simplifyEnrichment: an R/Bioconductor package for Clustering and Visualizing Functional Enrichment Results

Zuguang Gu[1,2], Daniel Hübschmann[1,3,4,5]

[1]Computational Oncology, Molecular Diagnostics Program, National Center for Tumor Diseases (NCT) and German Cancer Research Center (DKFZ), Im Neuenheimer Feld 280, 69120 Heidelberg, Germany
[2]DKFZ-HIPO (Heidelberg Center for Personalized Oncology), 69120 Heidelberg, Germany
[3]Heidelberg Institute of Stem cell Technology and Experimental Medicine (HI-STEM), Im Neuenheimer Feld 280, 69120 Heidelberg, Germany
[4]German Cancer Consortium (DKTK), Im Neuenheimer Feld 280, 69120 Heidelberg, Germany
[5]Department of Pediatric Immunology, Hematology and Oncology, University Hospital Heidelberg, 69120 Heidelberg, Germany

Abstract

Motivation: Functional enrichment analysis or gene set enrichment analysis is a basic bioinformatics method that evaluates biological importance of a list of genes of interest. However, it may produce a long list of significant terms with highly redundant information that is difficult to summarize. Current tools to simplify enrichment results by clustering them into groups either still produce redundancy between clusters or do not retain consistent term similarities within clusters.
Results: we proposed a new method named *binary cut* for clustering similarity matrices of functional terms. Through comprehensive benchmarks on both simulated and real-world datasets, we demonstrated that binary cut can efficiently cluster functional terms into groups where terms showed more consistent similarities within groups and were more mutually exclusive between groups. We compared binary cut clustering on the similarity matrices from different similarity measurements and we found the semantic similarity worked well with binary cut while the similarity matrices based on gene overlap showed less consistent patterns and they were not recommended to work with binary cut. We implemented the binary cut algorithm into an R package *simplifyEnrichment* which additionally provides functionalities for visualizing, summarizing and comparing the clusterings.
Availability and implementation: The *simplifyEnrichment* package and the documentations are available at https://bioconductor.org/packages/simplifyEnrichment/. The reports for the analysis of all datasets benchmarked in the paper are available at https://simplifyenrichment.github.io/. The scripts that performed the analysis are available at https://github.com/jokergoo/simplifyEnrichment_manuscript.

1.Introduction

Functional enrichment analysis or gene set enrichment analysis is a type of method widely used to evaluate whether genes of interests, *e.g.,* differentially expressed genes, are over-represented or depleted in specific biological processes in forms of gene sets. It is performed by testing genes in the list against gene sets *e.g.*, through a hypergeometric test (Khatri and Drăghici, 2005), by analyzing whether the gene sets are enriched on the top of a

ranked gene list (Subramanian *et al.*, 2005), or by evaluating the gene expression profile in individual gene sets through univariate or multivariate approaches (Hung *et al.*, 2012). There are also methods to associate genomic regions of interests to gene sets, *e.g.*, to identify which biological processes the regulatory domains might affect (McLean *et al.*, 2010). Functional enrichment analysis depends on *a priori* biological knowledge represented as pre-defined gene sets, which are obtained from multiple sources. Among them, Gene Ontology (GO) (The Gene Ontology Consortium, 2019) is the most comprehensive source which organizes the biological terms in a hierarchical tree structure, in the form of a Directed Acyclic Graph (DAG). Other popular sources are, for example, KEGG pathways (Kanehisa *et al.*, 2017) and MsigDB catalogues (Liberzon *et al.*, 2015).

The enrichment results usually contain a long list of significantly enriched terms which have highly redundant information and are difficult to summarize, *e.g.*, an analysis on 671 EBI Expression Atlas differential expression datasets (Kapushesky *et al.*, 2010) using GO gene sets with Biological Process ontology showed there were 543 (80.9%) datasets having more than 250 significantly enriched GO terms under FDR < 0.05. The enrichment results can be simplified by clustering functional terms into groups where in the same group the terms provide similar information. The similarities between terms are important for the clustering and the calculation varies for different sources of gene sets. In the simplest and most general form, the similarity between two gene sets is based on gene overlap and is calculated as Jaccard coefficient, Dice coefficient or overlap coefficient (Merico *et al.*, 2010). Kappa coefficient is also suggested as more robust, which is used in the widely used *DAVID* tool (Huang *et al.*, 2007). For GO gene sets, there are more advanced methods which integrate the DAG structure to measure the semantic similarity between terms, *e.g.* the methods proposed in (Wang *et al.*, 2007; Schlicker *et al.*, 2006) and implemented in the R package *GOSemSim* (Yu *et al.*, 2010). The semantic similarity can also be calculated on other ontologies with DAG structure such as the Disease Ontology (Yu *et al.*, 2015; Schriml *et al.*, 2012).

