**Title:** Multiscale Mutation Clustering Algorithm Identifies Pan-Cancer Mutational Clusters Associated with Pathway-Level Changes in Gene Expression

**Short Title:** Multiscale Mutation Clustering in TCGA

**Authors:** William Poole1†, Kalle Leinonen1, Ilya Shmulevich1, Theo Knijnenburg1\*‡, and Brady Bernard1\*‡

1Institute for Systems Biology, Seattle, WA.

†Currently at California Institute of Technology, Pasadena, CA

 \*Corresponding Authors

‡Contributed Equally

**S1 Text: Data, Methods, and Algorithm Details.** This document contains 3 sections. (A) details the data sources and how they were processed. (B) Discusses how genes were chosen for this study. (C) details how the M2C algorithm works.

 **A) Data Sources:**

**Pathway Interaction Database Processing:**

A copy of the National Cancer Institute Pathway Ontology Downloaded in September, 2012, was used to construct a directed graph of pathway relationships. Pathways were labelled as leaves if they had no sub-pathways or children. This set of leaf pathways and the genes they contained within were used to identify pathway cluster associations. This approach was taken because many pathways are highly redundant differing by only a few genes. By only using leaf pathways, we limited this redundancy and thereby unnecessary multiple testing. Pathways and the genes within them are listed in SI Table 9.

**TCGA Firehose:**

Mutation calls and gene expression data were downloaded for call cancer types on 05/17/2015. This data can be downloaded from: http://ezid.cdlib.org/id/doi:10.7908/C1K64H78 or <http://gdac.broadinstitute.org/runs/analyses__2014_10_17/data/>. Filtered raw mutation data is also available in T10 in S1 Tables.

**B) Selecting Genes for Our Clustering Algorithm:**

An initial list of 628 genes was compiled by taking the highest rated genes from Mutsig using a q-value threshold of 0.1. These genes were further filtered to ensure that there are in total at least 10 mutations in each gene added together across all 23 cancer types considered. This resulted in a list of 549 genes which we ran our multiscale clustering algorithm on. The cancer types considered are: ACC, BLCA, BRCA, CESC, COAD, GBM, HNSC, KICH, KIRC, KIRP, LAML, LGG, LIHC, LUAD, LUSC, OV, PRAD, READ, SKCM, STAD, THCA, UCEC, and UCS.

Additionally, we compiled a list of top Mutsig genes by taking the union of the top five genes for each tumor type provides they had a q-value less than or equal to 0.1. This list included many of the most well studied cancer genes.

**C) The Multiscale Information-Based Clustering Algorithm**:

**The Multi-Bandwidth Mixture Model:**

1. **Starting Data:** First, raw TCGA nucleotide mutation calls are converted into amino acid-space for each gene. Each amino acid in a protein is given an index from 1 to­­ *L* where *L* is the length of the protein. Amino acid mutations are classified as synonymous, non-synonymous, and nonsense (insertion, deletion, and stop). The raw data used to identify mutation clusters is the combined set of all kinds of mutations from the 23 tumor types listed above. We include synonymous mutations in our data set because our mixture model explicitly included a uniform distribution to represent background noise, which we also believe is reflected by the synonymous mutations in a gene.
2. **Kernel Density Estimates:** In order to identify multiple scales of mutation clusters, we use kernel density estimates at different bandwidths. Specifically we used Gaussian kernels across the following bandwidth (scales) in amino-acid units: 2, 3, 4, 5, 6, 8, 10, 12, 14, 18, 22, 26, 30, 38, 46, 54, 62, 78, 94, 110, 126, 158, 190, 222, 254, 318, 382, and 446. This process created 28 continuous mutation density estimates denoted *Kh(x)* where *x* is the position along the gene in amino acid-space and is the bandwidth. This set of bandwidths was chosen because it is broad and thoroughly covers the set of possible scales without being overly redundant. Additionally it ensures the algorithm is robust – minor changes in this set of bandwidths do not significantly change the clusters generated by the algorithm (data not shown).
3. **Seeding the Mixture Models:** For each scale *h* in part two, an initial set *nh* of seed Gaussians and one uniform distribution is identified, where *nh* is the number of local maxima in the density estimate. The initial weight of the uniform distribution, denoted *U*, is set as the percentage of synonymous mutations in the gene;

where *M* is the number of mutations in the gene of a given type. The *­ith* Gaussian in the mixture model is defined by 3 variables, its weight, its mean, and its standard deviation denoted *wi, ui,* and *𝜎I* respectively. More specifically,

The parameters *wi, ui,* and *𝜎I* are initially estimated from each of the kernel density estimates, *Kh(x)*, found in step two and later optimized with an expectation maximization (EM) algorithm. Let the amino acid location of the *ith* local maximum be denoted *(ai, Kh(ai))* . This maximum will be bordered by two local minima, *(bi, Kh(bi))* and *(bi+1, Kh(bi+1))* to the left and right, respectively. In the edge cases where there is no local minima before or after , the values *(0, Kh(0))* and *(L, Kh(L))* are used as local minima depending upon the case, where *L* is the length of the gene in amino acids. Then, the initial parameters of *Gi* are given by:

These seed values are then optimized using an EM algorithm resulting in a final set of clusters for each bandwidth. During this process, it is possible for clusters to removed but not created. Thus the final number of clusters will always be less than or equal to *n*.

