neighbor joining tree of the tests built according to their geometric mean similarities. Four sets color the tree, each one holds about a quarter of the tests. The tests were divided according to the number of events they cover. It can be seen that tests that hit a similar number of events also hit a similar subset of events, since they appear near each other in the tree. This property of the test data should be analyzed by the validation team in order to design a more diverse test suite.

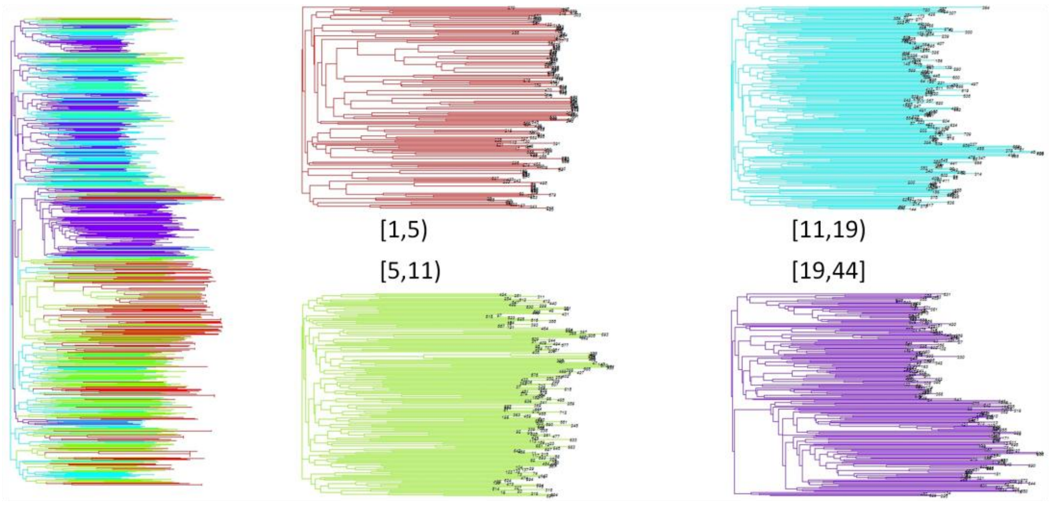


Figure 23: A neighbor joining tree solution. The tree was partitioned into four sets (colors) based on the number of events each test covers. The number of events each test in a group covers is marked in brackets.

6.4. Clustering using Click

Here we try to use the Click algorithm for analyzing our post-silicon data. Recall that Click can produce a solution that contains unclustered singletons. In the context of test clustering, such outliers should probably be included in any test cover. They are not similar to any of the clusters, so they should be selected for the test suit as they reflect unique behavior.

By default, Click uses Pearson correlation as a similarity measure. It also enables using the dot product similarity measure, which is the same as the geometrical mean for binary vectors.

The Click algorithm receives a homogeneity threshold value θ. Click was run with several values of θ. Value of 0 refers to the algorithm's default value.

Figure 24 displays solution properties when using Click. As in K-means, the solutions always contain one big cluster. As the threshold for homogeneity increases, the algorithm tends to divide the data and keep more tests un-clustered (singletons). High homogeneity causes smaller clusters on the one hand, but on the other hand more tests remain un-clustered. When setting the threshold too high, Click does not produce any clusters.

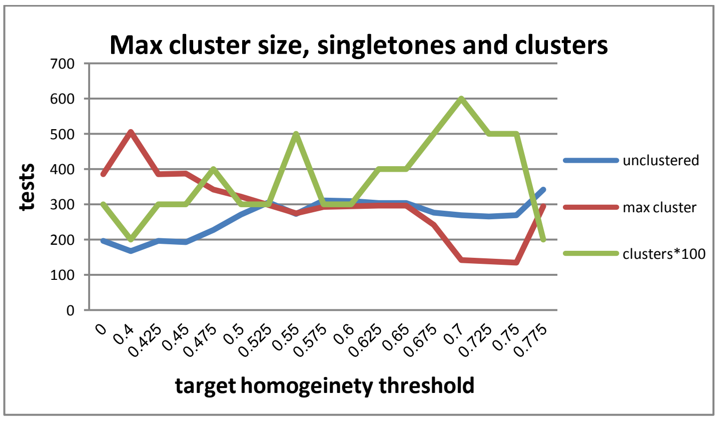


Figure 24: Characteristics of Click solution for different homogeneity thresholds. The graph plots the number of unclustered tests and the size of the maximal cluster for each homogeneity threshold (θ). It also shows the number of clusters (multiplied by hundred to fit the scale).

Figure 25 shows the homogeneity and separation for each threshold value. There is no clear value that achieves both high homogeneity and low separation. In comparison with K-means with the same similarity measure (geometric mean/dot product), Click has better homogeneity values but slightly worse separation. Click has slightly better silhouette values than its K-means counterparts. Click has an advantage over K-means by not clustering all the data.

In addition to the average homogeneity, individual cluster homogeneity is also important. Highly homogenous clusters indicate similar tests groups that can be reduced. Usually the most homogenous cluster is small and the average homogeneity is closer to the minimum homogeneity which is obtained by a large cluster. Individual cluster homogeneity data are presented in Figure 25.

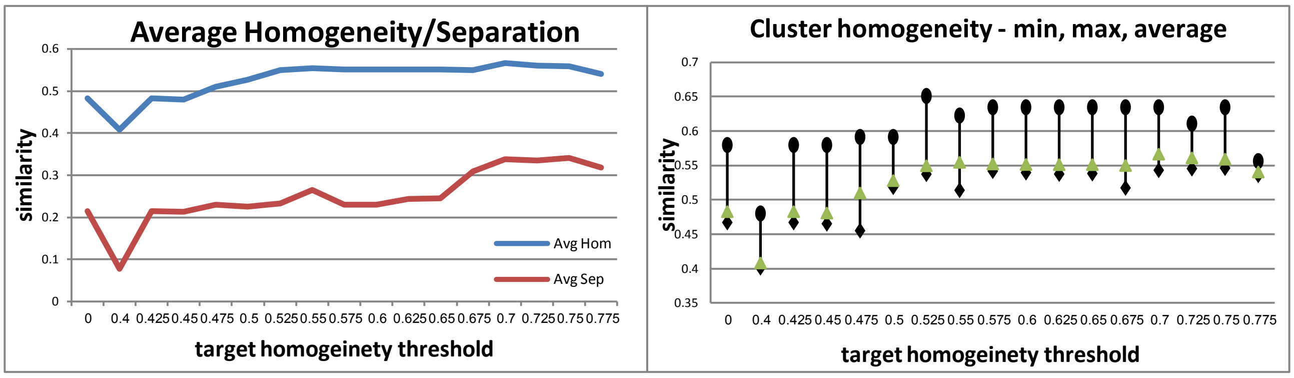


Figure 25: Characteristics of clustering solutions produced by Click. Left: the homogeneity and separation of Click's solution as a function of the threshold (θ). Right: the minimal, maximal and average cluster homogeneity in every solution.

Figure 26 shows an example of the highest individual homogenous cluster found for a threshold of $\theta = 0.525$. The cluster homogeneity is 0.65 and it holds 32 tests. The figure exemplifies Expander's visual abilities displaying the data matrix.

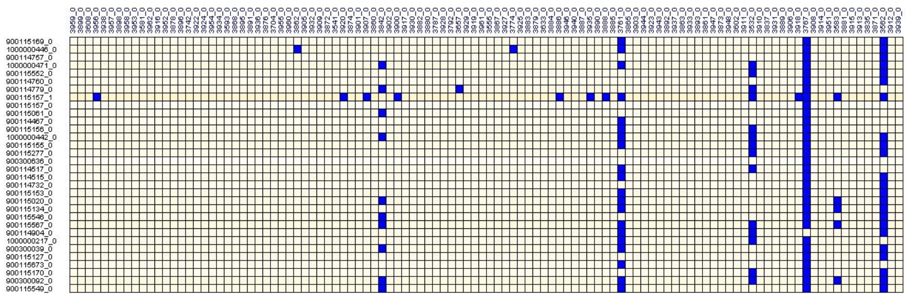


Figure 26: A homogenous cluster produced by Click as visualized using Expander. The vertical and horizontal axes correspond to the tests and events respectively. Blue dots indicate an event covered by a test.

6.5. Consensus clustering and Model explorer

When we do not know the real underlying cluster structure of our data, each clustering algorithm may provide us with a different solution. Algorithms with a stochastic component may provide different solutions on different runs. How can we trust our solution in this case? Inspired by cross validation techniques, *consensus clustering* method [42] captures majority vote across multiple runs of a clustering algorithm. The method determines the number of clusters in the data and assesses the stability of the discovered clusters. We say that a clustering solution is *stable* if different runs of an algorithm (with different seeds, for example) provide similar solutions. Intuitively, if there is a real cluster structure hidden in the data, every run should provide similar results.

While consensus methods were originally used to assess the number of clusters in a good clustering solution, they can also be used for choosing other parameters of clustering algorithms.

The consensus method works in the following way:

1) Initialize a consensus matrix with rows and columns corresponding to elements.
2) Sample a subset (say, 80%) of the data.
3) Run the clustering algorithm on the subset.
4) Increment results of cells in the consensus matrix if the corresponding two elements are in the same cluster.

5) Repeat steps 2-4 several times.
6) Calculate similarity between elements as the percentage of solutions clustering them together.
7) Repeat 1-7 for different parameters of algorithm (number of clusters, for example).
8) Choose the best parameters and use the corresponding similarity matrix.

The choice of best parameters is done using two methods which we now outline. These methods are fully described in [42], [43].

If we were to plot a histogram of the consensus matrix entries, perfect consensus would translate into two bins centered at 0 and 1. As the number of clusters increases the distribution of the similarity values would further dissolve. For a given histogram, we can define and plot the corresponding empirical cumulative distribution function (CDF). The CDF is defined as the probability of a random variable to get values up to a certain value, i.e., $CDF(x) = \Pr(X \leq x)$. We can then compute the area under the CDF. Monti et al. [42] recommend using the parameter set for which the area under the CDF stops changing, i.e., when the difference between two parameter sets is reaching 0.

The *model explorer* method [43] is similar to consensus clustering in using multiple runs of the clustering algorithm to decide the correct number of clusters. In this method two algorithms are run in each iteration and the similarity between the two solutions is recorded. The distribution of the similarity can then be used to choose a stable clustering solution. To assess the similarity between two solutions the *adjusted rand* score [42] was used. Given two clustering solutions, $\mathcal{S}$ and $\mathcal{C}$, we define a contingency table $N \in \mathbb{N}^{|\mathcal{S}|\times|\mathcal{C}|}$ by $N_{ij} = |S_i \cap C_j|$. Denote row sum as $N_{i\cdot} = \sum_j N_{ij}$, column sum as $N_{\cdot j} = \sum_i N_{ij}$ and number of elements $N = \sum_i \sum_j N_{ij}$. The adjusted rand score is defined as:

$$r = \frac{\sum_{ij}\binom{N_{ij}}{2} - \left[\sum_i \binom{N_{i\cdot}}{2}\sum_j \binom{N_{\cdot j}}{2}\right]\Big/\binom{N}{2}}{\frac{1}{2}\left[\sum_i \binom{N_{i\cdot}}{2} + \sum_j \binom{N_{\cdot j}}{2}\right] - \left[\sum_i \binom{N_{i\cdot}}{2}\sum_j \binom{N_{\cdot j}}{2}\right]\Big/\binom{N}{2}}$$

6.5.1. Application to Click

While the original motivation of the consensus method was to find the correct number of clusters for a given algorithm (such as K-means), the Click algorithm determines the number by itself. However, Click accepts a threshold parameter θ for determining the cluster homogeneity. Consensus methods were used here to determine the most suitable threshold. Another difference in Click is allowing elements to remain un-clustered (singletons). It does not limit the use of consensus methods since if an element is truly a singleton then it is expected to be so in every clustering solution.

Figure 27 shows the adjusted rand index of Click's clustering solution as a function of θ. The inner small plot is a zoom-in on the section between 0.8 and 1. It can be seen that the most stable solutions are obtained for $\theta = 0.7$. The median similarity score is the highest and the distribution is narrow for $\theta = 0.7$. For $\theta \geq 0.7$ the scores completely deteriorate.

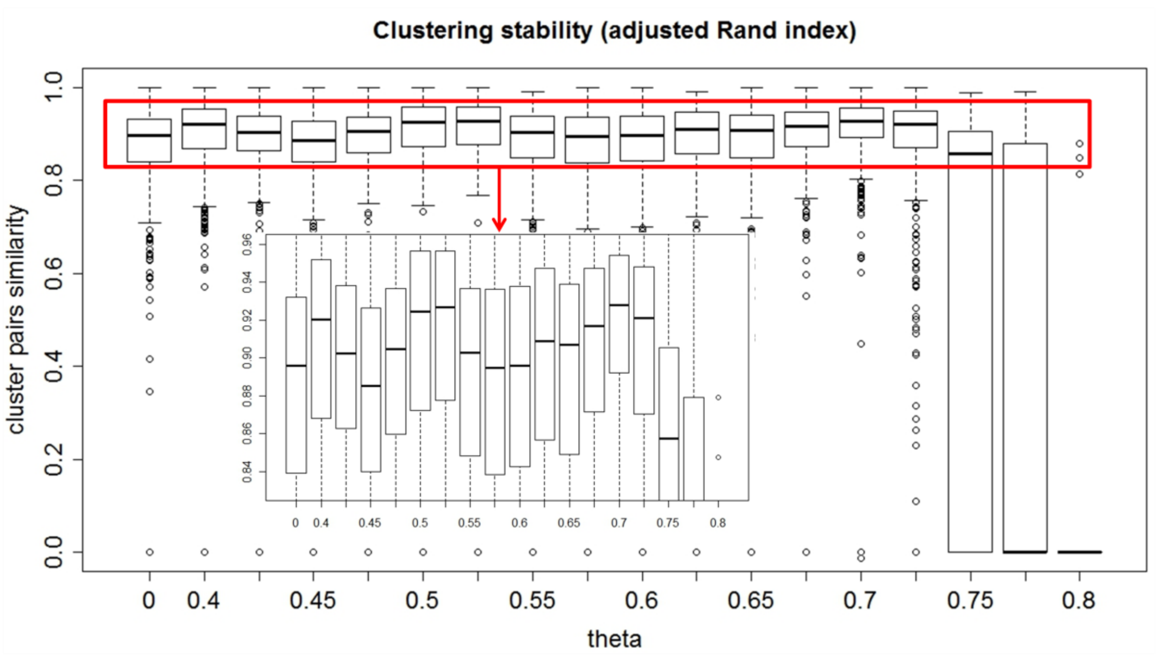


Figure 27: Adjusted rand index of Click clustering solutions. The index is shown for different values of theta. The inner plot is a zoom-in on the section between 0.8 and 1.