Simplifying the enrichment results can be transformed into the task of clustering the similarity matrix of gene sets. Various clustering methods can be applied, for example, hierarchical clustering (Ayllon-Benitez *et al.*, 2020), affinity propagation clustering (Zhao and Wang, 2018) or density-based clustering (Huang *et al.*, 2007) that have already been implemented in current tools. The similarity matrix can also be treated as an adjacency matrix and be converted to a graph where the gene sets are the nodes and the similarity values are the weights of edges. Later, algorithms for detecting graph communities or modules are applied for identifying the clusters. A popular tool is *Enrichment Map* (Merico *et al.*, 2010).

Nevertheless, there are some common issues for clustering the similarity matrix of gene sets. In the following text, we take GO gene sets as an example and we call the root nodes (*i.e.*, the three top terms of Biological Process, Molecular Function and Cellular Component) as *the upstream* of the GO tree. In the GO tree, there exist large branches and small branches with different depths, which causes two major problems for clustering GO terms. First, the sizes of GO clusters vary a lot, which means, there are large clusters which normally correspond to large branches inheriting from very upstream of the GO tree, while at the same time, there are also many tiny clusters which correspond to the small branches

wherein terms concentrate in the very downstream of the GO tree. Due to the hierarchical relations of the tree, large clusters tend to have relatively lower mean similarities among terms while small clusters generally have relatively higher mean similarities. Methods such as *k*-means clustering sometimes cannot preserve the large clusters but further split them into smaller clusters, which might still produce redundant information between clusters. Moreover, they are not able to separate tiny clusters while they prefer to merge them into one single larger cluster. Second, large GO clusters contain large amounts of terms and it is possible that a small fraction of terms in cluster A also share similarities to terms in cluster B, in the other word, larger clusters are less mutually exclusive than smaller clusters, and this results in that some graph community methods merge cluster A and B into one larger cluster due to the connections shared by the two clusters. This increases the inconsistency among gene sets in that cluster. Thus, an effective method is needed to balance between these two scenarios, which are the generality and specificity of GO clustering.

In this paper, we proposed a new method named *binary cut* that clusters functional terms based on their similarity matrix. It recursively applies partition around medoids (PAM) with two groups on the similarity matrix and in each iteration step, a score is assigned to the submatrix to decide whether the corresponding terms should be further split into smaller clusters or not. We compared binary cut to a wide range of other clustering methods with 100 simulated GO lists and 648 GO enrichment results from real-world datasets, and we demonstrated that, with the similarities calculated by semantic measurements, binary cut was more efficient to cluster GO terms where inside clusters, terms shared more consistent similarity while they were more mutually exclusive between clusters. Binary cut can also be applied to general similarity matrices, *i.e.*, those based on gene overlap between gene sets and on any type of ontologies. However, the performance of binary cut varied, depending on the sources of gene sets. The similarity matrices based on gene overlap generally showed less consistent patterns as the semantic similarity matrices and were not suggested to work with binary cut.

We implemented binary cut in an R/Bioconductor package named *simplifyEnrichment*. After the gene sets have been clustered, *simplifyEnrichment* visualizes the summaries of gene sets clusters by word clouds, which helps users easily interpret the common biological functions shared in the clusters. Additionally, *simplifyEnrichment* provides a framework where user-defined clustering methods can be integrated and utilize the visualization functionality from *simplifyEnrichment*. *simplifyEnrichment* can compare multiple results from different clustering methods in a straightforward way.

2.Methods

2.1 Similarity measurement

If functional terms can be clustered into groups, the similarity matrices of corresponding gene sets are expected to represent a pattern where blocks can be observed in the diagonals if they are visualized as heatmaps. Thus, to efficiently cluster gene sets is the task of identifying *the diagonal blocks* in the similarity heatmap. The similarity between gene sets can be calculated as gene overlap by, *e.g.*, Jaccard coefficient, Dice coefficient or overlap

coefficient. The three coefficients have similar forms. It is also suggested to calculate the gene overlap by kappa coefficient which takes into account the possibility of gene occurrence in the two gene sets just due to chance (Huang *et al.*, 2007). The formal definition of the four gene overlap coefficients can be found in Supplementary file 1. If the ontologies have DAG structures, the topology information can be used to calculate the similarities, to generate the so-called *semantic similarity*, *e.g.*, through information content-based approaches which measure the information each GO term provides from its offspring terms. There exist a broad amount of algorithms for calculating semantic similarities, see (Mazandu *et al.*, 2017) for a comprehensive overview.

