GraphPAC: Graph Theoretical Identification of Mutated Amino Acid Clusters in Proteins

Gregory Ryslik
Yale University
gregory.ryslik@yale.edu

Hongyu Zhao
Yale University
hongyu.zhao@yale.edu

May 3, 2016

Abstract

The **GraphPAC** package is a novel tool that identifies clusters of mutated amino acids in proteins by using graph theory to take into account protein tertiary structure. Specifically, the protein is mapped onto a one dimensional space by solving the Traveling Salesman Problem (TSP) heuristically via the **TSP** package (**?**). Once a hueristic solution to the TSP has been found, the protein is reorganized to a one-dimensional space by walking the path from the first amino acid to the last. The *Nonrandom Mutation Clustering* (NMC) (**?**) algorithm is then run on the reordered protein to identify if any pairwise mutations are closer together than expected by chance alone. **GraphPAC** is designed to be a companion package to **iPAC** (**?**) and provides the researcher with a different toolset to identify mutational clusters. By using a graph theoretical approach to map the protein to a one dimesional spacing, mutational clusters that are otherwise missed by the *NMC* and **iPAC** algorithms are found.

1 Introduction

Due to recent pharmacological advances in treating tumorogenic driver mutations (**?**), several methods have been developed to identify amino acid mutational clusters. One of the most recent methods, *NMC* considered all pairwise mutations and identified those that are closer than expected by chance alone under the assumption that each amino acid has an equal probability of mutation. *NMC*, which considers the protein linearly might potentially exclude amino acids that are close together in 3D space but far apart in 1D space. To address this issue, the **iPAC** methodology (**?**) reorganized the protein via MultiDimensional Scaling (MDS) (**?**). This package is designed to overcome the reliance on MDS and provides the researcher a different toolset for identifying mutational clusters.

Under a MDS approach, ever pairwise distance between amino acids is considered when the protein is mapped to a one dimensional space. Thus, amino acids that are very far apart from each other in 3D space still influence each other's final position in 1D space. The graph theoretical approach does not suffer from this limitation and would be more effective in reorganizing proteins that have several domains which are connected by domain linkers (see Figure **??**). By solving the TSP, we attempt to find the shortest path between all the amino acids. Amino acids that are in the same region of space (such as in a specific domain) will likely be close to each other in the path, while amino acids that are far apart in space (seperated by one or more domain linkers) will be far apart in the final path.

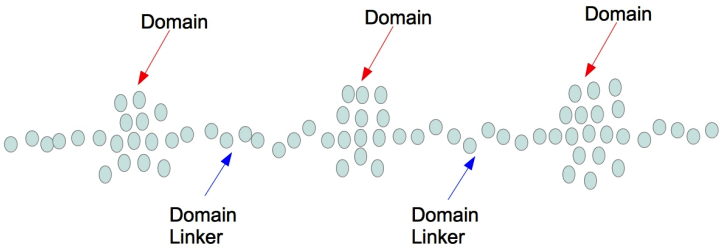


Figure 1: Possible Protein Arrangement of domain linkers and domains. The amino acids on the very left should have no effect on the reordering position of the amino acids on the right.

In order to run the clustering methodology we will describe below, 3 types of data are required. First, you need the amino acid sequence of the protein. Second, you need the protein tertiary structure and third you need the somatic mutational data. The amino acid sequence is obtained from the Sanger Institute and the protein tertiary structure is obtained from the PDB database.

An alignment (or other reconciliation) must be done in order to match the structural data with the amino acid sequence. Once that's done, the structural data is then matched with the mutational data which is obtained from the COSMIC database. The raw mutational data is available from the COSMIC website as a SQL database. Additional prior work is necessary to set up a local copy of the database and create the appropriate queries required to extract the mutational data. However, the end result is simply a $n \times m$ matrix where there are n samples for a protein which has a total of m amino acids. A "1" in the (i, j) position signifies that sample i had a mutation in amino acid j. If you have your own mutational data, you do not need to acess the COSMIC database and can simply create the mutational matrix described. Please ensure that your mutational matrix has the default R column headings of "V1,V2...Vm" where m is the number of the last amino acid in the protein.