Figure 28 shows the area under the consensus matrix cumulative distribution function (CDF) as a function of theta. We can see that $\theta = 0.7$ gives the maximal area.

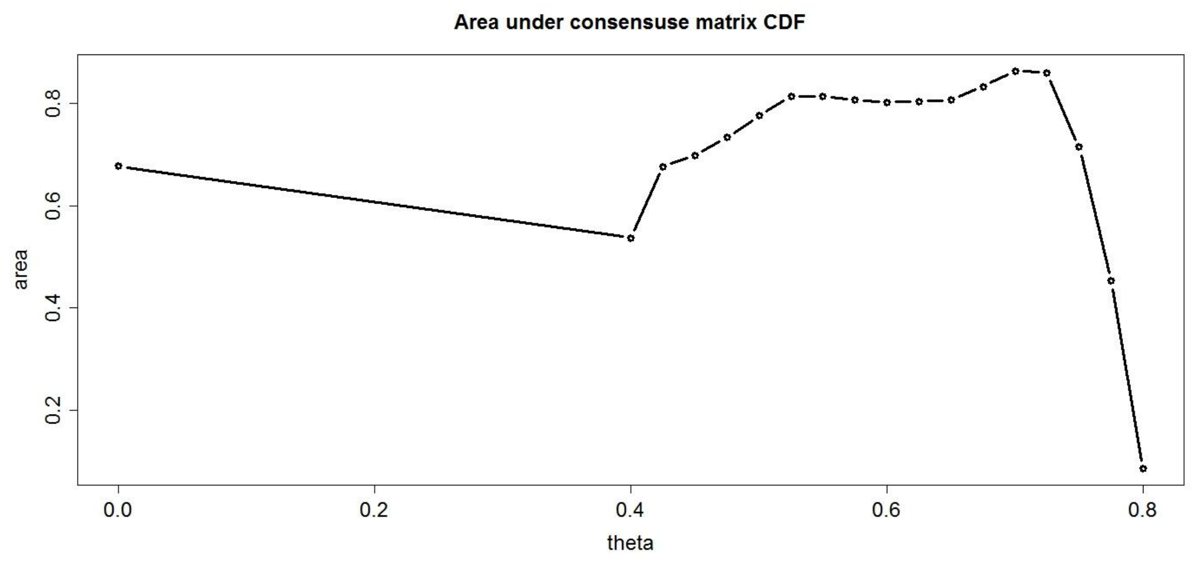


Figure 28: Area under the consensus matrix CDF for each threshold. The maximal value is for 0.7.

In Figure 29 we show the number of singletons as a function of θ. We can see that when we start increasing θ the number of singletons increases as well. Higher levels of θ make Click groups more homogenous, thus leaving more elements unclustered. For $\theta = 0.7$ there is a local minimum with small variance in comparison to adjacent values of θ.

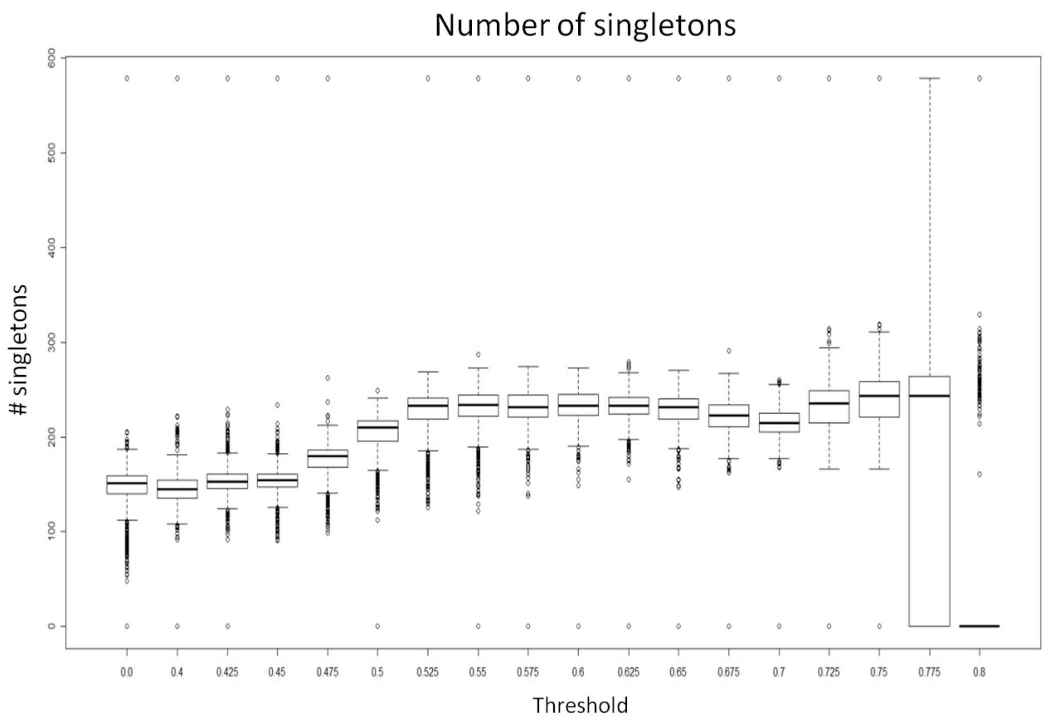


Figure 29: Number of singletons as a function of the threshold.

The consensus clustering matrix can also be used as a similarity matrix between elements. Figure 30 shows a heatmap of the original similarity matrix given as input to Click. Figure 31 shows the consensus clustering matrix obtained by running Click with $\theta = 0.7$. We can see that the consensus matrix is smoother than the original similarity matrix. Four main clusters appear in the consensus map and about 200 tests remain dissimilar to all other tests. These tests correspond to singletons in most of Click's runs.

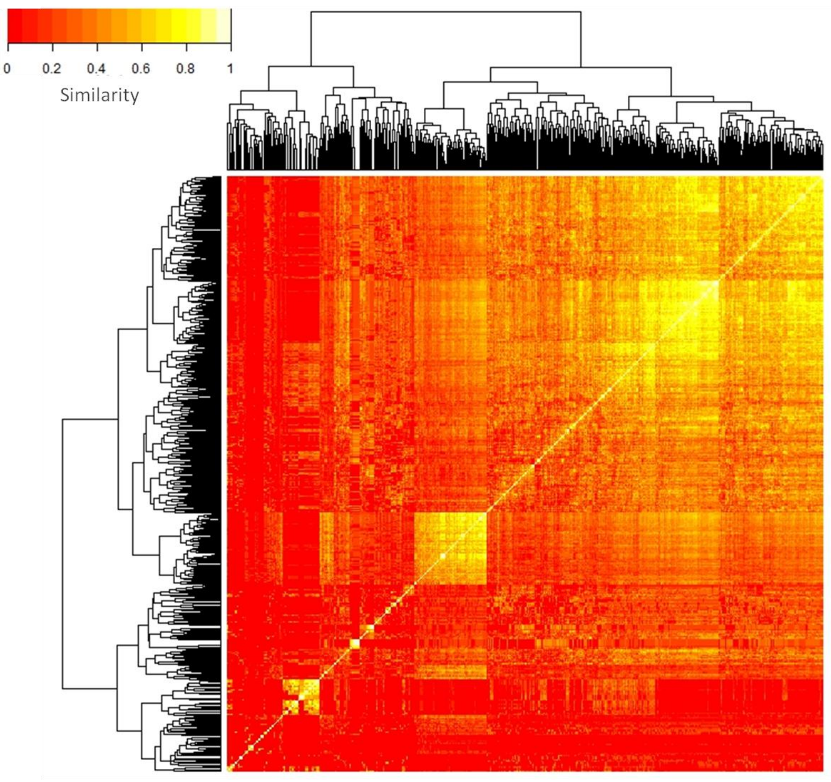


Figure 30: The original similarity matrix.

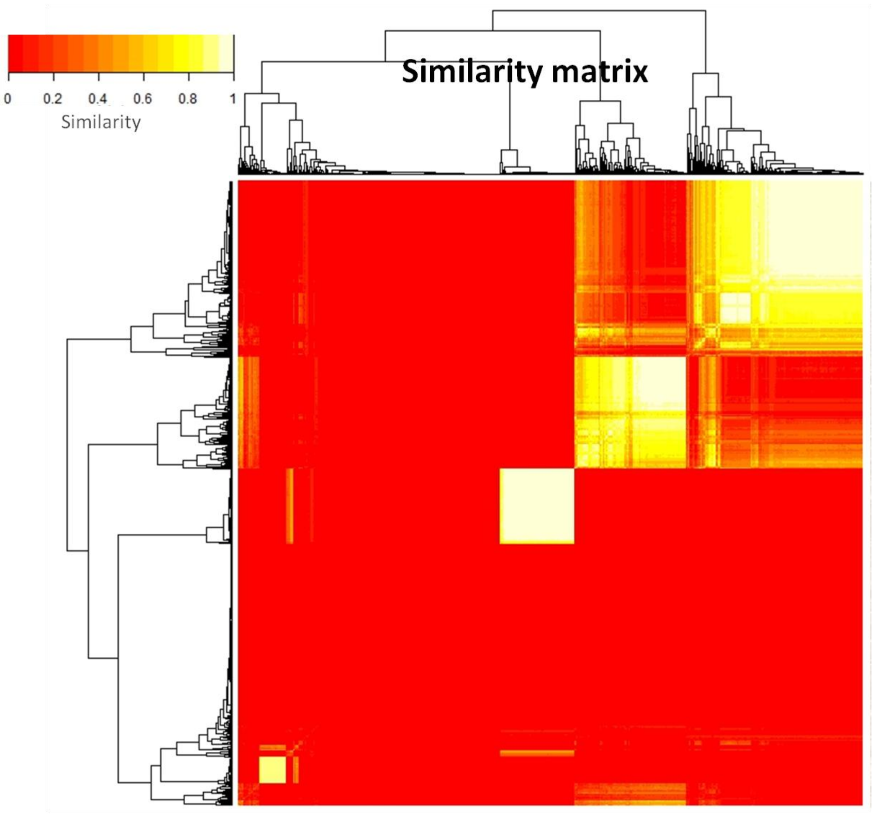


Figure 31: Consensus similarity matrix for $\theta = 0.7$.

In the criteria we examined, running Click with $\theta = 0.7$ gave the most stable results. We showed that consensus and stability methods can be used to determine clustering parameters and not just the number of clusters. In addition, these methods also work on clustering algorithms that do not produce a full partition of the data, i.e., allowing singletons.

6.5.2. Application to K-means and hierarchical clustering

Figure 32 and Figure 33 below present the use of consensus clustering and model explorer methods to determine the number of clusters. We used K-means and hierarchical clustering as they get as input the number of clusters. Because the consensus method does not depend on the clustering method nor on the similarity measure, it enables us to compare different parameters.

We ran all algorithms on the post-silicon test data. Hierarchical clustering was run with two merge criteria: Ward and average linkage. K-means was run with Jaccard or dot-product similarity measure. The results are inconclusive as to which method or measure is better. Nor is there a number of clusters that all methods agree upon. We can see that there is high variability between the two objectives of hierarchical clustering, while both K-means versions are about the same.

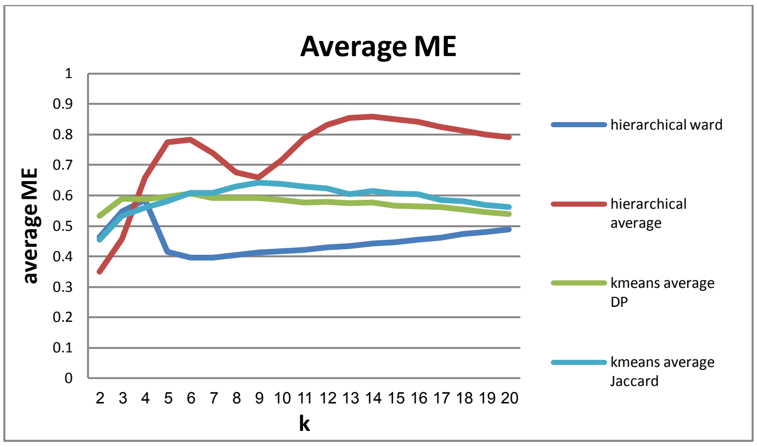


Figure 32: Rand index of K-means and hierarchical clustering solutions. The graph shows the average model explorer measure between two successive clustering solutions for each k.

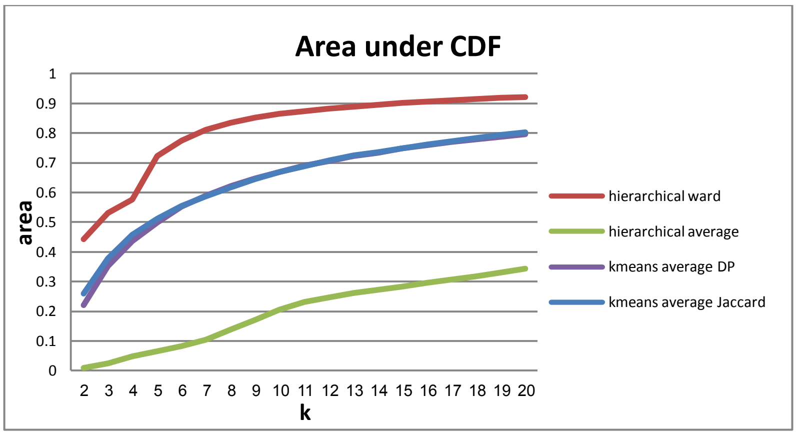


Figure 33: Consensus clustering quality of K-means and hierarchical clustering solutions. The graph shows the area under the consensus matrix's CDF for each k.

6.6. Analysis of clustering solutions using enrichment

Given a set of elements X, some of which have property π, and given a subset $S \subseteq X$, we say that S is *enriched* for property π if it contains significantly more elements with that property then expected by chance.