As we observed from the benchmark datasets, the semantic similarities between GO terms showed more clear diagonal block patterns than the similarities based on gene overlap. Additionally, the diagonal block pattern of semantic matrix was common even for randomly sampled GO terms (Supplementary file 2). Thus, we mainly considered clustering GO terms with semantic similarities in the paper. *simplifyEnrichment* uses the semantic measurements implemented in the R package *GOSemSim* (Yu et al., 2010) and takes the relevance method (Schlicker *et al.*, 2006) as default (see comparisons of different semantic measurements in Supplementary file 3). Nevertheless, a similarity matrix by any other method and on any other ontology can be provided for *simplifyEnrichment* analysis.

2.2 Clustering Process

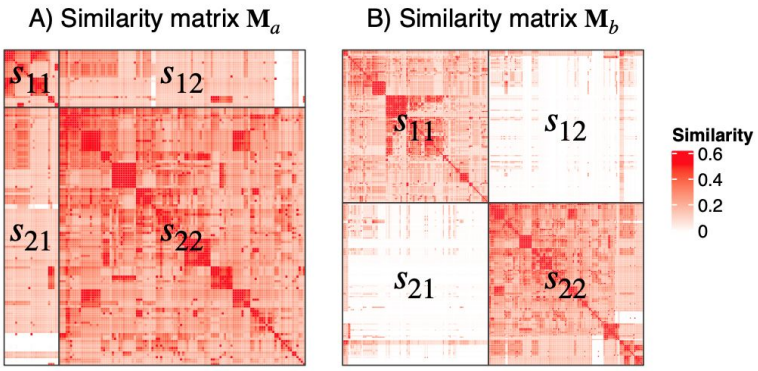


Figure 1. Two examples of GO similarity matrix. Both matrices are split into two groups in the two dimensions. The values s_{11}, s_{12}, s_{21} and s_{22} are the medians in the corresponding submatrices. The values on diagonals are excluded when calculating s_{11} and s_{22}.

Denote the similarity matrices of two lists of GO terms as $\mathbf{M}_a$ and $\mathbf{M}_b$, as illustrated in Figure 1. Intuitively, GO terms in $\mathbf{M}_a$ have overall high pairwise similarities, thus they should be treated as one single cluster (Figure 1A), while terms in $\mathbf{M}_b$ still show a two-block pattern in the diagonal and it should be further split into two more sub-clusters (Figure 1B). Therefore, a metric can be designed to decide whether a group of terms should be split or not. Based on this idea, we proposed a new method named *binary cut* to cluster the similarity matrix of functional terms, executed in two phases :

Phase 1: Applying divisive clustering and generating a dendrogram. It includes two steps:

Step 1. Apply partitioning around medoids (PAM) with two-group classification on the similarity matrix denoted as $\mathbf{M}$ on both rows and columns, which partitions it into four submatrices, denoted as $\mathbf{M}_{11}$, $\mathbf{M}_{12}$, $\mathbf{M}_{21}$, and $\mathbf{M}_{22}$ where the indices represent the groups in the two matrix dimensions. Next calculate the following score s_{11}, s_{12}, s_{21} and s_{22} where s_{11} and s_{22} are the median values in $\mathbf{M}_{11}$ and $\mathbf{M}_{22}$ excluding the values in the diagonals. s_{12} and s_{21} are the median values in $\mathbf{M}_{12}$ and $\mathbf{M}_{21}$. s_{11} or s_{22} is assigned to 1 if $\mathbf{M}_{11}$ or $\mathbf{M}_{22}$ is a one-row matrix. A score denoted as s is assigned to the group of terms corresponding to $\mathbf{M}$ and is calculated as

$$s = \frac{s_{11} + s_{22}}{s_{11} + s_{12} + s_{21} + s_{22}}.$$

The value s approximately ranges between 0.5 and 1. Then, if the matrix shows overall similarities, s is close to 0.5 (Figure 1A) and it should not be split any more, while if the GO terms can still be split into more groups, s_{12} and s_{21} are close to 0, which results in s close to 1 (Figure 1B).

Step 2. Apply Step 1 on the two submatrices $\mathbf{M}_{11}$ and $\mathbf{M}_{22}$ respectively. The clusterings in Step 1 and 2 are executed recursively and the process is saved as a dendrogram where a score s is attached to every node in the dendrogram which corresponds to every submatrix in the iteration. The clustering stops until the numbers of terms in a group reach 1 which are treated as the leaves of the dendrogram.