We provide sample mutational data for the PIK3Cα and KRAS proteins. We also provide a brief description of how to obtain the amino acid sequence and the tertiary structure data in Code Example 1. For a full description of how to extract the correct mutational and positional data (via such functions as *get.Positions()* and *get.AlignedPositions()*), along with a description of the NMC algorithm please refer to the documentation provided in the **iPAC** package.

For the rest of this vignette, we will assume the user is familiar with these functions.

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

2 Finding Clusters in 3D Space via Graph Theory

Once the appropriate positional and mutational data has been loaded, the *GraphClust* function is run to identify the mutational clusters. Specifically, *GraphClust* will first reorder the protein by solving the TSP using one of the four insertion methods available in the **TSP** package (nearest, farthest, cheapest and arbitrary instertion). Once the protein is reordered, the mutational clusters are found and reported back to the user. An example of the code and ouput is shown in *Example 1* below.

Code Example 1: Running the GraphClust using the cheapest insertion method

```
> library(GraphPAC)
> #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<-"http://www.pdb.org/pdb/files/3GFT.cif"
> Fasta<-"http://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.
> my.graph.clusters <- GraphClust(KRAS.Mutations,KRAS.Positions$Positions,
+                                 insertion.type = "cheapest_insertion",
+                                 alpha = 0.05, MultComp = "Bonferroni")

Calculating Remapped Clusters.Calculating Culled Clusters.Calculating Full Clusters.

> my.graph.clusters
```

```
$Remapped
      cluster_size start end number       p_value
 [1,]           49    13  61     38 5.130714e-241
 [2,]            2    12  13    131 8.946018e-229
 [3,]            1    12  12    100 8.932390e-183
 [4,]           50    12  61    138 5.485758e-164
 [5,]           39    23  61      6 1.006405e-105
 [6,]           40    22  61      7 7.408106e-105
 [7,]           12    12  23    133  1.307698e-99
 [8,]           11    12  22    132  1.353429e-98
 [9,]            1    13  13     31  8.857871e-38
[10,]           57    61 117      6  4.352226e-31
[11,]          106    12 117    139  1.248463e-26
[12,]           86    61 146     16  1.214868e-21
[13,]          135    12 146    149  2.871632e-16
[14,]            1   146 146     10  5.331155e-08
[15,]           11    13  23     33  6.450857e-04
[16,]           10    13  22     32  7.246194e-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

$candidate.path
  [1]   1   2   3   4   5   6  53  52  51  50  49  48  47  46  45  44  43  42
 [19]  41  40  39  54  55  56  57   7   8   9  10  60  58  59  62  64  63  61
```

```
 [37]  12  13  18  19  20  29  33  35  38  37  36  34  32  31  30  28  27  26
 [55]  25  24  23  22  21  17  16  15  14  11  89  86  87  88  92  93  97  98
 [73]  99 103 104 106 105 102 101 100  96  95  94  91  90 125 124 123 115 114
 [91] 113 112 111 110 109 108 107 137 136 135 131 130 129 126 127 128 132 133
[109] 134 138 139 140 141 142 143 116 117 118 119 120 121 122  85  84  83  82
[127]  81  80  79  78  77  75  74  73  69  68  67  65  66  70  71  72  76 160
[145] 159 158 154 153 152 149 148 147 146 145 144 150 151 155 156 157 161 162
[163] 163 164 165 166 167

$path.distance
[1] 761.6425

$linear.path.distance
[1] 629.1051

$protein.graph
IGRAPH DN-- 167 166 --
+ attr: name (v/n), label (v/n)
+ edges (vertex names):
 [1]  1->  2  2->  3  3->  4  4->  5  5->  6  6-> 53 53-> 52 52-> 51 51-> 50
[10] 50-> 49 49-> 48 48-> 47 47-> 46 46-> 45 45-> 44 44-> 43 43-> 42 42-> 41
[19] 41-> 40 40-> 39 39-> 54 54-> 55 55-> 56 56-> 57 57->  7  7->  8  8->  9
[28]  9-> 10 10-> 60 60-> 58 58-> 59 59-> 62 62-> 64 64-> 63 63-> 61 61-> 12
[37] 12-> 13 13-> 18 18-> 19 19-> 20 20-> 29 29-> 33 33-> 35 35-> 38 38-> 37
[46] 37-> 36 36-> 34 34-> 32 32-> 31 31-> 30 30-> 28 28-> 27 27-> 26 26-> 25
[55] 25-> 24 24-> 23 23-> 22 22-> 21 21-> 17 17-> 16 16-> 15 15-> 14 14-> 11
[64] 11-> 89 89-> 86 86-> 87 87-> 88 88-> 92 92-> 93 93-> 97 97-> 98 98-> 99
+ ... omitted several edges

$missing.positions
     LHS RHS
[1,] 168 188
```