In computational biology, gene groups can be tested for their enrichment by biological properties. Similarly, in post-silicon testing, groups of tests can be enriched for some properties of the chip or system. This is motivated by the reasoning that tests that cover similar events tend also to have similar configuration of the chip and test's inputs. Test groups with significant overrepresented configurations can shed light on the structure of the tests. It can also help reduce the number of configurations and provide a base for construction of more efficient tests that may act as a class representative.

To evaluate the significance of a configuration appearing in a cluster we use the hypergeometric score. In our case, N is the total number of tests, n is the number of tests in the cluster under investigation, m is the total number of tests that use the property, and k is the number of tests in the cluster that use the property. The statistical significance (or P-value) of a property in a cluster is the probability to get a cluster with the same size that has at least the same number of tests with the property.

In order to account for multiple hypotheses of all clusters against all properties we use FDR correction [15] with threshold α. We say that a property is significant in a cluster of its corrected p-value is below a threshold α.

The following table presents an example of clustering the post-silicon data using Click with a homogeneity threshold of $\theta = 0.7$. Significance threshold was set to α=0.01. Inspired by the hypothesis that homogenous clusters will use similar and more specific configs, we measure the configuration homogeneity between tests. We also measure what fraction of the configs is used in each cluster, denoted as the config usage percentage. Clusters that use less configs can be viewed as more specific. Out of all significant configs we also look on those that appear in the majority (>50%) of the tests in the cluster, as they can be viewed as representative configs. By keeping the same cluster structure and shuffling the tests' labels we generate a random solution and evaluate its enrichment as well.

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Singletons
Size	141	136	54	51	38	33	269
# Significant configs	21	8	3	11	3	5	38
# Significant configs above 50%	15	8	3	1	2	4	17
Config homogeneity	0.4269	0.4301	0.4334	0.3743	0.3552	0.4728	0.3918
Config usage percentage	0.8914	0.9457	0.7674	0.8565	0.8643	0.7093	1
Random # significant	0	0	0	0	0	0	0
Random homogeneity	0.3929	0.3959	0.351	0.38	0.389	0.4036	0.3936
Random usage	0.9573	0.9302	0.9186	0.8798	0.841	0.8798	0.9728

Table 3: Analyzing configuration enrichment of a clustering result. Clustering was done using Click with a homogeneity threshold of $\theta = 0.7$.

The table shows there are indeed significantly enriched configs in some of the clusters in comparison to randomly permutated clusters. Most of the significant configs are also prevalent in their clusters. However homogeneity is not that conclusive. Clusters 2 and 5 have significantly low config usage in comparison to random while other clusters give inconclusive results. The singletons group has more significant number of configs than any cluster. This may be an artifact of the hypergeometric score, which is susceptible to large groups. When looking only on prevalent configs (i.e., configs occurring in at least 50% of the tests), the number of significant configs in the singletons group decreases from 38 to 17. In clusters 2 and 3 all significant configs appear in most of the tests.

The following table presents the significant configs with p-values above 1E-9. The config usage measures how many tests use that specific config. Different significant configs appear in different clusters, except for configs 168, 169, which appear in clusters 4 and 5.

Config ID	Usage in cluster	Usage in all tests	P-Value
Cluster 1			
80	81.56%	54.16%	4.72E-14
8	89.36%	66.07%	2.66E-12
113	62.41%	39.75%	1.13E-09
54	47.52%	26.59%	1.61E-09
56	47.52%	26.73%	2.13E-09
114	88.65%	69.67%	4.82E-09
Cluster 2			
76	77.94%	56.09%	4.07E-09
Cluster 4			
168	66.67%	13.85%	6.34E-20
169	47.06%	9.56%	2.08E-13
Cluster 5			
169	55.26%	9.56%	1.56E-13
168	60.53%	13.85%	4.70E-12
Cluster 6			
102	84.85%	32.41%	2.75E-10
Singletons			
115	79.55%	58.45%	8.22E-20
81	59.85%	39.89%	2.29E-17
52	81.78%	64.68%	2.51E-14
95	22.68%	11.36%	3.98E-13
189	45.35%	30.06%	6.67E-12
88	38.29%	24.38%	3.11E-11
77	62.08%	46.26%	3.68E-11
192	61.34%	46.12%	1.93E-10
194	40.89%	27.42%	5.27E-10
108	20.45%	10.94%	6.20E-10
78	63.20%	48.48%	7.39E-10
193	14.13%	6.65%	1.04E-09

Table 4: Significant configurations in clusters.

The same analysis on the system elements (SEs) of the tests gave 93, 69, 84, 152, 65, 149, 16 significant SE in each cluster, respectively. There are 57, 32, 20, 18, 2, 65, 14 significant system elements that appear in more than 50% of the tests in their cluster. In a random permutation of the labels no system element is significantly enriched in any cluster.

Such an analysis can assist the validation in comprehension and improvement of the tests suite. For example, in clusters 2 and 3 all significant configs appear in most of the tests. These configs can be thought of as representative for the cluster and allow the development of new tests from that group. Clusters 3 and 6 give significantly low config usage. This means they use highly specific configs in comparison to just random selection and even other cluster groups. Configs 168 and 169 appear in 10% and 14% of tests, but in cluster 4 and 5 they appear in the majority of the tests. These two configs clearly have a major impact on the test outcomes.

7. Algorithms for cohesive subgraphs

In this section we define cohesion in an undirected weighted graph. The objective function of cohesion is a generalization of subgraph density. Inspired by graph clustering, cohesion discourages the inclusion of inter-cluster edges and high degree nodes. We give a polynomial algorithm for finding a maximum cohesion subgraph, based on iterated flow computations.

7.1. Finding maximum cohesion subgraphs

In Section 2.2.5.1 we discussed the problem of finding a maximum density subgraph in an undirected (possibly weighted) graph. That problem can be solved in polynomial time using a series of min-cut computations. The density of a subgraph was defined as the ratio between the number of its edges and the number of its nodes.

We suggest modifying the definition of density to take into consideration the separation between the subgraph and the rest of the graph. Intuitively, a good module should be not only dense but also well-separated from the rest of the graph. Here we modify the original density definition, and provide it with a new name. As a first attempt we define the *cohesion* of a subgraph with node set S as

Definition 1
$$D(S) = \frac{W(S) - \beta W(S, V \setminus S)}{|S|}$$

$W(S)$ is the sum of edge weights in the subgraph induced by S, $W(S, V \backslash S)$ is the sum of the weights of the edges with one end in S, and $\beta \geq 0$. This new definition is motivated by graph clustering, where we seek to partition the graph into sets of nodes that are both highly connected internally and separated from each other.

This first definition of cohesion is biased toward large subgraphs. For example:

- Consider a graph composed of two cliques of size $\frac{n}{2}$ with $k > 0$ edges between them. The cohesion of one clique in the graph is $\frac{\binom{n/2}{2}-\beta k}{n/2} = \frac{n}{4} - \frac{1}{2} - \frac{2\beta k}{n}$, while the cohesion of the whole graph is $\frac{2\binom{n/2}{2}+k}{n} = \frac{n}{4} - \frac{1}{2} + \frac{k}{n}$. So, for any $k > 0$, a maximum cohesion subgraph will be the whole graph and not each of the cliques composing it, as we expect for small k.

- Consider a random graph built in the following manner; there are $\frac{n}{2}$ nodes connected in a clique and $\frac{n}{2}$ satellite nodes, each connected to every other node independently with probability $p < \frac{1}{2}$. The expected cohesion of the clique is $\frac{\binom{n/2}{2}-\beta\frac{n}{2}\frac{n}{2}p}{n/2} \approx \frac{n}{2}(\frac{1}{2} - \beta p)$, and the cohesion of the entire graph is $\frac{\binom{n/2}{2}+\binom{n/2}{2}p+\frac{n}{2}\frac{n}{2}p}{n} \approx \frac{n}{8}(1 + 3p)$. Now, the cohesion of the entire graph will be greater when $p > \frac{1}{3+4\beta}$. So when β increases we may encounter unintuitive behavior even for low values of p.

To overcome this bias we suggest redefining cohesion by adding another term that will penalize the use of low degree nodes. We subtract from the numerator the sum of complement degrees of nodes in S. Recall $\bar{d}_v \triangleq |\{(u,v)|u \in V, (u,v) \notin E\}|$. For $\gamma \geq 0$ we redefine *cohesion* as

Definition 2
$$D(S) = \frac{W(S) - \beta W(S, V \setminus S) - \gamma \sum_{v \in S} \overline{d}_v}{|S|}$$

Notice that $D(S)$ might be negative. This is the definition of cohesion we shall use from now on.

The maximum cohesion subgraph problem seeks $\hat{S} = argmax_{S \subseteq V}\, D(S)$. We will describe a polynomial algorithm for solving the problem on a graph with positive integer weights. Denote the maximum cohesion value by $D^* = D_{opt} = max_{S \subseteq V}\, D(S)$.

Goldberg [18] and Saha et al. [20] proposed two different algorithms for finding a maximum density subgraph based on searching flow networks. We will describe an algorithm based on Goldberg's method because it uses a simpler construction with $\Theta(|V|)$ nodes, whereas Saha's construction uses $\Theta(|V| + |E|)$ nodes. We note that Saha's construction could have been used for our purpose as well.

<u>The algorithm</u>

The algorithm reduces the problem of finding a maximum cohesion subgraph to a series of minimum cut problems, which are solvable using network flow techniques. The algorithm requires a logarithmic number of min-cut computations on networks with $\Theta(|V|)$ nodes and $\Theta(|E|)$ edges.

The algorithm works as follows: Let D^* be the maximum cohesion. At each stage of the algorithm there is a guess g for D^*. Then, a flow network is constructed and a minimum cut computation provides an answer whether $D^* \leq g$ or $g \leq D^*$, and a new guess is computed. When the search terminates $g = D^*$ and a maximum cohesion subgraph is found.

Given a guess g, we convert the graph $G = (V, E)$ with edge weights w_{ij}, into a network $N(g) = (V_N, E_N)$ as follows.

Let d_i be the weighted degree of node i of G as defined in Section 2.2.1 . Let $\widetilde{m} = (2\beta + 2\gamma + 1) * \max(W(V), n^2)$. Notice that $\widetilde{m}$ is a constant depending only on the graph and not on g. $\widetilde{m}$ is chosen so that all network capacities will be positive, as we shall see.

We add a source node s and a sink t to the set of nodes; replace each (undirected) edge (i, j) of G by two directed edges (i, j) and (j, i) of capacity $(1 + 2\beta)w_{ij}$ each; connect the source s to every node i of G by an edge of capacity $\widetilde{m} - 2\gamma\bar{d}_j$ and connect every node i of G to the sink t by an edge of capacity $\widetilde{m} + 2g - d_i$. Figure 34 illustrates the construction.

More formally,

$$V_N = V \cup \{s, t\}$$

$$E_N = \{(i, j), (j, i) | (i, j) \in E\} \cup \{(s, i) | i \in V\} \cup \{(i, t) | i \in V\}$$

$$C_{ij} = C_{ji} = (1 + 2\beta)w_{ij} \quad \forall (i, j) \in E$$

$$C_{sj} = \widetilde{m} - 2\gamma\bar{d}_j \quad \forall j \in V$$

$$C_{it} = \widetilde{m} + 2g - d_i \quad \forall i \in V$$

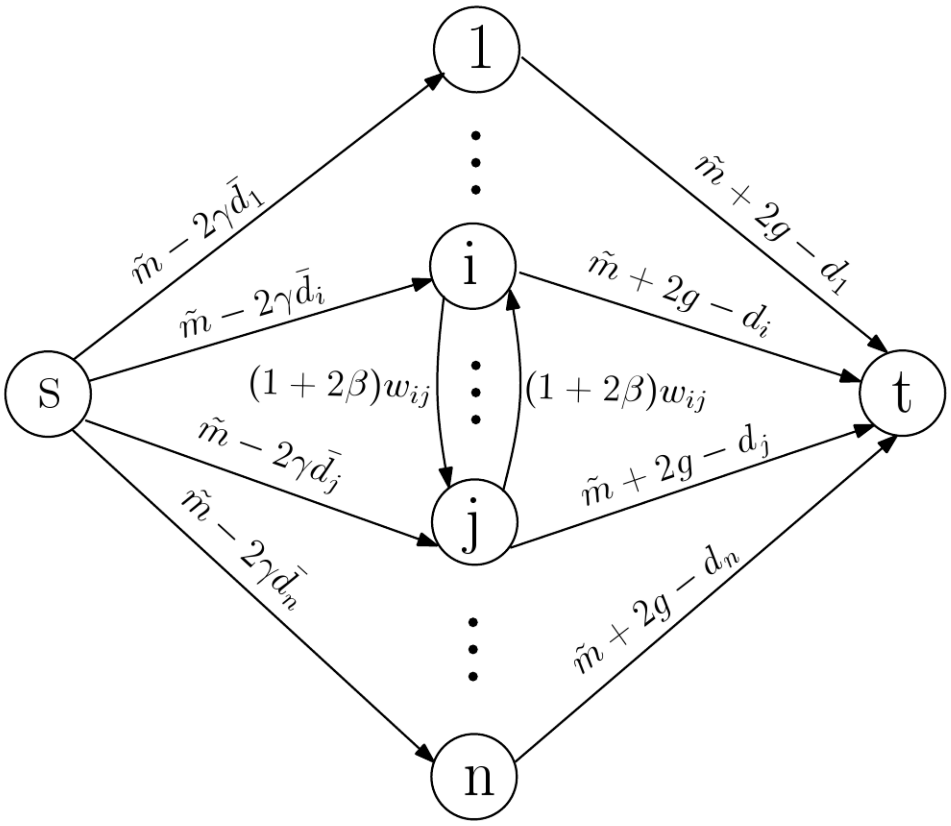


Figure 34: A flow network for detecting a most cohesive subgraph. Nodes $1, \dots, n$ are the original graph's nodes and s, t are added as a source and a sink respectively.

For any $v \in V$, $d_v \leq \mathrm{W(V)}$, $\bar{d}_v \leq n - 1$ because a node may be connected to at most all other nodes. As we shall see, our guess always satisfies $W(V) \geq g \geq -\beta W(V) - \gamma n^2$, which implies that $\tilde{m} + 2g - d_i \geq 0$. Moreover, $\tilde{m} - 2\gamma\bar{d}_j \geq 0$.