Phase 2: Cutting the dendrogram and generating clusters. Since every node in the dendrogram has a score s attached in Phase 1, s is simply tested to a cutoff (0.85 as the default and it can be optimistically selected by *simplifyEnrichment*). If s is larger than the cutoff, the two branches from the node are split, or else all the terms under the node are taken as a single cluster.

Nodes with large numbers of terms tend to have relatively smaller s, thus, it is possible at a certain node, s does not exceed the cutoff but is very close to it, while its child nodes have s larger than the cutoff. In this case, we don't want to close the node so early and we still split this node into two subgroups so that its child nodes can be split furthermore. Thus, the rule in Phase 2 is modified as: If s score in a node does not exceed the cutoff but it is larger than 0.8*cutoff, the node is still split as long as at least one of its child nodes with s score exceeds the cutoff.

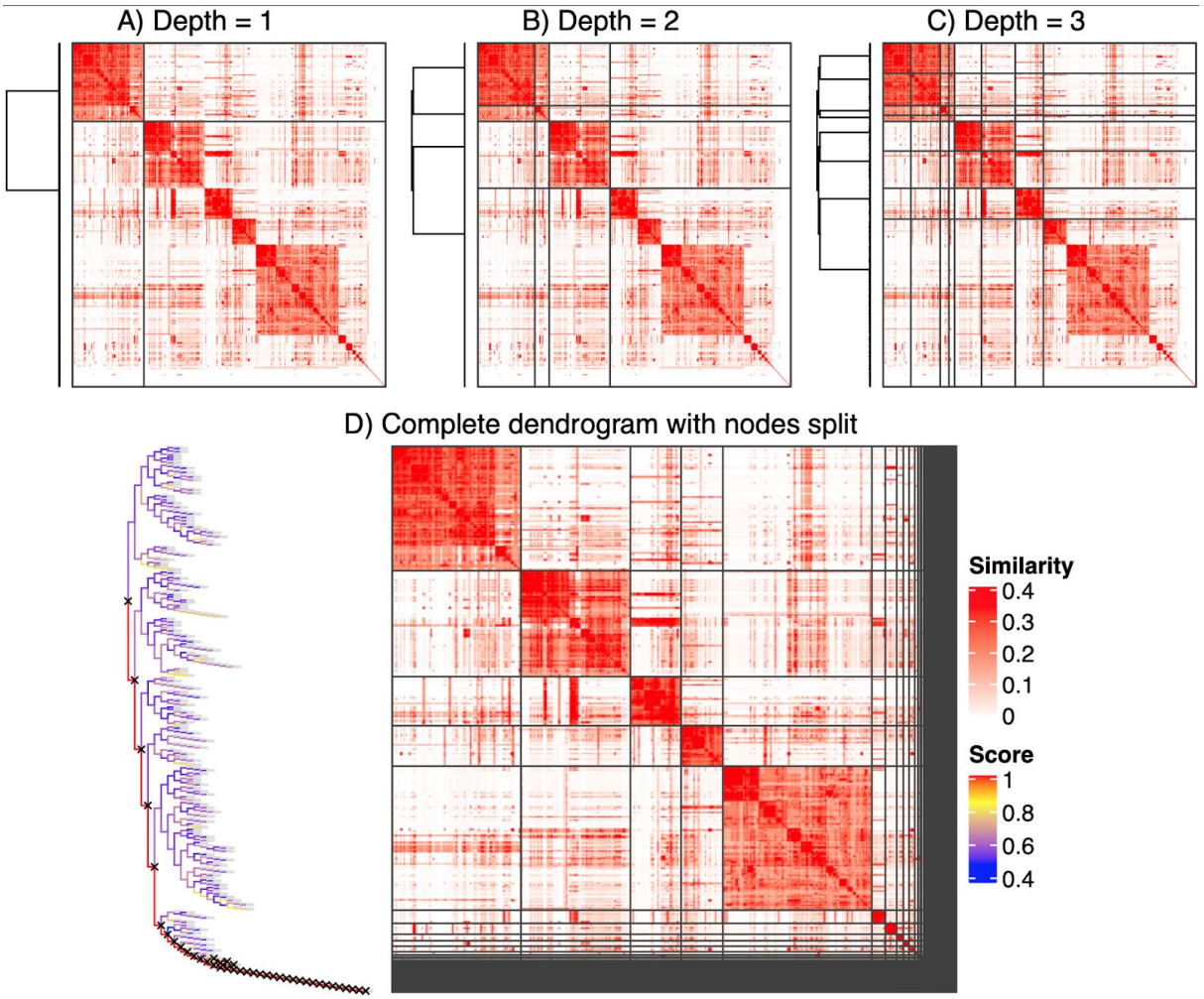


Figure 2. A demonstration of the binary cut clustering with 500 random GO terms. A-C) The clusterings in the first three iterations; D) The complete dendrogram from binary cut clustering. The colors of the dendrogram segments correspond to the s scores assigned to the nodes. The nodes that are split are marked with crosses.