As we can see, the first 3 elements returned, *Remapped, OriginalCulled, and Original* are similar to those returned by **iPAC**. The *Remapped* element returns the clusters after the protein is reordered using the graph theory methodology described above. The *OriginalCulled* element returns the clusters found when the protein is considered linearly (with no reordering) but with all the amino acids that don't have positional data removed. The *Original* element shows the clustering results as found by the original NMC algorithm without taking any of the positional data into account.

The next 3 elements provide information regarding the path that was found by solving the TSP. The *candidate.path* element displays the actual path found. The *path.distance* element shows the total distance if one were to traverse the protein in the remapped order. The *linear.path.distance* element shows the total distance if one were to traverse the protein in the original linear form from the first to the last (Nth) amino acid: $1 \rightarrow 2 \rightarrow 3 \rightarrow ... \rightarrow N$ (the amino acids that had no positional data are skipped). The distance provided in *path.distance* and

linear.path.distance are measured in angstroms (Å).

The *protein.graph* element is a graph structure as defined in the the **igraph** package (**?**). Specifically, each amino acid is treated as a vertex. Then a directed edge from vertex i to vertex j is added if and only if the traveling salesman solution has the path going from i to j. This element is passed to the *plot.protein* function described in Section **??** below.

Finally, the *missing.positions* element provides a matrix that details which amino acids did not have positional data. These amino acids are removed when calculating clusters for the *Remapped* and *OriginalCulled* elements.

3 Plotting

Two types of plots have been implemented so far. Please ensure that you are using a terminal capable of graphical output before running these commands.

3.1 Jump Plots

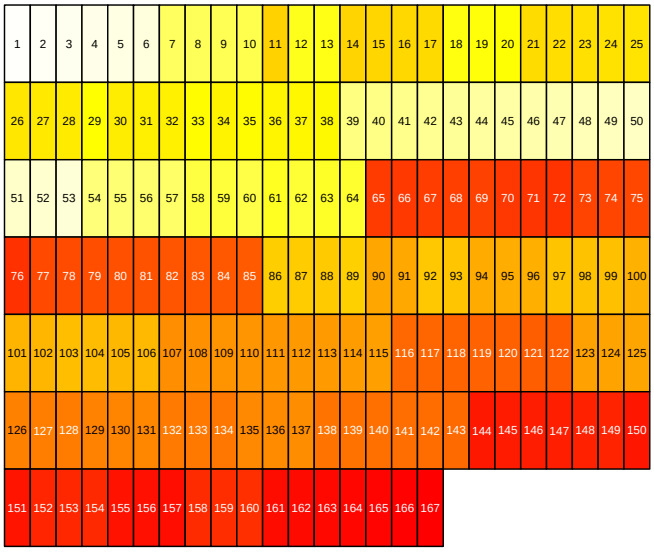
A jump plot displays the protein as matrix. The number of columns are specified by the user allowing control over how wide the resulting picture is. Once the color palette is selected, each element is then colored in with a different color which designates the position of the amino acid in the reordered protein.

