iPAC: identifcation of Protein Amino acid Mutations

Gregory Ryslik
Yale University
gregory.ryslik@yale.edu

Hongyu Zhao
Yale University
hongyu.zhao@yale.edu

November 1, 2022

Abstract

The **iPAC** package provides a novel tool to identify somatic mutation clustering of amino acids while taking into account their three dimensional structure. Currently, **iPAC** maps the protein's amino acids into a one dimensional space while preserving, as best as possible, the three dimensional local neighbor relationships. Mutation clusters are then found by considering if pairwise mutations are closer together than expected by chance alone via the the *Nonrandom Mutation Clustering* (NMC) algorithm [Ye et al., 2010]. Finally, the clustering results are mapped back onto the original protein and reported back to the user. A paper detailing this methodology and results is currently in preparation. *Additional methodologies based on different algorithms will be added in the future.*

1 Introduction

Recently, there have been significant pharmacological advances in treating on-congenic driver mutations [Croce, 2008]. Several methods that rely on amino acid mutational clusters have been developed in order to identify these mutations. One of the most recent methods was presented by Ye et al. [2010]. Their algorithm identifies mutation clusters by calculating whether pairwise mutations are closer on the the line than expected by chance alone when assuming that each amino acid has an equal probability of mutation. As their algorithm relies on considering the protein in linear form, it can potentially exclude clusters that are close together in 3D space but far apart in 1D space. This package is specifically designed to overcome this limitation.

Currently, this package has two methods that deal with the 3D structure of the protein: 1) linear and 2) MDS [Borg and Groenen, 1997]. The user should primarily use MDS as it is more statistically rigorous. We include the linear method as an example that the general package is itself flexible. Should the user want to map the protein to 1D space using their own algorithm, they can thus do so.

If users want to contribute to the code base, please contact the author.

2   The NMC Algorithm

The NMC algorithm, proposed by Ye et al. [2010], finds mutational clusters when the protein is considered to be a straight line. While the full alogrithm is presented in their paper, we provide a brief overview here for completeness.

Suppose that the protein was N amino acids long and that each amino acid had a $\frac{1}{N}$ probability of mutation. We can then construct order statistics over many samples as follows:
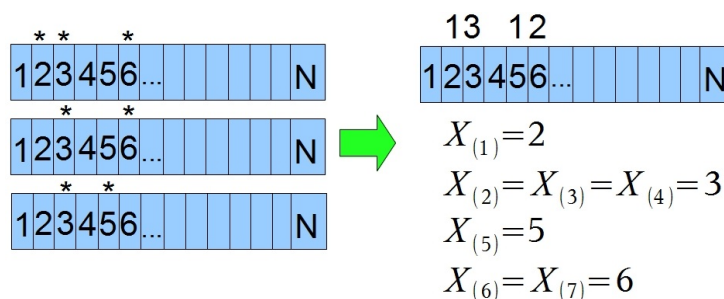

Figure 1: Three samples of the same protein. An asterisk above a number indicates a non-synonomous mutation in that sample for that amino acid.

Letting $R_{k,i} = X_{(k)} - X_{(i)}$, one can calculate if the $Pr(R_{k,i} \leq r) \leq \alpha$ using well known results about order statistics on the uniform distribution. While discrete formulas exist for $Pr(R_{k,i} \leq r)$, they are often too costly to calculate when $R_{k,i} > 1$. In these cases, we scale the protein onto the interval (0,1) by calculating $Pr(\frac{X_{(k)} - X_{(i)}}{N} \leq r)$ which turns out to equal $Pr(Beta(k - i, i + n - k + 1) \leq r)$. Finally, since this calculation is done for every pair of mutations in the protein, a multiple comparisons adjustment is performed.

The original NMC algorithm is included in this package via the *nmc* command. We provide an example of its use below.

First, we load **iPAC** and then the mutation matrix. The mutation matrix is a matrix of 0's and 1's where each column represents an amino acid in the protein and each row represents a sample (or a mutation). Thus, the entry for row i column j, represents the ith sample (or mutation) and the jth amino acid.