**Merging Multiscale Clusters**

1. **Creating a Tree of Clusters:** Step 3 of the algorithm produces a large set of clusters across many bandwidths. These clusters are often duplicates and also typically overlap with each other. We wish to find a set of non-overlapping clusters across all the different scales. This is a difficult optimization problem which involves choosing a set of non-overlapping clusters from all the combinatoric possibilities generated by the different mixture models. We simplify this problem by placing all clusters into a binary tree using a hierarchal clustering algorithm. This simplifies the combinatoric optimization problem to that of choosing between (or merging) two clustering options, each represented by a branch in the tree. To do this, we create a pairwise distance matrix between all clusters. First, all clusters with fewer than 15 mutations are excluded. Second all clusters are paired with the uniform distribution from their given mixture model, represented by a single parameter, *Uh*. Effectively this means each cluster is defined by 4 terms – its weight, mean, standard deviation, and corresponding uniform noise distribution. Third, in order to avoid redundancy within the cluster tree and increase efficiency, clusters that contain the same set of mutations (but possibly small differences in their other parameters) are averaged together into one single cluster. Let there be *c* clusters denoted *{G1, G2 … Gc}* which contain the same mutations. These clusters are merged into one new cluster, *Gmerged*, by taking the arithmetic mean of their parameters;

.

Once a total of *N* clusters remain, an *N* by *N* pairwise distance matrix is created. The distance metric between two Gaussian clusters *Gi(x; wi, μi, σi)* and *Gj(x; wj, μj, σj)* is given by the unsigned area between the two curves, specifically

Where *L* is the length of the gene in amino acids.

This matrix is used by Scipy’s hierarchical clustering [1] algorithm to produce a binary cluster tree (using default parameters). This tree consists of all Gaussian clusters from different bandwidths used to create the distance matrix.

1. **Flattening the Tree:** A final set of non-overlapping clusters is identified by flattening the binary tree. This is done using a recursive greedy algorithm which locally maximizes the Akaike Information Criteria with finite size correction (AICc) between two adjacent nodes in the tree. Let *L* be the likelihood that a given set of *g* Gaussians and one uniform distribution emit set of *Mtotal* mutations found a gene. Then,

Where *k = 3g+1* and is the number of parameters in the model.

The greedy algorithm recursively works its way up the tree building a set of non-overlapping clusters. Let *Ti* be a node in the tree with *T0* the root of the tree. Each node *Ti* has a left and a right child, *Cr* and *Cl* respectively. A node is a leaf if it has no children. Furthermore, each leaf represents a Gaussian cluster, *Gi(x; wi, μi, σi)* paired with a uniform distribution, and the mutations inside the cluster. Two clusters *i* and *j* are said to intersect if there exists some mutation *m* such that *m ϵ Gi* and *m ϵ Gi*. When the algorithm reaches a given node *i*, if the node is a leaf it returns the set of clusters *{Gi}.* Otherwise, the algorithm enumerates all the possible sets of non-overlapping clusters from the non-overlapping subsets of clusters returned by *Cr* and *Cl*, denoted *Sr* and *Sl*, respectively. Let this set of sets be called *Ω = {S1…Sk}.* A non-overlapping subset can be defined as

.

For each non-overlapping set *S ϵ Ω*, the AICc is calculated. The algorithm then returns the set *S* which minimizes the AICc to the parent node along with the average uniform distribution from the set of Gaussians contained in *S*. Although, in theory, enumerating all the non-overlapping combinations of nodes is computationally intensive, this is made manageable by the algorithm because each set recursively returned by a child node contains no overlaps, dramatically reducing the number of possible combinations in practice. The pseudo code for the flattening algorithm is as follows:

**)**

 **)**

**.**

Finally, due to the greedy nature of this algorithm occasionally a non-overlapping cluster is missed as the tree is flattened; in other words the when the final set of non-overlapping clusters *Sf* is returned, there exists some *Gi* which does not overlap with any cluster in *Sf* . This is fixed in a final correction step. In a manner akin to the flattening stage, let *Σ={G1…GN}* be the set of all clusters in the tree. The correction step finds all non-overlapping subsets between *Sf* and *Σ.* Again, denoting the set of sets as *Ω = {S1…Sk},* with

.

For each *Si* the AICc is calculated and the *Si* with the minimum AICc is selected as the final multi-scale clustering. This corrections step was only necessary on 20 out of the 549 genes considered.

**Bibliography:**

1. Eric Jones and Travis Oliphant and Pearu Peterson and others. (2001--). SciPy: Open source scientific tools for Python. Scipy.org. 2016: 9, 30. Available: http://www.scipy.org/