A partition of V_N into two sets, S and T, such that $s \in S$ and $t \in T$, determines an *s-t cut*. Let $V_1 = S \setminus \{s\} \subseteq V$, and $V_2 = T \setminus \{t\} = V \setminus V_1$. We denote the value of the cut defined by $V_1 \subseteq V$ in $N(g)$ by $c(V_1)$:

Definition 3:
$$c(V_1) = c(S,T) = c\big(\{s\} \cup V_1, \{t\} \cup (V \setminus V_1)\big) = \sum_{i \in S, j \in T} c_{ij}$$

Theorem 1. For any $V_1 \subseteq V$, $c(V_1) = C_0 + 2|V_1|(g - D(V_1))$, where C_0 is a constant and $D(V_1)$ is the cohesion of V_1.

Proof. If $V_1 = \emptyset$, then the capacity of the cut satisfies $c(\emptyset) = \tilde{m}|V| - 2\gamma \sum_{i \in V} \bar{d}_i = C_0 > 0$. Otherwise if $V_1 \neq \emptyset$, the capacity of the cut is given by (see Figure 35):

$$c(S,T) = \sum_{i\in S, j\in T} c_{ij} = \sum_{j\in V_2} c_{sj} + \sum_{i\in V_1} c_{it} + \sum_{i\in V_1, j\in V_2} c_{ij} =$$

Here each parenthesis corresponds to a sum.

$$= \left(\widetilde{m}|V_2| - 2\gamma \sum_{j\in V_2} \bar{d}_j\right) + \left(\widetilde{m}|V_1| + 2g|V_1| - \sum_{i\in V_1} d_i\right) + \left(\sum_{\substack{i\in V_1, j\in V_2\\(i,j)\in E}} (1+2\beta) w_{ij}\right) =$$

Since $\sum_{j\in V} \bar{d}_j = \sum_{j\in V_1} \bar{d}_j + \sum_{j\in V_2} \bar{d}_j$, rewriting the second term we get:

$$= \widetilde{m}(|V_1| + |V_2|) - 2\gamma \sum_{j\in V} \bar{d}_j + 2\gamma \sum_{i\in V_1} \bar{d}_i + 2g|V_1| - \sum_{i\in V_1} d_i + \sum_{\substack{i\in V_1, j\in V_2\\(i,j)\in E}} w_{ij}$$

$$+ \sum_{\substack{i\in V_1, j\in V_2\\(i,j)\in E}} 2\beta w_{ij}$$

$$= \left(\widetilde{m}n - 2\gamma \sum_{j\in V} \bar{d}_j\right)$$

$$+ 2|V_1| \left(g\right.$$

$$\left. - \left(\frac{\left(\sum_{i\in V_1} d_i - \sum_{i\in V_1, j\in V_2} w_{ij}\right)/2 - \beta \sum_{i\in V_1, j\in V_2} w_{ij} \; - \gamma \sum_{i\in V_1} \bar{d}_i}{|V_1|}\right)\right) =$$

$$= C_0 + 2|V_1|(g - D(V_1))$$

Where,

$$C_0 = \widetilde{m}n - 2\gamma \sum_{j\in V} \bar{d}_j$$

C_0 is a positive constant that does not depend on the cut but only on the graph, and

$$D_1 = D(V_1) = \frac{\left(\sum_{i\in V_1} d_i - \sum_{i\in V_1, j\in V_2} w_{ij}\right)/2 - \beta \sum_{i\in V_1, j\in V_2} w_{ij} \; - \gamma \sum_{i\in V_1} \bar{d}_i}{|V_1|}$$

Notice that $\left(\sum_{i\in V_1} d_i - \sum_{i\in V_1, j\in V_2} w_{ij}\right)/2$ is the weight of edges in the subgraph of G induced by V_1, so

$$D_1 = D(V_1) = \frac{W(V_1) - \beta W(V_1, V_2) - \gamma \sum_{v\in V_1} \bar{d}_v}{|V_1|}$$

is, by Definition 2, the cohesion of the subgraph of G generated by V_1. We therefore get,

(1) $$c(V_1) = C_0 + 2|V_1|(g - D(V_1)) \square$$

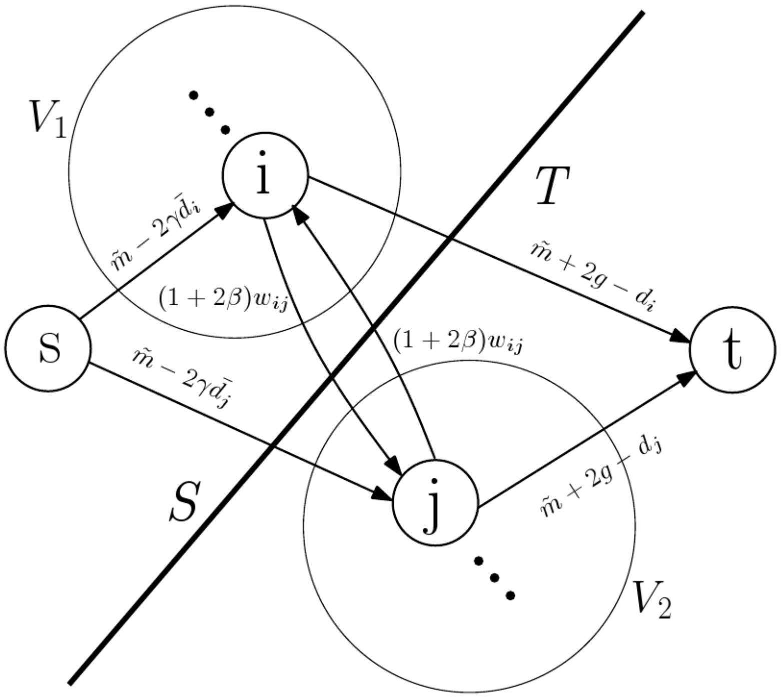


Figure 35: An s-t cut in the flow network $N(g)$.

The following theorem gives a way to tell whether g is too large or too small.

Theorem 2. Assume that (S, T) is a minimum cut in the flow network $N(g)$, where $S = V_1 \cup \{s\}$. If $V_1 \neq \emptyset$, then, $g \leq D^*$; If $V_1 = \emptyset$ (i.e. $S = \{s\}$) then $g \geq D^*$.

Proof. Notice that the value of the cut separating just s from the rest of the graph is, by Theorem 1, $c(\emptyset) = C_0 = \tilde{m}n - 2\gamma \sum_{j\in V} \bar{d}_j$. Hence, the value of min cut is at most C_0. Using Theorem 1 again, the value of the minimum cut satisfies $c(V_1) = C_0 + 2|V_1|\left(g - D(V_1)\right) \leq C_0$, so $2|V_1|(g - D_1) \leq 0$. If $V_1 \neq \emptyset$ then this inequality implies $g \leq D(V_1) \leq D^*$, i.e., there exists a subgraph of G whose cohesion is at least g. Thus, $g \leq D^*$.

If $V_1 = \emptyset$ then the min-cut value is simply C_0. For any other $(\tilde{S}, \tilde{T})$ cut with $\tilde{S} = \{s\} \cup \widetilde{V_1}$ we have $\widetilde{V_1} \neq \emptyset$ and $c(\tilde{S}, \tilde{T}) \geq C_0$. By Theorem 1 we get $c(\widetilde{V_1}) = C_0 + 2|\widetilde{V_1}|\left(g - D(\widetilde{V_1})\right) \geq C_0$ and therefore $g \geq D(\widetilde{V_1})$. Denote a maximum cohesion subgraph $\emptyset \neq V_1^* \subseteq V$ for which $D(V_1^*) = D^*$. For such an optimal solution subset we get $g \geq D(V_1^*) = D^*$. □

To analyze the number of iterations, we make two assumptions:

1. All weights are integers.
2. β, γ are integer multiples of $\frac{1}{k}$. Denote $\beta = \frac{\beta'}{k}, \gamma = \frac{\gamma'}{k}$, where $\beta', \gamma', k \in \mathbb{N}$.

The maximum cohesion D^*, can take on a range of values.

$$D^* \in \left\{\frac{\alpha}{n'} \middle| -\beta W(V) - \gamma n^2 \leq \alpha \leq W(V), 1 \leq n' \leq n\right\}$$

Denote $\alpha' = \alpha k$, then $\alpha' \in \mathbb{Z}$ and we can rewrite:

$$D^* \in \left\{\frac{\alpha'}{n'k} \middle| -\beta' W(V) - \gamma' n^2 \leq \alpha' \leq kW(V), 1 \leq n' \leq n\right\}$$

Let $\frac{\alpha'_1}{n_1 k}, \frac{\alpha'_2}{n_2 k}$ be two different possible values of D^*. The difference Δ between these values is

$$|\Delta| = \left|\frac{\alpha'_1}{n_1 k} - \frac{\alpha'_2}{n_2 k}\right| = \frac{|\alpha'_1 n_2 - \alpha'_2 n_1|}{n_1 n_2 k}$$

For every we n_1, n_2 have $n_1 n_2 k \leq n(n-1)k$ and $|\alpha'_1 n_2 - \alpha'_2 n_1| \geq 1$, so $|\Delta| \geq \frac{1}{n(n-1)k}$.

Therefore, $|\Delta| \geq \frac{1}{n(n-1)k}$. We have the following theorem:

Theorem 3. If G' is a subgraph of G with cohesion D', and no subgraph of G has cohesion greater than or equal to $D' + \frac{1}{n(n-1)k}$, then G' is a maximum cohesion subgraph. □

The theorem tells us when we can stop the search. Now we can describe an algorithm to find a maximum cohesion subgraph.

During the execution of the algorithm, V_1 contains nodes of a subgraph of G with cohesion greater than or equal to l. When the algorithm terminates, we know that there is no subgraph with cohesion $l + \frac{1}{n(n-1)k}$ or greater, so, by Theorem 3, the subgraph returned is a maximum cohesion subgraph.

```
Max-cohesion-subgraph(G)
$l \leftarrow -\beta' W(V) - \gamma' n^2$;
$u \leftarrow kW(V)$;
$V_1 \leftarrow \emptyset$;
while $u - l \geq \frac{1}{n(n-1)k}$ do
  begin
        $g \leftarrow \frac{u-l}{2}$;
        Construct $N(g) = (V_N, E_N)$;
        $(S, T) \leftarrow$ Find min-cut $(N(g))$;
        If $S = \{s\}$ then $u \leftarrow g$
          else
            begin
                $l \leftarrow g$;
                $V_1 \leftarrow S \setminus \{s\}$;
            end;
  end;
return ($V_1$)
```

Algorithm 4: Finding a maximum cohesion subgraph

<u>Running time</u>:

Let $M(\lambda, \varphi)$ be the time required to find a minimum capacity cut in a network with λ nodes and φ edges, and let $w_{max} = \max_{e \in E} w_e$.

Theorem 4. The algorithm runs in $O(M(n, n+m) * \log(n * w_{max}))$ time. In particular, when w_{max} is polynomial in n the running time is $O(M(n, n+m) * \log(n))$.

Proof. Initially the search interval is $u - l = kW(V) + \beta' W(V) + \gamma' n^2$, and it is halved in each iteration. By Theorem 3, when its size is smaller than $\frac{1}{n^2 k}$, we are done. Hence, the binary search loop is executed $\eta = \lceil \log\left((kW(V) + \beta' W(V) + \gamma' n^2) n^2 k\right) \rceil$ times. Now, $W(V) \leq n^2 w_{max}$ so for constant k, β' and γ', we get $\eta \leq \log(cn^4 w_{max})$. The flow network contains $n + 2 = \Theta(n)$ nodes and $2n + 2m = \Theta(n + m)$ edges. So the running time is $O(M(n, n + m) * \log(n * w_{max}))$. □

It is also reasonable to define the cohesion for a node weighted graph. In the node weighted graph there is also a weight function for the nodes $w': V \to \mathbb{Q}^+$. The cohesion is then defined as

Definition 4: $$D(S) = \frac{W(S) - \beta W(S, V \setminus S) - \gamma \sum_{v \in S} \overline{d}_v}{\sum_{v \in S} w'(v)}$$

In this formulation, node weights act as the cost of each node. This target function tries to find a cohesive subgraph of "cheap" nodes.

The diligent reader can verify that the algorithm can be modified to handle this definition as well by changing capacities of the edges going into the sink t as follows:

$$C_{it} = \widetilde{m} + 2gw'(i) + d_i \quad \forall i \in V$$

8. Application of cohesion to graph clustering

In this section we test the utility of finding maximum cohesion subgraphs for clustering. We first define several models for generating graph clustering data and then we use them to simulate data and test max density and cohesion algorithms.

8.1. Models for simulating test data

8.1.1. The corrupted clique graph model

To generate simulated data, we used a *random corrupted clique graph* model similar to Ben-Dor et al. [44]. The model starts with a graph G composed from several disjoint cliques. Then we flip each edge/non-edge independently with probability $0 \leq p < \frac{1}{2}$. p is called the *contamination level*.

We now give a formal definition of the model. We also refer the reader to Figure 36.

i. A *cluster structure* is a vector $S = (s_1, s_2 \dots, s_d)$, where $s_i > 0$ and $\sum_i s_i \leq 1$. Denote $s_0 = 1 - \sum_i s_i$.
ii. We say that an n-node graph has a structure $S = (s_1, s_2 \dots, s_d)$ if it consists of d disjoint cliques of size $s_1 n, s_2 n, \dots, s_d n$ and $s_0 n$ singleton nodes. Two nodes from the same clique are called *mates*.

The random graph model $Q(n, p, S)$ (representing random corruption of clique graphs) is defined as follows: Given a clique graph $G = (V, E)$ of n nodes with structure S, and a value $0 \leq p < \frac{1}{2}$, the random graph $\tilde{G}_p$ is obtained from G by randomly:

- Removing each edge in G independently with probability p.
- Adding each edge not in G independently with probability p.

Formally,

$\tilde{G}_p = (V, \tilde{E})$ where $\tilde{E}$ is constructed as follows.

$$\forall u, v \in V, \qquad Pr\left((u,v) \in \tilde{E}\right) = \begin{cases} 1-p & if\ \ (u,v) \in E \\ p & if\ \ (u,v) \notin E \end{cases}$$

Hence, $Q(n, p, S)$ defines a probability for every n-node graph.