Code Example 1: Running the NMC algorithm

```
> library(iPAC)
> #For more information on the mutations matrix,
> #type ?KRAS.Mutations after executing the line below.
> data(KRAS.Mutations)
> nmc(KRAS.Mutations, alpha = 0.05, multtest = "Bonferroni")
```

```
       cluster_size start end number       p_value
V12               2   12  13    131 1.979447e-235
V12               1   12  12    100 6.486735e-188
V12              11   12  22    132 3.220145e-145
V12              12   12  23    133 6.524053e-142
V12              50   12  61    138   4.338908e-65
V13               1   13  13     31   2.732914e-39
V12             106   12 117    139   2.341227e-23
V12             135   12 146    149   1.356584e-20
V13              10   13  22     32   4.487362e-12
V13              11   13  23     33   1.279256e-11
V146              1  146 146     10   1.918440e-08
```

The results from *Code Example 1* show all the statistically significant clusters found, including the size of the cluster, the start and end positions and the number of mutations in that cluster.

3 Remapping Algorithm

3.1 Matching the Mutation and Position Information

Before we can run the 3D clustering algorithm while, we first need to come up with an alignment between the mutational information provided by a source such as COSMIC [Forbes et al., 2008] and the positional information provided by a source such as the PDB [Berman et al., 2000]. Such an alignment is necessary because mutational information is typically provided on the "canonical" amino acid numbering which often differs from the numbering used in the PDB database. Thus amino acid #i from the PDB database might not be amino acid #i from the mutational database.

To solve this problem, we consider the mutational database to contain the "canonical" ordering of the protein. We then attempt to map the structural information to the canonical ordering and create a new matrix of residues, their canonical counts, and their positions in 3D space. If successful, we then have a relational structure between the two databases allowing us to refer to amino acid #i where i represents the same amino acid in both databases.

We have created two methods that allow one to construct such a matrix: 1)*get.Positions* and 2)*get.AlignedPositions*.

The first method, *get.Positions* attemps to create the position matrix directly from the CIF file in the PDB database. It returns a list of several items, the first of which is $Positions, which must later be passed to the *ClusterFind* method. Due to the complexity of CIF files, *get.Positions* currently works on approximately 70% of the structures in the PDB database.

The second method, *get.AlignedPositions* performs a pairwise alignment algorithm to align the canonical protein ordering with the XYZ positions in the PDB. Since *get.AlignedPositions* runs an alignment algorithm, the ordering might not be perfect and we recommend the user to verify the results. However,

from our testing, the alignment procedure works quite well. Furthermore, since *get.AlignedPositions* does not have to consider as many aspects of the CIF file, it is more robust and often works when *get.Positions* fails.

Let us first consider the *get.Positions* function. We will consider three examples, one for KRAS protein and two for the PIK3CA protein. For each example, we need to input the location of the CIF file (this holds the structural information), the location of the FASTA file (this holds the canonical protein sequence) and the sidechain that we want to use from the CIF file.

As the entire position sequence is too long to print, we first save the result and then print the first 10 rows of the position matrix. The remaining elements of the result are printed in full.

Code Example 2: Extracting positions using the get.Positions function

```
> library(iPAC)
> CIF<-"https://files.rcsb.org/view/3GFT.cif"
> Fasta<-"https://www.uniprot.org/uniprot/P01116-2.fasta"
> KRAS.Positions<-get.Positions(CIF, Fasta, "A")
> names(KRAS.Positions)

[1] "Positions"        "External.Mismatch" "PDB.Mismatch"
[4] "Result"

> KRAS.Positions$Positions[1:10,]

   Residue Can.Count SideChain XCoord  YCoord ZCoord
1      MET         1         A 62.935  97.579 30.223
2      THR         2         A 63.155  95.525 27.079
3      GLU         3         A 65.289  96.895 24.308
4      TYR         4         A 64.899  96.220 20.615
5      LYS         5         A 67.593  96.715 18.023
6      LEU         6         A 65.898  97.863 14.816
7      VAL         7         A 67.664  98.557 11.533
8      VAL         8         A 66.263 100.550  8.617
9      VAL         9         A 67.484  99.500  5.194
10     GLY        10         A 66.575 100.328  1.605

> KRAS.Positions$External.Mismatch

  PDB.Residue Canonical.Residue Canonical.Num
1           H                 Q            61

> KRAS.Positions$PDB.Mismatch

   PDB.Residue Canonical.Residue Canonical.Num       Remark
19           H                 Q            61 SEE REMARK 999

> KRAS.Positions$Result
```

```
[1] "OK"
```

Observe that the final element in *Code Example 2* is "OK". That is because the only mismatched residue (at position 61), was documented in the CIF file as well. Thus it is considered a "reconciled" mismatch. It is up to the user to decide if they want to include it in the position sequence that is passed on to the *ClusterFind* method or to remove it.