In the iterations of binary cut clustering, there might exist more than two diagonal blocks in the submatrices. In the two-group partitioning on the submatrices, a proper method should be selected to ensure that each group contains complete diagonal blocks. As we have broadly tested on a large number of datasets, in most cases, PAM kept the completeness of diagonal blocks in the two-group classification.

An example of the process of binary cut clustering is demonstrated in Figure 2. Figure 2A-C illustrate the clusterings in the first three iterations. Figure 2D illustrates the final dendrogram where the nodes that are split are marked with crosses.

2.3 Software implementation

The binary cut algorithm is implemented in an R/Bioconductor package named *simplifyEnrichment*. The input of the package is a similarity matrix that is either user-provided or by the functions implemented in *simplifyEnrichment*, such as the function `GO_similarity()` for calculating GO semantic similarity or the function `term_similarity()` that measures general gene set similarity by Jaccard coefficient, Dice coefficient, overlap coefficient or kappa coefficient based on gene overlap.

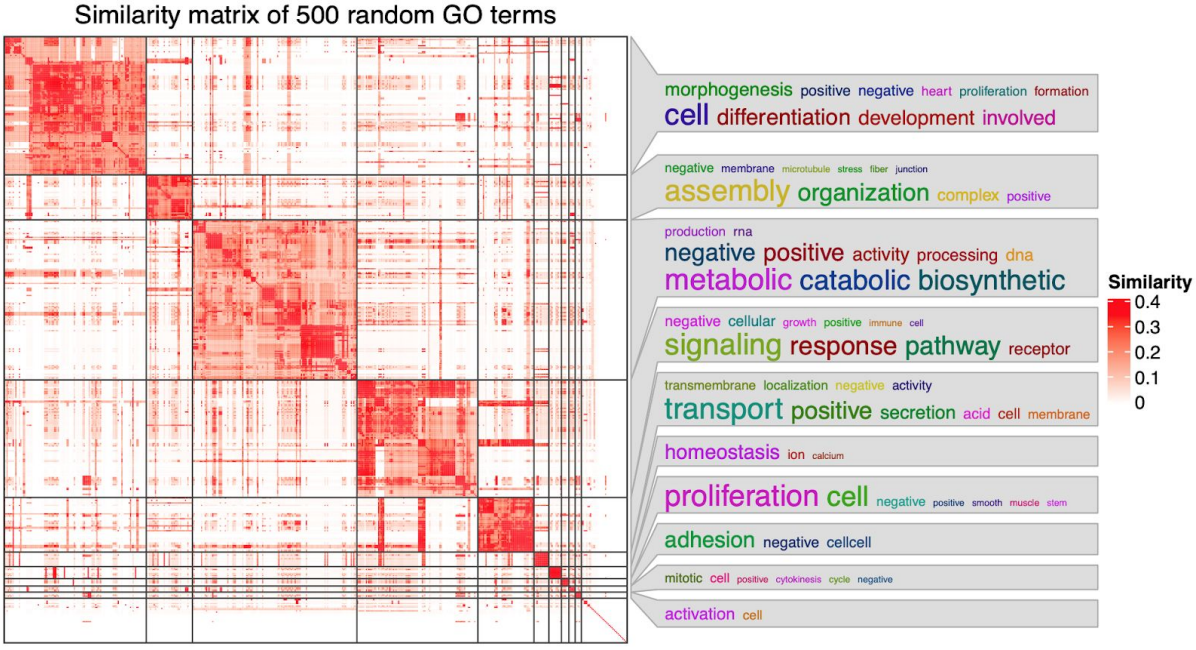


Figure 3. An example of similarity heatmap from 500 random GO terms that have been clustered and annotated with word clouds. The bottom right cluster with no word cloud annotation contains all other small clusters with numbers of terms less than 5.

The function `simplifyGO()` performs clustering for GO terms as well as visualizing the results. Once the GO terms have been clustered, the biological descriptions for the terms are automatically extracted. The summaries of the biological functions in clusters are visualized as word clouds and are attached to the similarity heatmap, which gives a direct illustration of the common biological functions involved in each cluster (Figure 3). The function `simplifyEnrichment()` performs clustering on similarity matrices from any type of ontology.