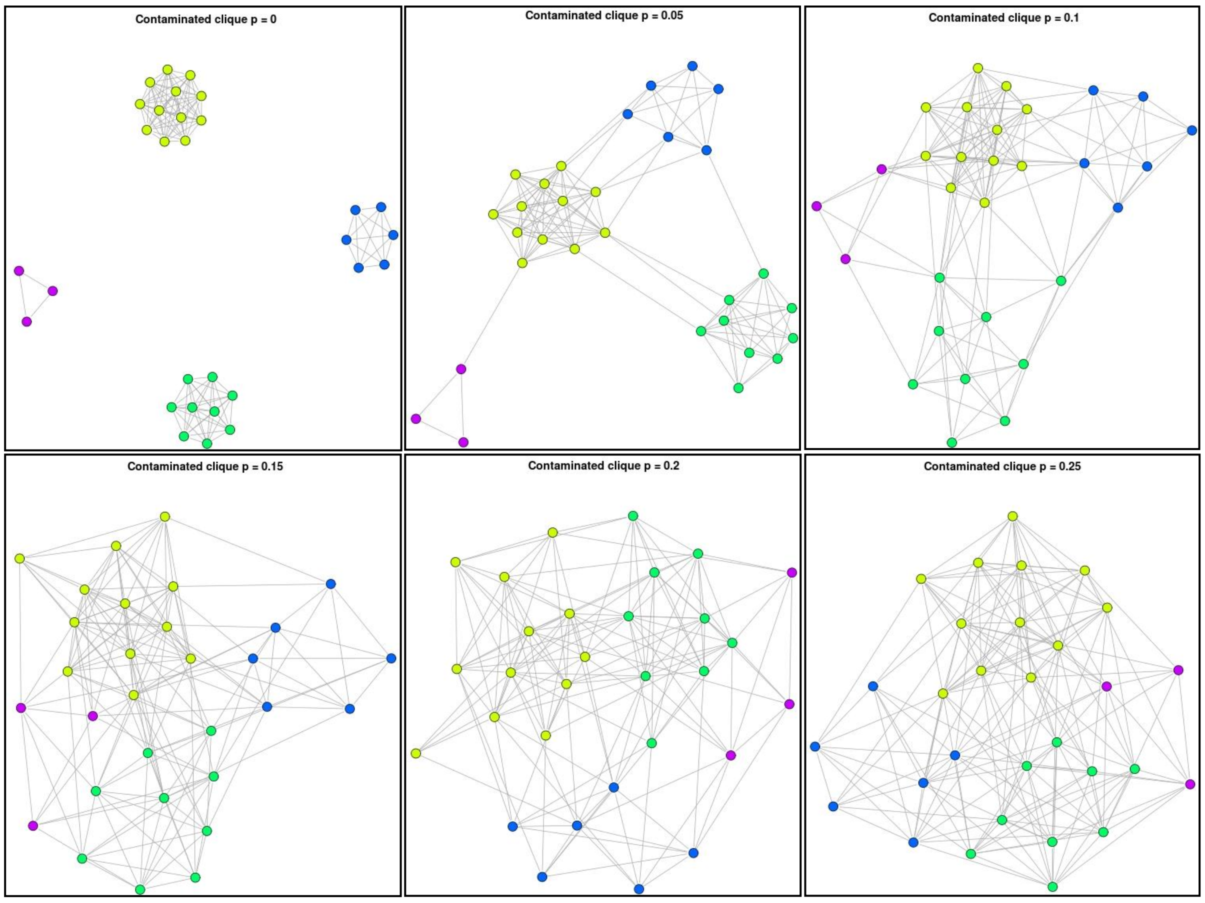

Contaminated clique p = 0
Contaminated clique p = 0.05
Contaminated clique p = 0.1
Contaminated clique p = 0.15
Contaminated clique p = 0.2
Contaminated clique p = 0.25

Figure 36: A visual representation of the corrupted clique model generation. The cluster structure is $S = (0.4,0.3,0.2,0.1), n = 30$. The six plots correspond to different contamination values $0 \leq p \leq 0.25$. The left plot is the original clique graph (i.e. $p = 0$). The rest of the plots correspond to corrupted clique graphs with $p > 0$.

8.1.2. The weighted corrupted clique graph model

In many applications edges have weights. For example, in protein networks, where nodes correspond to proteins and edges to interactions, the intensity or the confidence of protein interactions may be measured and attributed to the graph edges.

To model this, we enhance the model by adding a weight distribution w_{in} of intra-cluster edges and another weight distribution w_{out} of inter-cluster edges. We refer to this model as the *weighted corrupted clique graph* model $Q(n, p, S, w_{in}, w_{out})$. The graph topology is generated in the same manner as the unweighted model. In addition, for each edge generated between cluster mates, a weight is drawn from w_{in}, and for each edge generated between non-mates, a weight is drawn from w_{out}. All draws are independent.

For the edge weights we shall use two distributions:

1. For integers a, b, $a \le b$, a discrete uniform distribution $U(a,b)$ is defined by $P(x = k) = \begin{cases} \frac{1}{b-a+1} & a \le k \le b \\ 0 & o.w \end{cases}$, i.e., weights are selected from the set of integers $\{a \dots b\}$ with equal probability. The mean value of the distribution is $E\big(U(a,b)\big) = \frac{a+b}{2}$ and its variance is $Var\big(U(a,b)\big) = \frac{(b-a+1)^2-1}{12}$.

Uniform weight distribution might not be realistic. We would expect the weight distributions to have a mean value $\mu_{out} < \mu_{in}$ and some distribution overlap of the two distributions. We model this as follows:

2. The chi-square distribution (χ^2) with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables. Formally, $\chi_k^2 = \sum_{i=1}^{k} z_i^2$ where $z_i \sim N(0,1)$ and i.i.d. The mean value of the distribution is $E(\chi_k^2) = k$ and its variance is $Var(\chi_k^2) = 2k$. We shall generate values from the χ^2 distribution with different values of k for w_{in} and w_{out}. Values are rounded up to maintain integrality of the weights. Histograms of distributions with one and two degrees of freedom are presented in Figure 37. For $\lceil \chi_1^2 \rceil$, 85% of values are 1 and the rest are tailing up to 15. For $\lceil \chi_2^2 \rceil$, only 63% of values are 1.

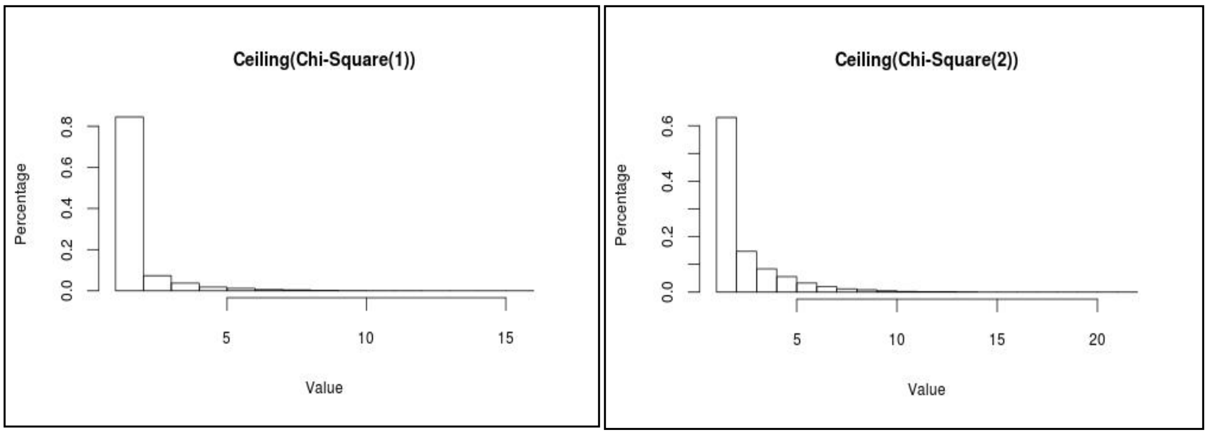


Figure 37: Discrete Chi-square distributions. The plot shows $\lceil \chi^2 \rceil$ for one (left) and two (right) degrees of freedom.

8.1.3. Random similarity graph model

The *random similarity graph* model $Q(n, S, w_{in}, w_{out})$ is defined as follows: Given a complete graph of n nodes with a cluster structure S, the similarity of two nodes of the same cluster s_i are drawn independently from distribution w_{in}; the similarity of two nodes originating from two different clusters is drawn independently from distribution w_{out} The differentiation of clusters depends here only on the level of separation between the similarity distributions w_{in} and w_{out}, unlike previous models where the topology plays a major part.

8.2. Results

We examined the clustering performance of finding maximum cohesion subgraphs on randomly generated instances. The clustering algorithm finds a maximum cohesion subgraph, identifies it as a cluster, and repeats the process on the remaining subgraph. A predefined criterion is used to decide whether a subgraph identified by the algorithm is considered a cluster or outliers. In case the whole graph is identified as the maximum cohesion subgraph we mark it as a cluster. The general scheme is presented in Algorithm 5.

```
Max-cohesion-clustering(G=(V,E))
$\mathcal{C} \leftarrow \{\emptyset\}$;
while $|V| > 0$ do
  begin
        $U \leftarrow MaxCohesionSubgraph(G)$;
        if $(ClusterCriterion(U))$ then $\mathcal{C} \leftarrow \mathcal{C} \cup \{U\}$
        else if $(U = V)$ then $\mathcal{C} \leftarrow \mathcal{C} \cup \{U\}$
                else Mark $U$ as singletons
        $V \leftarrow V \setminus U$;
  end;
return $(\mathcal{C})$
```

Algorithm 5: A clustering algorithm based on finding maximum cohesion subgraphs.

The cluster criterion may be a cohesion threshold, e.g., $Cohesion(U) > \tau$. Another criterion may be based on the contamination estimate of the subgraph, $\hat{p} = 1 - \frac{|E(U)|}{\binom{|U|}{2}}$. We can then reject subgraphs with contamination level higher than a certain threshold, $\hat{p} > \tau$.

We first generated graphs according to the random corrupted graph model with structure $S = (0.4,0.3,0.2,0.1), n = 1000$ and different values of p. We repeated each test 500 times. We used a contamination threshold of $\tau = 0.4$ as a stopping criterion. The algorithm was implemented using a preflow-push-relabel max flow algorithm from LEMON C++ graph library [45]. A full clustering of a 1000-node took 2-3 seconds.

We compared the performance of the original maximum density subgraph algorithm ($\beta = \gamma = 0$) to the maximum cohesion subgraph algorithm (using $\beta = 0, \gamma = 0.5$) in the reconstruction of clusters. The clustering quality assessment was done using Jaccard coefficient as defined in Section 2.2.3.4. The results are presented in Figure 38.

Interestingly, in all runs we noticed that the algorithm outputs a finite set of values corresponding to several clustering options:

- Perfect clustering with Jaccard score of 1.
- Forming one group from the $0.4n$ and $0.3n$ clusters and identifying the $0.2n$ and $0.1n$ clusters. This gives a score of 0.55.
- Identifying only the $0.1n$ cluster and grouping the rest of the graph together. This gives a score of 0.36.
- Identifying the whole graph as one cluster with a score of 0.3.

It follows that, for the simulation parameters, the clustering algorithm has only false positive pairs and almost no false negatives, meaning that if two nodes are in the same cluster in the correct solution they will be in the same output cluster. The algorithm never breaks a cluster but sometimes joins clusters together, even the whole graph in the worst case. We discuss this phenomenon below.

Another interesting property of the algorithm is the lack of solution variation. For every contamination level the prevalent result was also the median result, and obtained the same solution value in over 95% of the simulated data sets.

As we can see in Figure 38, both the cohesion and the density criteria reveal the cluster structure perfectly for $p \leq 0.1$. For $p \geq 0.125$ we see that the density definition deteriorates. On the other hand, cohesion gives perfect clustering until $p \leq 0.175$ and declines after $p \geq 0.2$, still identifying the one or two small clusters.

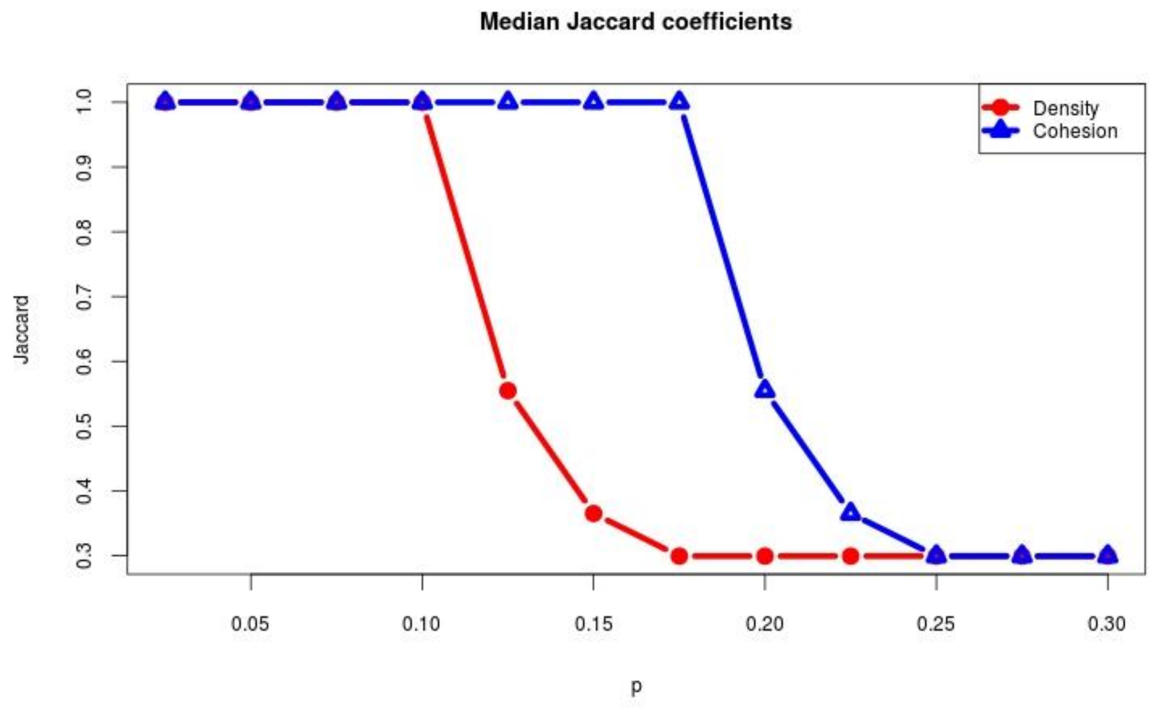


Figure 38: Clustering solutions in the corrupted graph model. Shown are median clustering quality scores of solutions based on density and cohesion for different levels of contamination in the corrupted graph model. Density ($\beta = \gamma = 0$) is presented in red and cohesion ($\beta = 0, \gamma = 0.5$) in blue. Standard deviation values are near zero and thus not shown.