Code Example 3: Final example of the get.Positions function

```
> CIF <- "https://files.rcsb.org/view/2RD0.cif"
> Fasta <- "https://www.uniprot.org/uniprot/P42336.fasta"
> PIK3CAV2.Positions <- get.Positions(CIF, Fasta, "A")
> names(PIK3CAV2.Positions)

[1] "Positions"         "External.Mismatch" "PDB.Mismatch"
[4] "Result"

> PIK3CAV2.Positions$Positions[1:10,]

   Residue Can.Count SideChain XCoord YCoord  ZCoord
1     GLY         8         A 88.344 61.306 112.918
2     GLU         9         A 90.119 58.543 111.029
3     LEU        10         A 92.954 56.400 109.709
4     TRP        11         A 93.105 53.251 107.542
5     GLY        12         A 91.616 50.221 109.372
6     ILE        13         A 90.825 52.285 112.474
7     HIS        14         A 87.540 54.192 112.953
8     LEU        15         A 88.806 56.633 115.544
9     MET        16         A 92.435 57.520 116.178
10    PRO        17         A 93.481 57.378 119.852

> PIK3CAV2.Positions$External.Mismatch

NULL

> PIK3CAV2.Positions$PDB.Mismatch

NULL

> PIK3CAV2.Positions$Result

[1] "OK"
```

Observe that the final result in *Code Example 3* is "OK". Here we use a different file location for the canonical sequence – the UNIPROT database. Here, the canonical sequence is slightly different and matches up exactly to the extracted positions. As there is only 1 isoform listed on UNIPROT for PIK3CA we suggest using the same source for both the mutational and canonical position

information. For example, if your mutation data was obtained from COSMIC, you should use COSMIC to get the canonical protein sequence.

Let us now consider the *get.AlignedPositions* function. This function automatically drops positions that do not match up.

Code Example 4: Extracting positions using the get.AlignedPositions function

```
> CIF<- "https://files.rcsb.org/view/2RD0.cif"
> Fasta <- "https://www.uniprot.org/uniprot/P42336.fasta"
> PIK3CAV3.Positions<-get.AlignedPositions(CIF,Fasta , "A")
> names(PIK3CAV3.Positions)

[1] "Positions"        "Diff.Count"       "Diff.Positions"   "Alignment.Result"
[5] "Result"

> PIK3CAV3.Positions$Positions[1:10,]

   Residue Can.Count SideChain XCoord YCoord  ZCoord
14     GLY         8         A 88.344 61.306 112.918
15     GLU         9         A 90.119 58.543 111.029
16     LEU        10         A 92.954 56.400 109.709
17     TRP        11         A 93.105 53.251 107.542
18     GLY        12         A 91.616 50.221 109.372
19     ILE        13         A 90.825 52.285 112.474
20     HIS        14         A 87.540 54.192 112.953
21     LEU        15         A 88.806 56.633 115.544
22     MET        16         A 92.435 57.520 116.178
23     PRO        17         A 93.481 57.378 119.852

> PIK3CAV3.Positions$External.Mismatch

NULL

> PIK3CAV3.Positions$PDB.Mismatch

NULL

> PIK3CAV3.Positions$Result

[1] "OK"
```

Both *get.AlignedPositions* and *get.Positions* are still in beta and are provided to the user for convenience only. Changes by the PDB or COSMIC to their file structure might result in errors and it is up to the user to ensure the correct data is supplied to the *ClusterFind* function.

3.2 Finding Clusters in 3D Space

Now that we have the positional data, we can find the mutational clusters while taking into account the 3D protein structure. We begin by slecting a method to map the protein down to a 1D space.

The first method, "linear", fixes a specified point (x_0, y_0, z_0) and then calculates the distance from each amino acid to that point. The amino acids are then rearranged in order from the shortest distance to the longest distance. The second method, "MDS", uses Multidimensional Scaling [Borg and Groenen, 1997] to map the protein to a 1D space. We strongly encourage the user to employ the MDS method as it is more statistically rigorous. The linear method is provided as an example to show how other mapping paradigms might be implemented.