Code Example 2: Making a Jump Plot

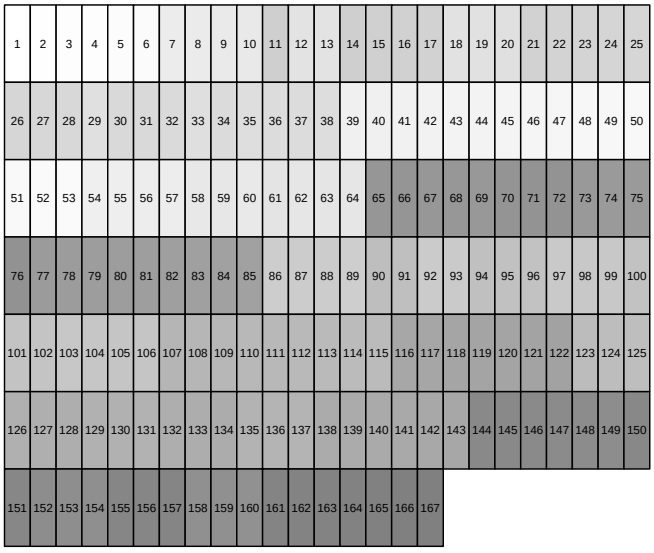
```
> #Using the heat color palette
> Plot.Protein.Linear(my.graph.clusters$candidate.path, 25, color.palette = "heat",
+                     title = "Protein Reordering - Heat Map")

> #Using the gray color palette
> Plot.Protein.Linear(my.graph.clusters$candidate.path, 25, color.palette = "gray",
+                     title = "Protein Reordering - Gray Color Scale")
```

Protein Reordering − Heat Map



Protein Reordering − Gray Color Scale



From the plot using the "heat" palette, we can see that the firt jump occurs from amino acid 6 to 7 since the color becomes much closer to red. Another large jump occurs between amino acids 64 to 67. Since the "heat" color palette ordering goes from white to red, amino acids that are reordered to the end of the protein will have a much redder color than those in the beginning. Similarly, amino acids reordered to the beginning of the protein will be almost completley white. Please run "?Plot.Protein.Linear" for a full description of the graphical parameters available for this function.