In order to understand the results, we studied in detail the density and cohesion of clusters and groups of clusters for different contamination levels. We used the cluster structure $S = (0.4,0.3,0.2,0.1)$ and $n = 1000$ nodes. As we can see in Figure 39, the $0.4n$ subgraph has the highest density for $p \leq 0.1$, which explains its identification as a separate cluster. For $p > 0.1$ we see a transition, in which the $0.4n + 0.3n$ cluster becomes the densest subgraph around $p = 0.125$, before losing to the $0.4n + 0.3n + 0.2n$ cluster around $p = 0.15$. For contamination levels of $p > 0.15$, the whole graph has the maximum density. This explains the identification merged clusters starting at $p > 0.1$.

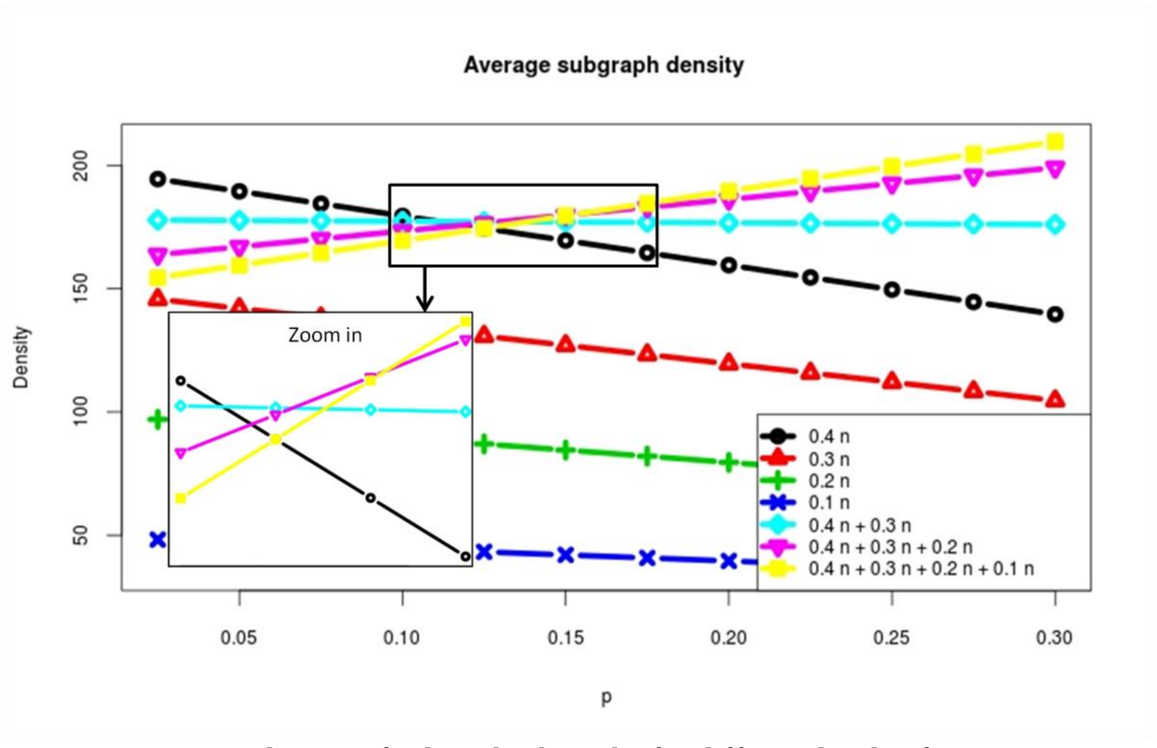


Figure 39: Average density of selected subgraphs for different levels of contamination in the corrupted graph model with cluster structure $S = (0.4, 0.3, 0.2, 0.1)$ and $n = 1000$ nodes.

Cohesion values for the same clusters and cluster groups are presented in Figure 40. The use of cohesion preserves the superiority of the $0.4n$ subgraph until $p \leq 0.2$. For higher values, the aggregate clusters have better cohesion. The results match those in Figure 38.

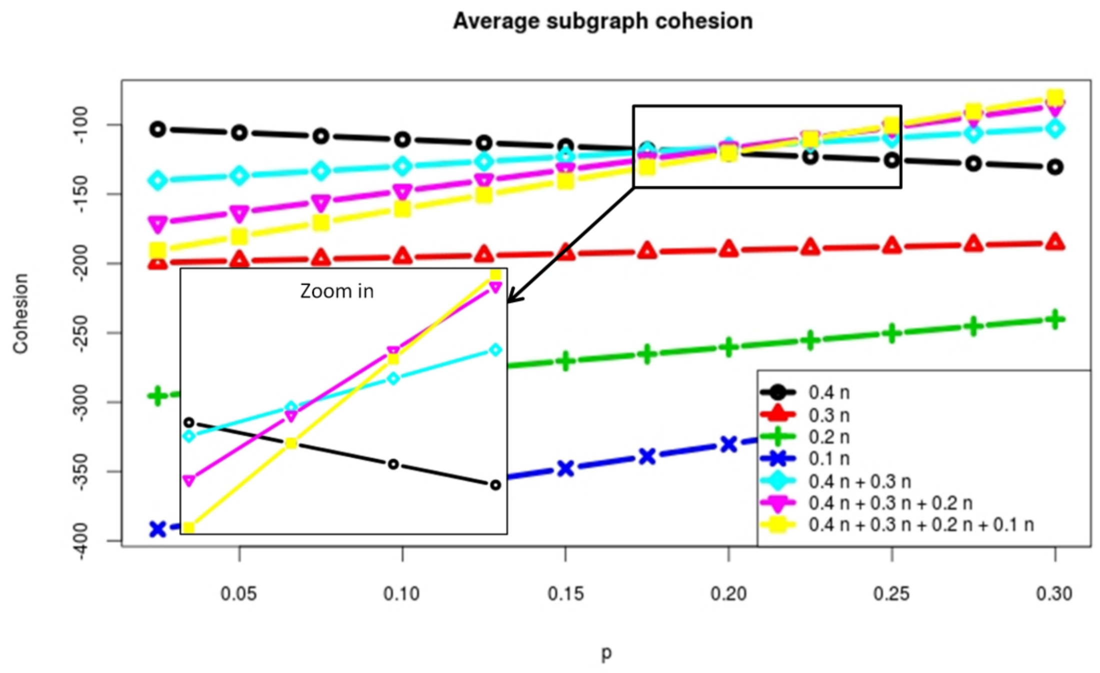


Figure 40: Average cohesion ($\beta = 0, \gamma = 0.5$) of selected subgraphs for different levels of contamination for the corrupted graph model.

To understand why the density algorithm rarely breaks a real cluster for our parameter set, take a cluster with k nodes. The expected density of such subgraph under the model is $\frac{(1-p)\binom{k}{2}}{k} = \frac{(1-p)(k-1)}{2}$. For any of its subgraphs with $k' < k$ nodes we have density $\frac{(1-p)(k'-1)}{2} < \frac{(1-p)(k-1)}{2}$. So at least when cluster and subcluster densities are close to their expected values, any split of a cluster will not result in better density.

The same analysis can be done for the expected cohesion value. The expected complement degree for a node v in a k node cluster is $(k-1)p$ for corrupted intra-cluster connections plus $(n-k)(1-p)$ for added inter-cluster connections. Because all nodes are independent, the expected cohesion of a k node cluster is

$$\frac{(1-p)(k-1)}{2} - \frac{\gamma k[p(k-1)+(n-k)(1-p)]}{k} = \left[\frac{1}{2} - \frac{p}{2} - \gamma p + \gamma - \gamma p\right] k + b(p,\gamma,n)$$

where $b(p,\gamma,n)$ is a function that does not depend on k. Hence, the expected cohesion is

(2) $$\left[\frac{1}{2} + \gamma - p\left(\frac{1}{2} + 2\gamma\right)\right] k + b(p,\gamma,n)$$

So the expected cohesion is a linear function of k with a slope of $a = \frac{1}{2} + \gamma - p\left(\frac{1}{2} + 2\gamma\right)$. For our parameter choice $\gamma = \frac{1}{2}$ we get, $a|_{\gamma=\frac{1}{2}} = 1 - \frac{3}{2}p$. The slope is positive iff $p < \frac{2}{3}$, which is always true. Thus, any subset of $k' < k$ nodes from the cluster is expected to have a lower value of cohesion.

Figure 41 compares density to cohesion with parameter values $\beta = 0.5$ and $\gamma = 0$. All other parameters are the same. We can see that this cohesion version clusters correctly only for $p \leq 0.05$. Hence, in all experiments we chose not use the β parameter since it gave inferior results in comparison to density.

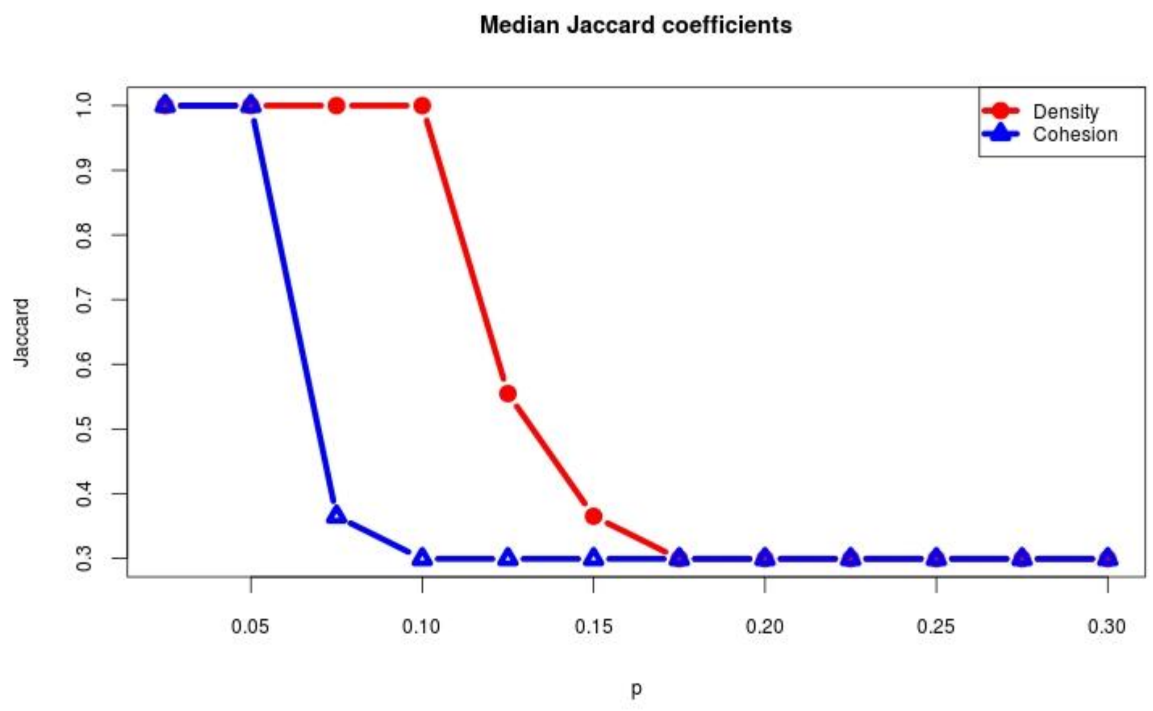


Figure 41: The effect of the β parameter on the cohesion performance. The graph shows the median clustering quality scores of solutions based on density and cohesion for different levels of contamination in the corrupted graph model. Density ($\beta = \gamma = 0$) is presented in red and cohesion ($\beta = 0, \gamma = 0.5$) in blue.

In the corrupted clique graph model, for the same cluster structure, graphs with less nodes are more susceptible to noise. To see this effect, we chose a certain contamination level and measured the clustering quality for different sizes of graphs with the same cluster structure. We chose contamination level to be the first probability for which the algorithms stopped identifying the all clusters (see Figure 38). We ran the density algorithm with contamination level of $p = 0.125$ and cohesion with $p = 0.2$ for different values of n. Results are presented in Figure 42. We can clearly see that for graphs with $n \geq 600$ nodes there is no variation in the algorithm's results.

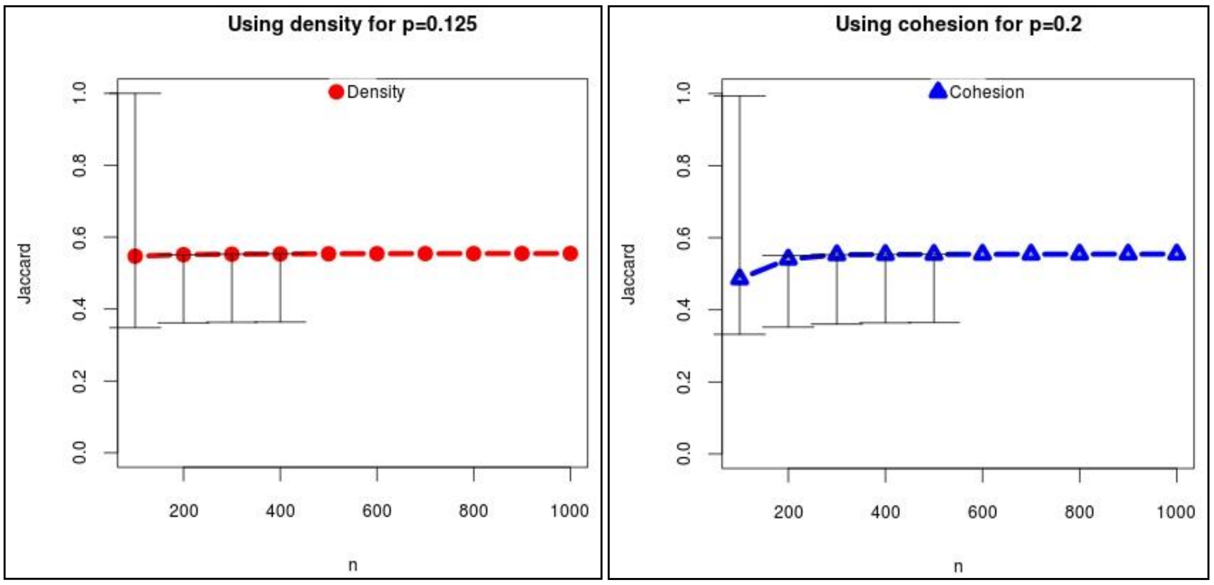


Figure 42: Clustering quality scores of solutions based on density and cohesion for different graph sizes. Density (left) was used with contamination level of $p = 0.125$ and cohesion (right) with $p = 0.2$. Error bars represent the 95% and 5% quantiles. Density ($\beta = \gamma = 0$) is presented in red on the left and cohesion ($\beta = 0, \gamma = 0.5$) in blue on the right.