A diagram of either the MDS or linear mapping can be displayed when the *ClusterFind* method is run. As the mapping algorithms are different, the mapping images created are different as well. The linear method will generate an image of the distances from (x_0, y_0, z_0) to each amino acid. These distance will be drawn as dotted green lines from each amino acid to the fixed point. Conversely, the MDS methodology will create lines from each amino acid to the x-axis which will mark where on the line the amino acid is positioned.

We begin by first running the algorithm on KRAS using the MDS method followed by the linear method. For a full list of all the possible parameters, simply type '?ClusterFind' after loading the **iPAC** library.

Code Example 5: Running the ClusterFind method with a MDS mapping

```
> #Extract the data from a CIF file and match it up with the canonical protein sequence.
> #Here we use the 3GFT structure from the PDB, which corresponds to the KRAS protein.
> CIF<-"https://files.rcsb.org/view/3GFT.cif"
> Fasta<-"https://www.uniprot.org/uniprot/P01116-2.fasta"
> KRAS.Positions<-get.Positions(CIF,Fasta, "A")
> #Load the mutational data for KRAS. Here the mutational data was obtained from the
> #COSMIC database (version 58).
> data(KRAS.Mutations)
> #Identify and report the clusters.
> ClusterFind(mutation.data=KRAS.Mutations,
+             position.data=KRAS.Positions$Positions,
+             create.map = "Y", Show.Graph = "Y",
+             Graph.Title = "MDS Mapping",
+             method = "MDS")

[1] "Running Remapped"
[1] "Running Full"
[1] "Running Culled"
$Remapped
     cluster_size start end number      p_value
```

```
V136            1   12  12     100 8.932390e-183
V136            2   12  13     131 3.908116e-165
V121           49   13  61      38 3.097954e-124
V124          134   13 146      49 1.017093e-122
V136          106   12 117     139 1.362730e-119
V121           57   61 117       6 3.016259e-106
V124           30  117 146      11 1.666182e-102
V124          135   12 146     149 3.813215e-90
V121           50   12  61     138 2.682445e-87
V93            10   13  22      32 3.434148e-73
V93            96   22 117       8 1.140497e-65
V73            11   13  23      33 1.429105e-53
V73            95   23 117       7 4.204875e-48
V93            11   12  22     132 3.398568e-39
V142            1   13  13      31 8.571798e-38
V73            12   12  23     133 1.963564e-23
V142          105   13 117      39 4.563510e-12
V124            1  146 146      10 3.897757e-08
V121           86   61 146      16 6.115605e-07

$OriginalCulled
     cluster_size start end number       p_value
V12             2   12  13     131 9.453887e-229
V12             1   12  12     100 7.630495e-183
V12            11   12  22     132 1.554973e-138
V12            12   12  23     133 3.526333e-135
V12            50   12  61     138 2.824800e-58
V13             1   13  13      31 8.857871e-38
V12           106   12 117     139 4.538089e-17
V12           135   12 146     149 3.853241e-13
V13            10   13  22      32 8.603544e-11
V13            11   13  23      33 2.553752e-10
V146            1  146 146      10 5.331155e-08

$Original
     cluster_size start end number       p_value
V12             2   12  13     131 1.979447e-235
V12             1   12  12     100 6.486735e-188
V12            11   12  22     132 3.220145e-145
V12            12   12  23     133 6.524053e-142
V12            50   12  61     138 4.338908e-65
V13             1   13  13      31 2.732914e-39
V12           106   12 117     139 2.341227e-23
V12           135   12 146     149 1.356584e-20
V13            10   13  22      32 4.487362e-12
V13            11   13  23      33 1.279256e-11
```
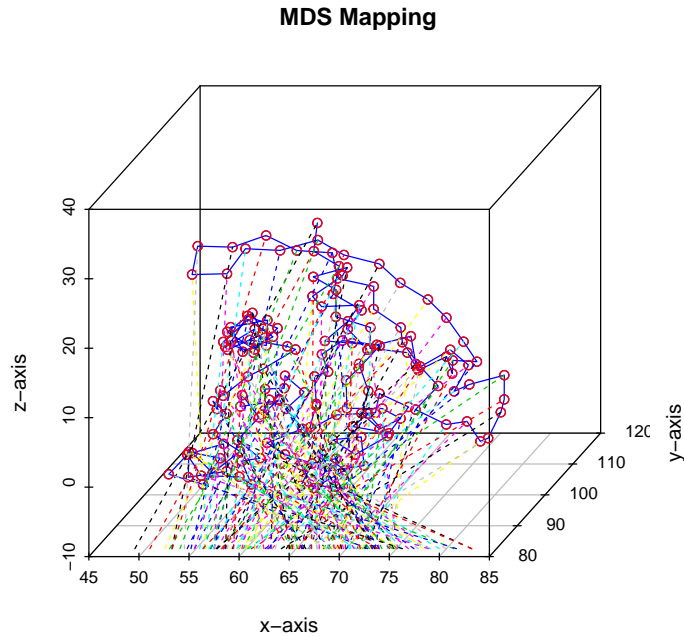
```
V146               1    146 146      10  1.918440e-08


$MissingPositions
     LHS RHS
[1,] 168 188
```