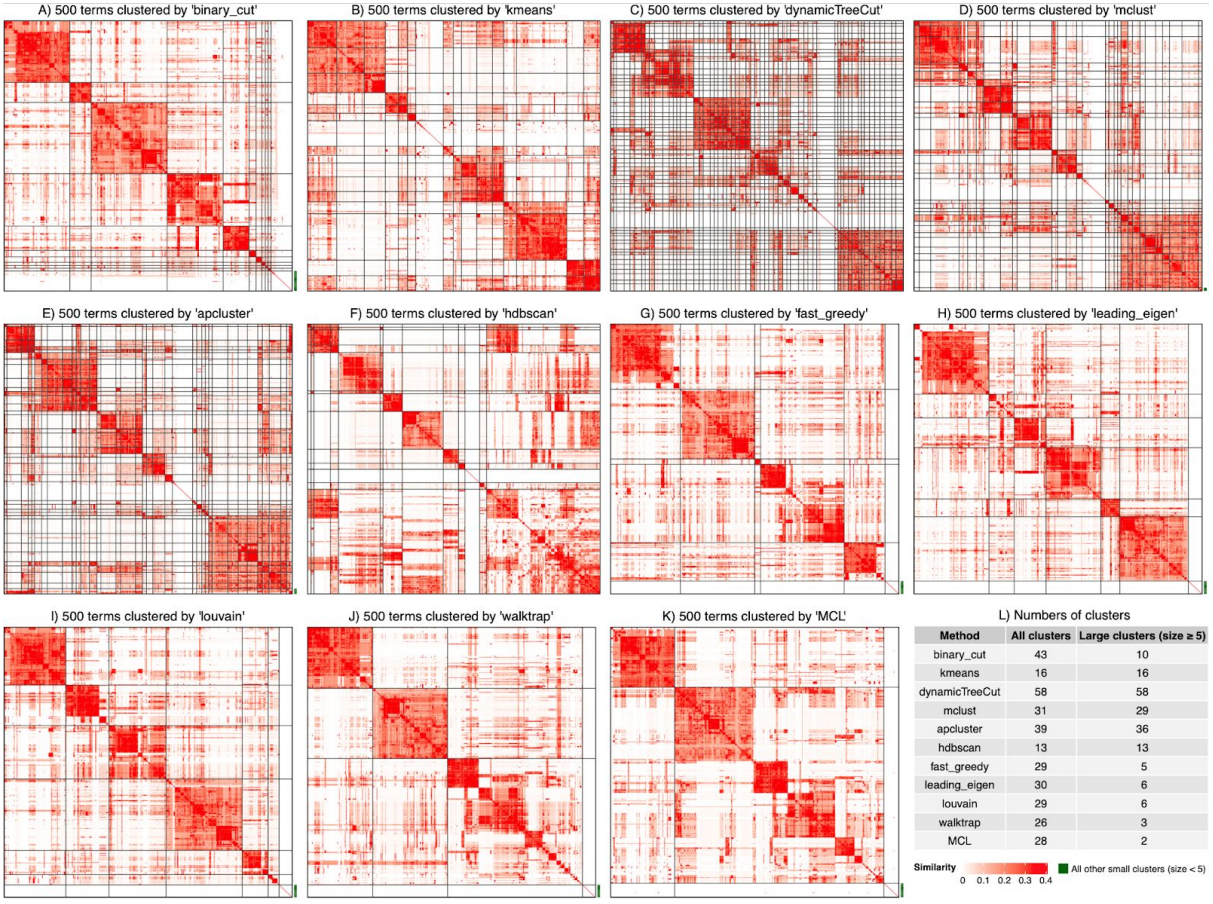
simplifyEnrichment also allows to integrate other clustering methods. New clustering functions can be added by the function `register_clustering_methods()` and the function `compare_clustering_methods()` applies multiple clustering methods on the same similarity matrix and compares between them via heatmaps (Figure 4).

3.Results

We compared the following 10 clustering methods to binary cut: *k*-means clustering (kmeans) where the optimized number of clusters was selected based on the distribution of within-cluster sum of squares (WSS), dynamicTreeCut (Langfelder *et al.*, 2008), mclust (Scrucca *et al.*, 2016), apcluster (Bodenhofer *et al.*, 2011), hdbscan (Hahsler *et al.*, 2019), the following graph community methods implemented in the R package *igraph* (Csardi and Nepusz, 2006): fast greedy, leading eigen, louvain and walktrap, as well as one additional community method MCL (Van Dongen, 2008) which was recently used in one implementation for clustering GO terms (Wang *et al.*, 2020). These methods are by default

supported in *simplifyEnrichment* for comparisons. The GO semantic similarities were calculated by the relevance method (Schlicker *et al.*, 2006) with the R package *GOSemSim*.