As a second test, we used the weighted corrupted graph model on 1000 node graphs with cluster structure $S = (0.4, 0.3, 0.2, 0.1)$, weight distributions $w_{in} \sim U(1,2)$ and $w_{out} \sim U(1,1)$ (i.e. $w_{out} = 1$) and different values of p. Each parameter set was tested on 500 instances. The median clustering quality scores of solutions based on density and cohesion for different levels of contamination are presented in Figure 43. As expected, better results were obtained in comparison to the unweighted graphs (Figure 38). Both objectives cluster perfectly for $p \leq 0.15$. The cohesion version can handle $p \leq 0.2$ while the density objective finds at most two small clusters in the same contamination levels. For $p \geq 0.225$ density cannot detect any cluster while the small clusters are still detectable using cohesion. Both versions cannot handle $p \geq 0.3$

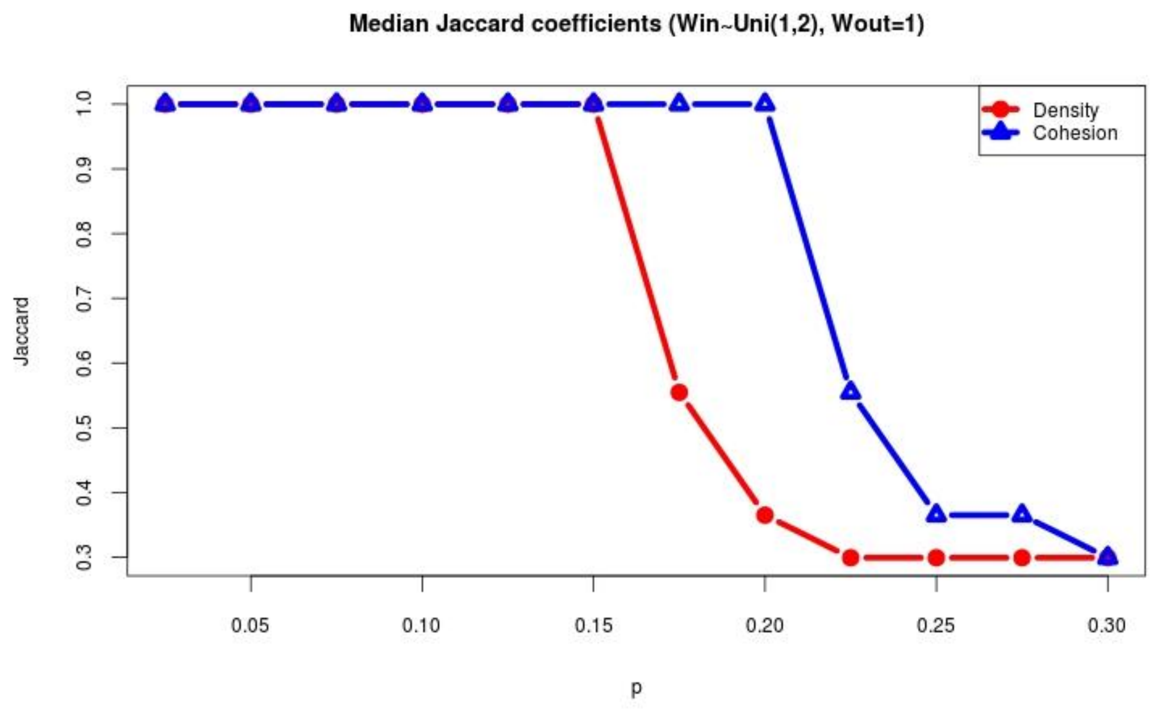


Figure 43: Performance under the weighted corrupted clique graph model. The graph shows the median clustering quality scores of solutions based on density and cohesion for different levels of contamination in the weighted corrupted clique graph model. Edge weights are distributed $w_{in} \sim U(1,2)$ and $w_{out} \sim U(1,1)$. Density $(\beta = \gamma = 0)$ is presented in red and cohesion $(\beta = 0, \gamma = 0.5)$ in blue.

Next, we examined the performance of the algorithms on data generated using the weighted corrupted clique graph model with weight distributions $w_{in} \sim \lceil \chi_2^2 \rceil$ and $w_{out} \sim \lceil \chi_1^2 \rceil$. The rest of the parameters were the same as in previous tests. The clustering quality scores are presented in Figure 44. As above, cohesion seems to be a better objective than density. There is little difference between the results for uniform distribution (Figure 43) and for the chi-square distribution in the density definition. On the other hand, cohesion slightly deteriorates when the chi-square distribution is used.

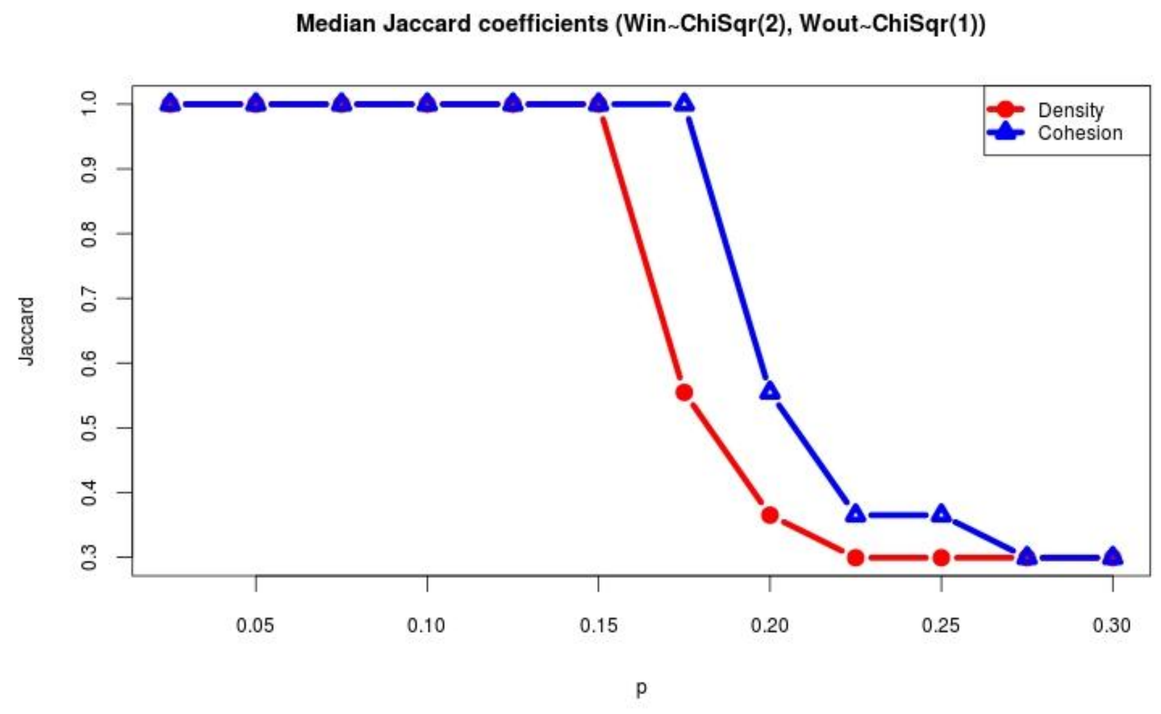


Figure 44: Performance under the weighted corrupted clique graph model. The graph shows the median clustering quality scores of solutions based on density and cohesion for different levels of contamination in the weighted corrupted clique graph model. Edge weights are distributed $w_{in}\sim[\chi_2^2]$ and $w_{out}\sim[\chi_1^2]$. Density ($\beta = \gamma = 0$) is presented in red and cohesion ($\beta = 0, \gamma = 0.5$) in blue.

Our next experiment studied the similarity graph model. We tested the performance of density and cohesion-based clustering for different similarity distributions of inter and intra-cluster pairs. The mean and variance of the distributions was kept within a multiplicative distance factor. The inter-cluster edge similarity distribution was fixed to $w_{out}\sim[\chi_5^2]$ in all graphs. The intra-cluster similarity distribution was $w_{in}\sim[\chi_{5x}^2]$, for x ranging from 4 to 10. We used the same cluster structure $S = (0.4,0.3,0.2,0.1)$ with $n = 1000$ nodes and 500 repeats for each combination of the parameters.

The clustering quality as a function of x is presented in Figure 45. Both algorithms can handle perfectly the case where the intra-cluster similarity values are on average 10 times larger than inter-cluster values. The cohesion version also solves perfectly for $x = 8$. It also has better results for $x = 6$, finding the $0.1n$ and $0.2n$ clusters. A value of $x = 4$ is too hard for both algorithms.

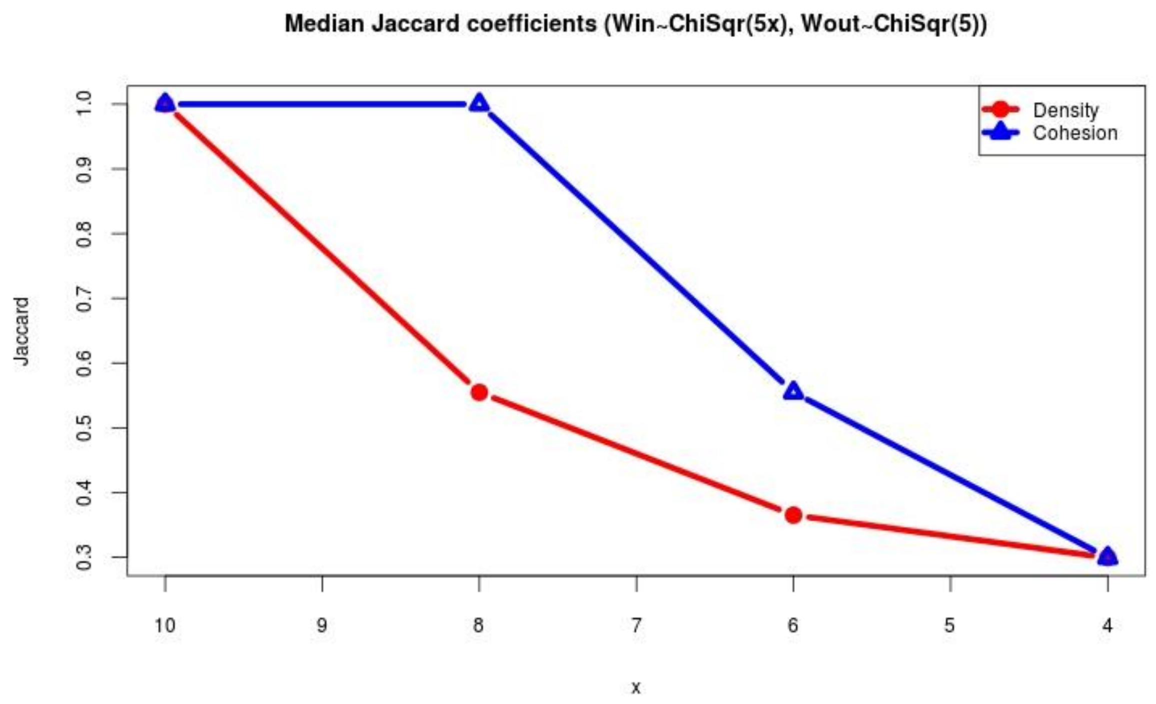


Figure 45: Performance under the random similarity graph model. The graph shows the median clustering quality scores of solutions based on density and cohesion for the random similarity graph model. Edge weights are distributed $w_{out} \sim \lceil \chi_5^2 \rceil$ and $w_{in} \sim \lceil \chi_{5x}^2 \rceil$. Density ($\beta = \gamma = 0$) is presented in red and cohesion ($\beta = 0, \gamma = 0.5$) in blue.

8.3. Discussion

All experiments show the superiority of cohesion over density. This is especially noticeable in the unweighted graph model. This is reasonable since $W(S)$ depends on the edge weights, while $\overline{d_v}$ depends only on the degrees and does not change in weighted graphs. A possible solution could be defining a weighted complement degree, for example $\overline{d_v} \triangleq n * \max_{e=(u,v)\in E} w_e - d_v$.

As shown in Equation (2), the expected cohesion value under the random corrupted clique model is linearly dependent on the cluster size. We also saw (Figure 40) that for our parameter selection, as the contamination level increases, more clusters and cluster sets achieve maximum cohesion. For high contamination rates the whole graphs is the most cohesive subgraph. So cohesion tends to prefer large subgraphs even if they contain several 'true' clusters. On the other hand, breaking a true cluster did not improve the cohesion value. At least for our parameter set, the clustering algorithm has only false positive and almost no false negative pairs. In other words, if two nodes are in the same cluster in the true solution they will be in the same output cluster in the solution produced by the algorithm.

As seen in Figure 39 and Figure 40, density and cohesion values of clustering groups behave linearly with respect to p. Also, the expected density of a subset $U \subseteq C$ of some cluster $C \in \mathcal{C}$ is roughly linear in the subset size: $\frac{(1-p)(|U|-1)}{2}$. A similar near-linear relation applies for the expected cohesion. As seen in the above figures, merging more clusters together changes the slope, causing the aggregated cluster to have higher cohesion. When using cohesion we can change the parameter γ (and possibly β) to affect the linear relation and enable the algorithm to cluster the graph better for higher levels of contamination. Further investigation can help tune the parameters for better clustering. This can be done either empirically using graphs such as Figure 40, or theoretically by analyzing the expected cohesion as in equation (2).

A quite surprising result is the detection of small clusters. When not detecting all clusters or a single cluster, the algorithm actually detects the smallest clusters of $0.2n$ and $0.1n$. The actual process is, first, detecting the several biggest clusters as one cluster, and then detecting the small clusters of $0.2n$ and $0.1n$. In contrast to most clustering algorithms, maximum cohesion is better at detecting small clusters than large ones. It might be beneficial to combine this method with a coarser clustering algorithm to exploit the strengths of both.