MDS Mapping

Code Example 6: Running the ClusterFind method with a Linear mapping

```
> #Extract the data from a CIF file and match it up with the canonical protein sequence.
> #Here we use the 3GFT structure from the PDB, which corresponds to the KRAS protein.
> CIF<-"https://files.rcsb.org/view/3GFT.cif"
> Fasta<-"https://www.uniprot.org/uniprot/P01116-2.fasta"
> KRAS.Positions<-get.Positions(CIF,Fasta, "A")
> #Load the mutational data for KRAS. Here the mutational data was obtained from the
> #COSMIC database (version 58).
> data(KRAS.Mutations)
> #Identify and report the clusters.
> ClusterFind(mutation.data=KRAS.Mutations,
+             position.data=KRAS.Positions$Positions,
+             create.map = "Y", Show.Graph = "Y",
+             Graph.Title = "Linear Mapping",
+             method = "Linear")
```

9

```
[1] "Running Remapped"
[1] "Running Full"
[1] "Running Culled"
$Remapped
     cluster_size start end number        p_value
V90             1    12  12    100 8.862875e-183
V90             2    12  13    131 2.234056e-175
V90           135    12 146    149 2.965106e-158
V90            50    12  61    138 1.092881e-149
V90            11    12  22    132  9.127863e-90
V66            57    61 117      6  3.393497e-89
V90            12    12  23    133  3.393497e-89
V66            30   117 146     11  2.804316e-86
V66           105    13 117     39  5.838579e-79
V66            96    22 117      8  8.125462e-61
V66            95    23 117      7  2.427091e-60
V66           106    12 117    139  2.047918e-48
V95             1    13  13     31  8.857871e-38
V95           134    13 146     49  1.438417e-26
V95            49    13  61     38  9.252899e-22
V100            1   146 146     10  3.897757e-08
V100           86    61 146     16  1.607373e-05
V95            11    13  23     33  8.412719e-04
V95            10    13  22     32  8.663956e-04


$OriginalCulled
     cluster_size start end number        p_value
V12             2    12  13    131 9.453887e-229
V12             1    12  12    100 7.630495e-183
V12            11    12  22    132 1.554973e-138
V12            12    12  23    133 3.526333e-135
V12            50    12  61    138  2.824800e-58
V13             1    13  13     31  8.857871e-38
V12           106    12 117    139  4.538089e-17
V12           135    12 146    149  3.853241e-13
V13            10    13  22     32  8.603544e-11
V13            11    13  23     33  2.553752e-10
V146            1   146 146     10  5.331155e-08


$Original
     cluster_size start end number        p_value
V12             2    12  13    131 1.979447e-235
V12             1    12  12    100 6.486735e-188
V12            11    12  22    132 3.220145e-145
V12            12    12  23    133 6.524053e-142
V12            50    12  61    138  4.338908e-65
```

```
V13                  1     13   13      31   2.732914e-39
V12                106     12  117     139   2.341227e-23
V12                135     12  146     149   1.356584e-20
V13                 10     13   22      32   4.487362e-12
V13                 11     13   23      33   1.279256e-11
V146                 1    146  146      10   1.918440e-08


$MissingPositions
     LHS RHS
[1,] 168 188
```
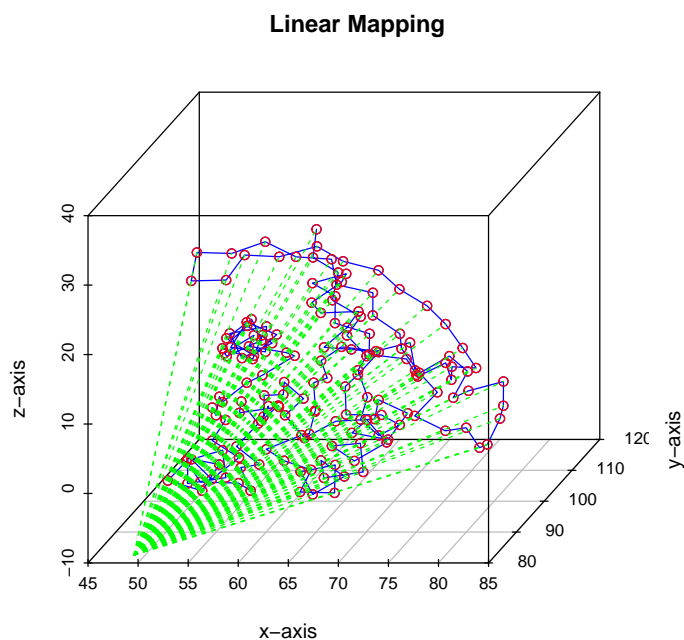
Linear Mapping