3.1 On random GO lists



Method	All clusters	Large clusters (size ≥ 5)
binary_cut	43	10
kmeans	16	16
dynamicTreeCut	58	58
mclust	31	29
apcluster	39	36
hdbscan	13	13
fast_greedy	29	5
leading_eigen	30	6
louvain	29	6
walktrap	26	3
MCL	28	2

Figure 4. Compare different clustering methods. A-K) The clusterings from 11 different methods. L) Numbers of all clusters and numbers of large clusters with size ≥ 5. For some methods, the small clusters (size < 5) were put into one single cluster on the bottom right of the heatmap and were marked by green lines. All the methods were applied to the same GO semantic similarity matrix from 500 random GO terms from Biological Process ontology.

500 random GO terms were uniformly sampled from Biology Process (BP) ontology. One example of the clusterings from 11 different methods is illustrated in Figure 4. It is straightforward to see dynamicTreeCut, mclust and apcluster generated huge numbers of small clusters (Figure 4C-E). Methods of kmeans and hdbscan generated intermediate numbers of clusters, but kmeans failed to preserve large clusters even if overall similarities in them were consistent (*e.g.*, the first three clusters in Figure 4B), while for hdbscan, the largest cluster did not show consistent similarities for all term pairs and it should be split furthermore (the bottom right cluster in Figure 4F). Methods of fast greedy, leading eigen, louvain, walktrap and MCL identified both small and large clusters, however, some large clusters should still be further split (Figure 4G-K). This scenario was even obvious for MCL (Figure 4K). As a comparison, binary cut generated clean clusters and it was able to identify large clusters and tiny clusters at the same time (Figure 4A).

The random GO lists of 500 BP terms were generated 100 times and the following metrics were used to benchmark the clustering methods quantitatively, compared in Figure 5.

Difference score: It measures the difference between the similarity values for the terms that belong to the same clusters and in different clusters. For a similarity matrix denoted as $\mathbf{M}$, and for term i and j where $i \neq j$, the similarity value $x_{i,j}$ is saved to the vector $\mathbf{x}_1$ when term i and j are in the same cluster. $x_{i,j}$ is saved to the vector $\mathbf{x}_2$ when term i and j are in different clusters. The difference score measures the distribution difference between $\mathbf{x}_1$ and $\mathbf{x}_2$, calculated as the Kolmogorov-Smirnov statistic between the two distributions of $\mathbf{x}_1$ and $\mathbf{x}_2$.

Number of clusters: For each clustering, there are two numbers: the total number of clusters and the number of clusters with size ≥ 5 (*i.e.*, the large clusters). The two values can be used to test whether the clustering methods can identify tiny clusters.

Block mean: It is calculated as the mean similarity values of the diagonal blocks in the similarity heatmap. Similar denotation as the difference score, the block mean is the mean value of $\mathbf{x}_1$.