A possible solution to merged cluster detection is applying the maximum cohesion clustering algorithm recursively, i.e., reapplying it on the maximum cohesion subgraph detected, in addition to the rest of the graph. Unlike density, cohesion depends not only on the selected subgraph, but also on the remaining subgraph, due to the term $\gamma \sum_{u \in U} \bar{d}_u$. If a subgraph U is a maximum cohesion subgraph for a graph G, it might not be a maximum cohesion subgraph for the graph induced by U.

A major disadvantage of both density and cohesion is symmetry. Since both objectives are defined by a ratio between edges and nodes, multiplying the numerator and denominator by a constant factor keeps the objective function the same. For example, consider a graph composed of two disjoint cliques of size k. The density of each clique is $\frac{\binom{k}{2}}{k} = \frac{k-1}{2}$ and so is the density of the whole graph $\frac{2\binom{k}{2}}{2k} = \frac{k-1}{2}$. For example, when running the clustering algorithms on a cluster structure of $S = (0.25, 0.25, 0.25, 0.25)$, no cluster was detected for any level of contamination. Both density and cohesion gave as a solution a single cluster of the entire graph for the above structure.

Finally, the results of this section should be viewed as preliminary and require further tests on real data, more diverse cluster structures, further parameter exploration and testing on other models. Comparison to other clustering algorithms is also required.

9. Future work

Our analysis of one industrial post-silicon test set show that combinatorial, graph theoretic and bioinformatics methods can help in revealing structure and redundancy in the data. Coverage and domination analysis suggest ways to trim the data and potentially can reduce the set of tests radically without loss of coverage. Bioinformatics methods can help in revealing subtle relations between tests and events and their relation to inner parameters. In our discussion with practitioners the visualizations were found to be very helpful and revealing. The use of cohesion has yet to be tested on real post-silicon data.

Further evaluation of our methods should be done on more diverse Post-silicon test data. In addition, cooperation with validation teams is needed to show our result's practicality in post-silicon optimization. Mapping clusters to certain chip functionalities, e.g. memory, can strengthen clustering solution validity. Measuring coverage rate when running a reordered test suite would demonstrate the importance of test order. Analysis of significant configurations in similar tests can lead to further understanding of the system and the development of new tests.

As mentioned in Section 8.3, cohesion requires further validation on real data and diverse cluster structure, in addition to comparison to other methods. When the underlying data generation model is known, we suggested analyzing the expected cohesion in order to calculate the optimal parameters. Another way of choosing parameters can be done with consensus methods as presented in Section 6.5. Combining the maximum cohesion subgraph clustering algorithm with a coarser clustering algorithm can use benefits of both and overcome their disadvantages.

A clustering objective function based on cohesion/density can be formulated and then is likely to be proven NP-hard. The greedy maximum cohesion subgraph clustering algorithm (Algorithm 5) can be analyzed as a possible approximation of the objective function. Developing further algorithms or heuristics to try and solve the problem is needed.

10. References:

[1] Intel Corporation, “Intel Platform and Component Validation,” 2003. [Online]. Available: http://download.intel.com/design/chipsets/labtour/PVPT_WhitePaper.pdf.

[2] N. Mitra, S. and Seshia, S.A. and Nicolici, “Post-silicon validation opportunities, challenges and recent advances,” in *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC)*, 2010, pp. 12–17.

[3] N. Hakim, “Introduction to Post-Silicon Validation,” *manuscript*, 2010. [Online]. Available: https://embedded.eecs.berkeley.edu/eecsx44/lectures/NagibHakim-PostSiValidation.pdf.

[4] B. Bentley, “Validating the Intel (R) Pentium (R) 4 microprocessor,” in *Proceeding of Design Automation Conference*, 2001, vol. 1, pp. 244–248.

[5] L. R. Ford Jr, D. R. Fulkerson, and A. Ziffer, “Flows in networks,” *Physics Today*, vol. 16, p. 54, 1963.

[6] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, 1st ed. Prentice Hall, 1993.

[7] V. V Vazirani, *Approximation algorithms*. Springer, 2004, pp. 15–17.

[8] J. Edmonds and R. M. Karp, “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems,” *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972.

[9] B. Mirkin, *Clustering for Data mining: A Data Recovery Approach*. Chapman & Hall/CRC, 2005.

[10] A. Jain, M. Murty, and P. Flynn, “Data clustering: a review,” *ACM Computing Surveys (CSUR)*, vol. 31, no. 3, 1999.

[11] C. D. Michener and R. R. Sokal, “A quantitative approach to a problem in classification,” *Evolution*, pp. 130–162, 1957.

[12] N. Saitou and M. Nei, “The neighbor-joining method: a new method for reconstructing phylogenetic trees,” *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406–425, Jul. 1987.

[13] R. Sharan and R. Shamir, “CLICK: a clustering algorithm with applications to gene expression analysis,” in *Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 2000, vol. 8, no. 307, p. 16.

[14] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, Nov. 1987.

[15] Y. Benjamini and Y. Hochberg, “Controlling the false discovery rate: a practical and powerful approach to multiple testing,” *Journal of the Royal Statistical Society. Series B (Methodological),* pp. 289–300, 1995.

[16] J. W. Schneider and P. Borlund, “Matrix comparison, Part 1: Motivation and important issues for measuring the resemblance between proximity measures or ordination results,” *Journal of the American Society for Information Science and Technology*, vol. 58, no. 11, pp. 1586–1595, Sep. 2007.

[17] J. Schneider and P. Borlund, “Matrix comparison, Part 2: Measuring the resemblance between proximity measures or ordination results by use of the Mantel and Procrustes statistics,” *Journal of the American Society for Information Science and Technology*, vol. 58, no. July, pp. 1596–1609, 2007.

[18] A. V Goldberg, “Finding a maximum density subgraph,” *Technical Report UCB/CSB 84/171*. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, 1984.

[19] M. Charikar, “Greedy approximation algorithms for finding dense components in a graph,” *Approximation Algorithms for Combinatorial Optimization*, pp. 139–152, 2000.

[20] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, “Dense Subgraphs with Restrictions and Applications to Gene Annotation Graphs,” in *Research in Computational Molecular Biology*, vol. LNBI 6044, B. Berger, Ed. Springer Berlin Heidelberg, 2010, pp. 456–472.

[21] R. Andersen, “Finding large and small dense subgraphs,” *arXiv preprint cs/0702032*, pp. 1–12, 2007.

[22] S. Khuller and B. Saha, “On finding dense subgraphs,” in *ICALP*, 2009, vol. 5555, pp. 597–608.

[23] U. Feige, G. Kortsarz, and D. Peleg, “The Dense k-Subgraph Problem,” *Algorithmica*, vol. 29, pp. 410–421, 2001.

[24] S. Khot, “Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique,” in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, 2004, pp. 136–145.

[25] R. M. Karp, “Reducibility among Combinatorial Problems,” *Complexity of Computer Computations*, p. 85, 1972.

[26] O. J. Dunn, “Multiple Comparisons among Means,” *Journal of the American Statistical Association*, vol. 56, no. 293, pp. 52–64, Mar. 1961.

[27] A. Bonato, F. Chung, R. Andersen, and K. Chellapilla, “Finding Dense Subgraphs with Size Bounds,” in *Algorithms and Models for the Web-Graph*, vol. 5427, K. Avrachenkov, D. Donato, and N. Litvak, Eds. Springer Berlin Heidelberg, 2009, pp. 25–37.

[28] H. Rotithor, “Postsilicon validation methodology for microprocessors,” *IEEE Design & Test of Computers*, no. December, pp. 77–88, 2000.

[29] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, “A reconfigurable design-for-debug infrastructure for SoCs,” in *Proceedings of the 43rd annual conference on Design automation - DAC*, 2006, p. 7.

[30] H. B. Carter and S. Hemmady, *Metric-driven Design Verification: An Engineer’s and Executive's Guide to First Pass Success*. Springer, 2007.

[31] M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, and M. Vinov, “Industrial experience with test generation languages for processor verification,” in *Proceedings of the 41st annual conference on Design automation - DAC*, 2004, p. 36.

[32] A. Piziali, *Functional Verification Coverage Measurement and Analysis (Google eBook)*. Springer, 2004, p. 213.

[33] S. Fine and A. Ziv, “Coverage directed test generation for functional verification using bayesian networks,” in *Proceedings of the 40th conference on Design automation - DAC*, 2003, p. 286.

[34] L.-C. Wang, “Coverage-directed test generation through automatic constraint extraction,” in *Proc. IEEE International High Level Design Validation and Test Workshop*, 2007, pp. 151–158.

[35] D. Baras, S. Fine, L. Fournier, D. Geiger, and A. Ziv, “Automatic boosting of cross-product coverage using Bayesian networks,” *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 3, pp. 247–261, May 2010.

[36] M. Braun, S. Fine, and A. Ziv, “Enhancing the efficiency of Bayesian network based coverage directed test generation,” in *Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop*, 2004, pp. 75–80.

[37] S. Fine and A. Ziv, “Enhancing the control and efficiency of the covering process,” in *Proceedings of the Eighth IEEE International High-Level Design Validation and Test Workshop*, 2003, pp. 96–101.

[38] D. S. Hochbaum and D. B. Shmoys, “A Best Possible Heuristic for the k-Center Problem,” *Mathematics of Operations Research*, vol. 10, no. 2, pp. 180–184, May 1985.

[39] T. S. Huang, “One-class SVM for learning in image retrieval,” in *Proceedings of the International Conference on Image Processing*, 2001, vol. 1, pp. 34–37.

[40] R. Sharan, A. Maron-Katz, and R. Shamir, “CLICK and EXPANDER: a system for clustering and visualizing gene expression data,” *Bioinformatics*, vol. 19, no. 14, pp. 1787–1799, 2003.

[41] T. Li, “A Unified View on Clustering Binary Data,” *Machine Learning*, vol. 62, no. 3, pp. 199–215, Jan. 2006.

[42] S. Monti, P. Tamayo, J. Mesirov, T. Golub, P. Sebastiani, I. S. Kohane, and M. F. Ramoni, “Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data,” *Machine Learning*, vol. 52, no. 1–2, pp. 91–118, 2003.

[43] R. Giancarlo, D. Scaturro, and F. Utro, “Computational cluster validation for microarray data analysis: experimental assessment of Clest, Consensus Clustering, Figure of Merit, Gap Statistics and Model Explorer.,” *BMC bioinformatics*, vol. 9, p. 462, Jan. 2008.

[44] A. Ben-Dor, R. Shamir, and Z. Yakhini, “Clustering gene expression patterns.,” *Journal of computational biology : a journal of computational molecular cell biology*, vol. 6, no. 3–4, pp. 281–97, Jan. 2004.

[45] B. Dezső, A. Jüttner, and P. Kovács, “LEMON – an Open Source C++ Graph Template Library,” *Electronic Notes in Theoretical Computer Science*, vol. 264, no. 5, pp. 23–45, Jul. 2011.

תקציר

עבודה זו עוסקת בבעיות אופטימיזציה של בדיקות חומרה בעזרת שיטות מביולוגיה חישובית (ביואינפורמטיקה). ניתן לייצג את הפלט מבדיקות החומרה בצורה מתמטית על ידי טבלה ששורותיה מייצגות את הבדיקות שנעשו על החומרה ועמודותיה מייצגות מאורעות חשובים אשר נמדדו במהלך כל בדיקה. ערכי הטבלה הם מספר הפעמים שכל מאורע נצפה במהלך הבדיקה. טבלה זו היא אנלוגית לטבלת ביטוי גנים, בה השורות מתייחסות לגנים והעמודות הן תנאים. כהמשך ישיר לאנלוגיה, ניתן להשתמש בשיטות שפותחו לניתוח ביטויי גנים. למשל, שיטות קיבוץ יכולות לשמש לחלוקה של הבדיקות (או המאורעות) למחלקות דמיון. הקבוצות שיתגלו יכולות להיות מנותחות על ידי מהנדסי הבדיקות כדי למצוא יתירות בבדיקות והחלפתן בבדיקות ייצוגיות. כלים לניתוח ביטויי גנים והצגתם, כדוגמת EXPANDER, יכולים לסייע בהבנת תהליך הבדיקה. בנוסף, אנו בוחנים שיטות קומבינטוריות עבור בעיות כיסוי על מנת למצוא קבוצת בדיקות קטנה בעלת אותה רמת כיסוי.

אנחנו גם מגדירים ובוחנים גישה חדשה לקיבוץ המבוססת על מציאת תתי-גרפים מגובשים בגרף ממושקל לא מכוון. פונקצית המטרה, הנקראת גיבוש (cohesion), היא הכללה של צפיפות (density) של תת-גרף, המוגדרת כיחס בין מספר הקשתות למספר הצמתים בתת הגרף. פונקצית המטרה של גיבוש, שהוגדרה בהשראת קיבוץ גרפים, מקטינה שימוש בקשתות בין קבוצות ובצמתים מדרגה גבוה. אנחנו מפתחים אלגוריתם פולינומיאלי למציאת תת הגרף המגובש ביותר בגרף ממושקל ולא מכוון, המבוסס על סדרה של חישובי זרימה. אנחנו בודקים את הגישה החדשה שלנו על קבוצות אקראיות המיוצרות בעזרת מודלים שונים, ומראים שיפור בביצועים בשימוש בגיבוש לעומת השימוש בצפיפות.

אוניברסיטת תל-אביב
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
בית הספר לתארים מתקדמים ע"ש זנדמן-סליינר

שיפור בדיקות חומרה בעזרת שיטות מביואינפורמטיקה

חיבור זה הוגש כעבודת גמר לקראת התואר "מוסמך אוניברסיטה" בהנדסת חשמל ואלקטרוניקה

על-ידי

רון זעירא

העבודה נעשתה בבית הספר להנדסת חשמל
המחלקה להנדסת חשמל – מערכות
בהנחיית פרופ' רון שמיר ופרופ' דנה רון

שבט התשע"ג

אוניברסיטת תל-אביב
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
בית הספר לתארים מתקדמים ע"ש זנדמן-סליינר

שיפור בדיקות חומרה בעזרת שיטות מביואינפורמטיקה

חיבור זה הוגש כעבודת גמר לקראת התואר "מוסמך אוניברסיטה" בהנדסת חשמל ואלקטרוניקה

על-ידי

רון זעירא

שבט התשע"ג