3.2 Interactive Circle Jump Plots

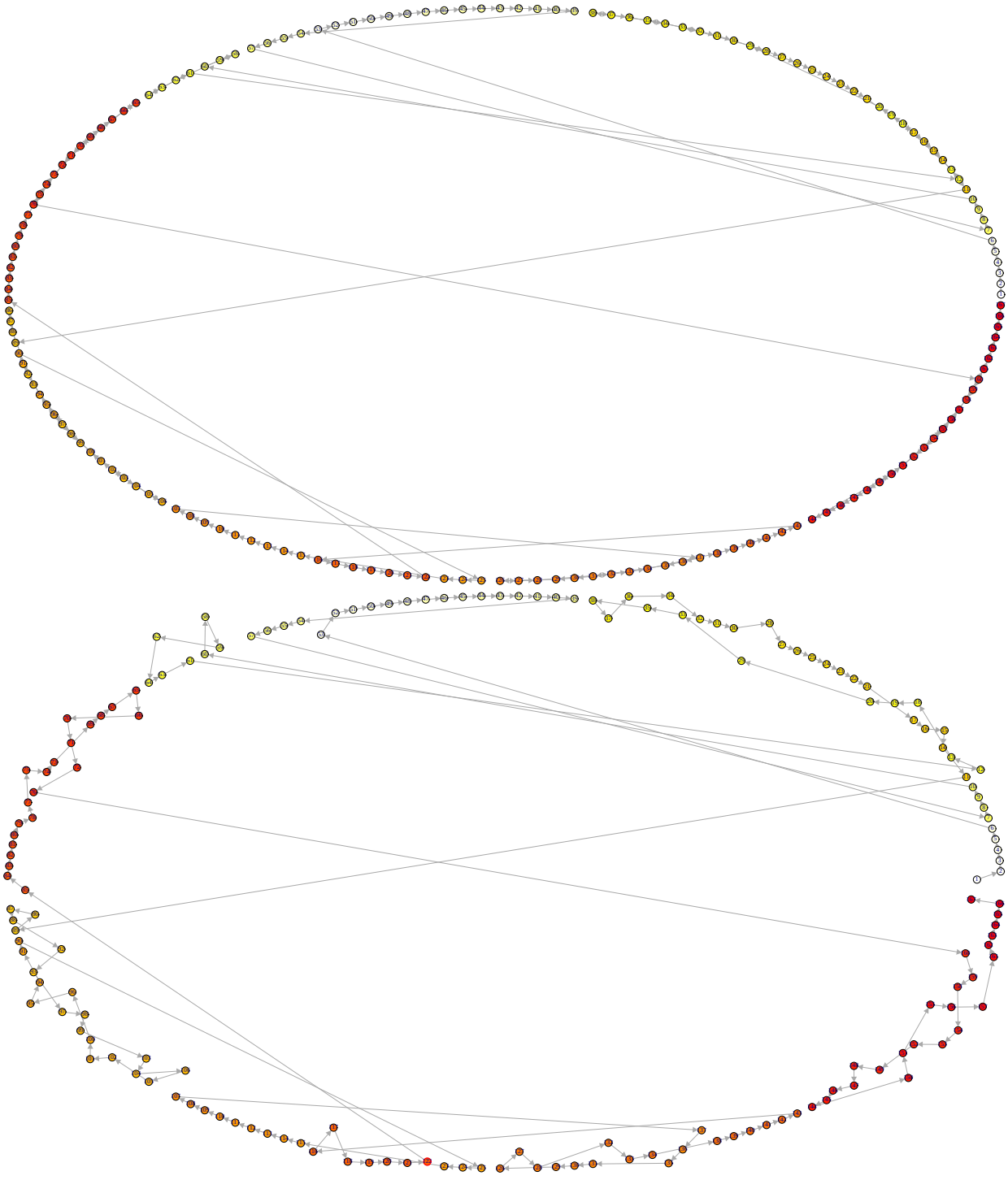
In addition to the static plot described in Section **??**, a circular jump plot allows for you to interactively see the graph. The plot use a TCL/TK window to plot the protein in circular form. The color coding of each amino acid represents it's position in the reordered protein in the same way as for regular Jump Plots. However, one can click and drag any amino acid in the window to see exactly how the edges connect.

Below we provide pictures of the circle plot as created by the algorithm and then the circle plot after manually adjusting the position of some vertices. As there are many vertices, please zoom in on the pdf to see all the details. For more information, run "?Plot.Protein". Finally, this function is a wrapper to the *tkplot* function in the **igraph** package, please look there for full technical specifications and additional options.

Special thanks to Dr. Gábor Csárdi (creator of the **igraph** package) for his help.

Code Example 3: Making a Circle Jump Plot

```
> #Using the heat color palette
> Plot.Protein(my.graph.clusters$protein.graph, my.graph.clusters$candidate.path,
+              vertex.size=5, color.palette="heat")
```



4 Comparing Path Differences

In addition to the graphical options provided above, one might want to consider a numerical measure of the reordering of the amino acid when comparing the **iPAC** and **GraphPAC** methodologies. One possible measure could be Kendall's Tau (**?**) which is equivalent to the number of reorderings performed during a bubble sort. This can be easily done via the **RMallow** package (**?**).

```
> library(RMallow)
> graph.path <-my.graph.clusters$candidate.path
> #get.Remapped.Order is a function in the \iPAC package
> mds.path <- get.Remapped.Order(KRAS.Mutations,KRAS.Positions$Positions)
> path.matrix <- rbind (original.seq = sort(graph.path), graph.path, mds.path)
> AllSeqDists(path.matrix)
```

```
original.seq   graph.path     mds.path
           0         2991         6357
```

Observe that the "original.seq" value will always be 0 since the original protein is already in order.