As can be seen from *Code Examples 5* and *6* above, the *ClusterFind* method returns a list of four elements. The first element, $MissingPositions, displays all the clusters found while taking into account the 3D structure of the protein by utilizing either the linear or MDS methodology. The next element, $OriginalCulled, displays all the clusters found using the original NMC algorithm after removing all the amino acids for which we do not have (x, y, z) positions. The $Original element displays all the clusters found using the NMC algorithm without removing the amino acids for which we do not have 3D positional information.

If the user wants to compare the results generated when taking protein structure into account versus those when protein structure is ignored, it is recommended that the user compare the matrices in $Remapped versus $Original-Culled. In this way, the user is considering the differences that arise strictly

11

from the protein structure as the amino acids with missing 3D positions have been removed prior to the analysis.

Finally, the $MissingPositions element displays a matrix of all the amino acids for which we had mutational data but for which we did not have positional data. For instance, in *Code Example 5*, the mutation data matrix had 188 columns while we were able to extract positional information only for amino acids 1-167. Furthermore, amino acid 61 was excluded from the final position matrix when the *get.AlignedPositions* function was run as the amino acid listed in the CIF file did not match the canonical sequence in the FASTA file. As such, the $MissingPositions element has a matrix of 2 rows as follows:

```
$MissingPositions
     LHS RHS
[1,]  61  61
[2,] 168 188
```

References

H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000. ISSN 13624962. doi: 10.1093/nar/28.1.235. URL www.pdb.org.

Ingwer Borg and Patrick J. F Groenen. *Modern multidimensional scaling : theory and applications*. Springer, New York, 1997. ISBN 0387948457 9780387948454.

Carlo M Croce. Oncogenes and cancer. *The New England Journal of Medicine*, 358(5):502–511, January 2008. ISSN 1533-4406. doi: 10.1056/NEJMra072367. URL http://www.ncbi.nlm.nih.gov/pubmed/18234754. PMID: 18234754.

S A Forbes, G Bhamra, S Bamford, E Dawson, C Kok, J Clements, A Menzies, J W Teague, P A Futreal, and M R Stratton. The catalogue of somatic mutations in cancer (COSMIC). *Current Protocols in Human Genetics / Editorial Board, Jonathan L. Haines ... [et Al.]*, Chapter 10:Unit 10.11, April 2008. ISSN 1934-8258. doi: 10.1002/0471142905.hg1011s57. URL http://www.ncbi.nlm.nih.gov/pubmed/18428421. PMID: 18428421.

Jingjing Ye, Adam Pavlicek, Elizabeth A Lunney, Paul A Rejto, and Chi-Hse Teng. Statistical method on nonrandom clustering with application to somatic mutations in cancer. 11(1):11, 2010. ISSN 1471-2105. doi: 10.1186/1471-2105-11-11. URL http://www.biomedcentral.com/1471-2105/11/11.