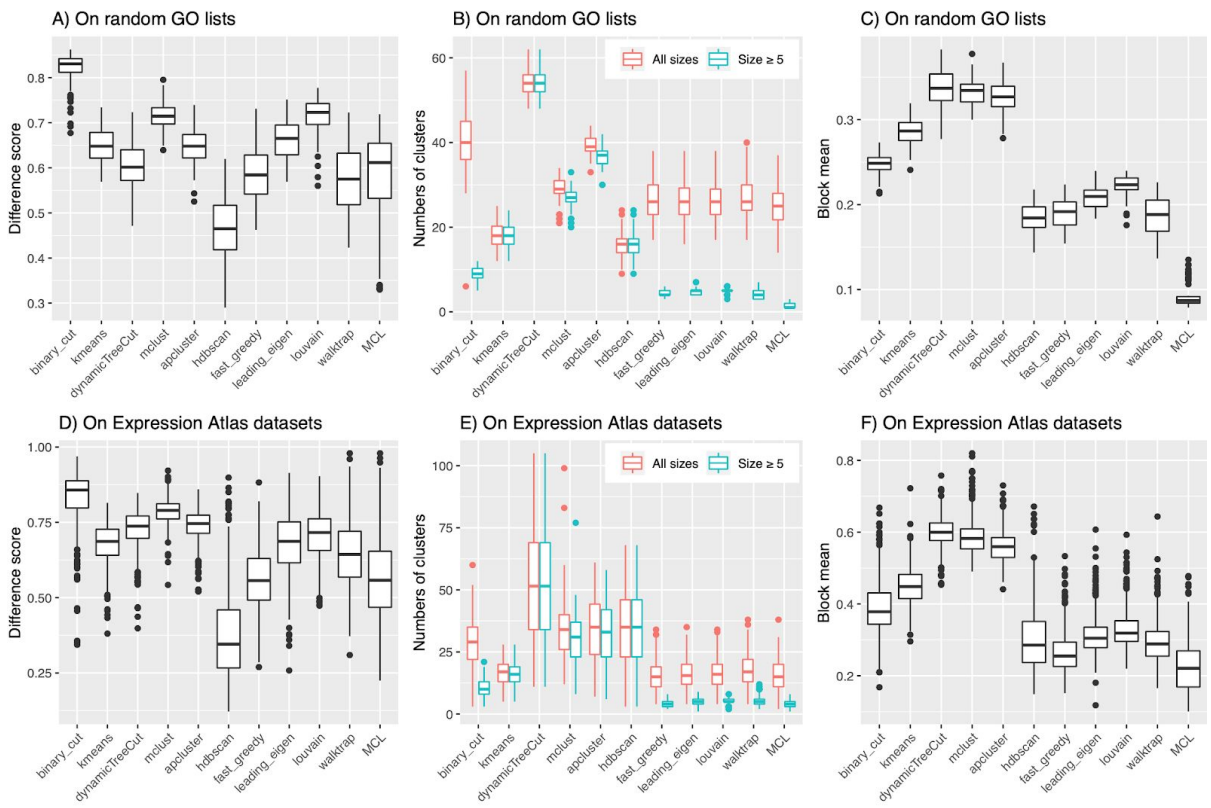


Figure 5. Benchmarks of different clustering methods. A-C) The difference scores, numbers of clusters and the block mean values for the clusterings identified by 11 methods. The analysis was applied on 100 random GO lists of 500 BP terms. D-F) Similar as Figure A-C, but on the functional enrichment results from Expression Atlas datasets.

In Figure 5A, binary cut had the highest difference score, which showed, in the clusterings by binary cut, the similarity values of the terms in the same clusters had the most distinct distribution to the rest of the similarity values that measured the term similarity in different

clusters. In Figure 5B, dynamicTreeCut and apcluster generated a huge number of clusters (examples are in Figure 4C and 4E). The two methods were too stringent for clustering GO terms and there still contained high levels of redundant information among clusters. As a comparison, graph community methods and binary cut generated moderate numbers of clusters, and especially, the numbers of "large clusters" (size ≥ 5) dropped dramatically compared to the numbers of total clusters, indicating these methods were able to identify tiny clusters. In Figure 5C, binary cut generated clusters with moderate block mean values. As a comparison, kmeans, dynamicTreeCut, mclust and apcluster generated high block mean values, which associated to that they were not able to preserve the large clusters with intermediate similarities. Graph community methods generated low block mean values, which was mainly because terms in some of the large clusters did not show similarities for all term pairs. The method hdbscan generated intermediate numbers of clusters and had intermediate block mean values, but it had the lowest difference score in the comparisons, which implied, the difference between the similarities within clusters and between clusters were small (an example is in Figure 4F). As a conclusion, the comparisons indicated binary cut kept both generality and specificity of GO clusters. The analysis reports for all 100 random GO lists can be found in Supplementary file 4.

3.2 On real-world datasets

We analyzed all datasets from Expression Atlas (Kapushesky *et al.*, 2010) that have the differential expression analysis results. We applied functional enrichment analysis with the R package *clusterProfiler* (Yu *et al.*, 2012) to the significantly expressed genes (FDR < 0.05) with the GO Biological Process ontology. We only took the datasets with the number of significant genes in [500, 3000] and the number of significant GO terms (FDR < 0.05) in [100, 1000]. Finally there were 468 GO lists. Figure 5D-F illustrated the results for the comparisons among the 11 clustering methods. The conclusion was very similar as the random GO datasets that binary cut outperformed other clustering methods. The analysis reports for 468 individual Expression Atlas datasets can be found in Supplementary file 4.

3.3 General similarity by gene overlap

There are still a lot of ontologies (*e.g.* MsigDB gene sets) that are only represented as a list of genes where the similarities between gene sets are merely measured by gene overlap. We compared the performance of binary cut on similarity matrices calculated by semantic measurement and four gene overlap measurements, *i.e.*, Jaccard coefficient, Dice coefficient, overlap coefficient and kappa coefficient. Similarly, we applied binary cut to the 100 random GO lists with 500 BP terms as well as 468 significant GO lists from Expression Atlas datasets.

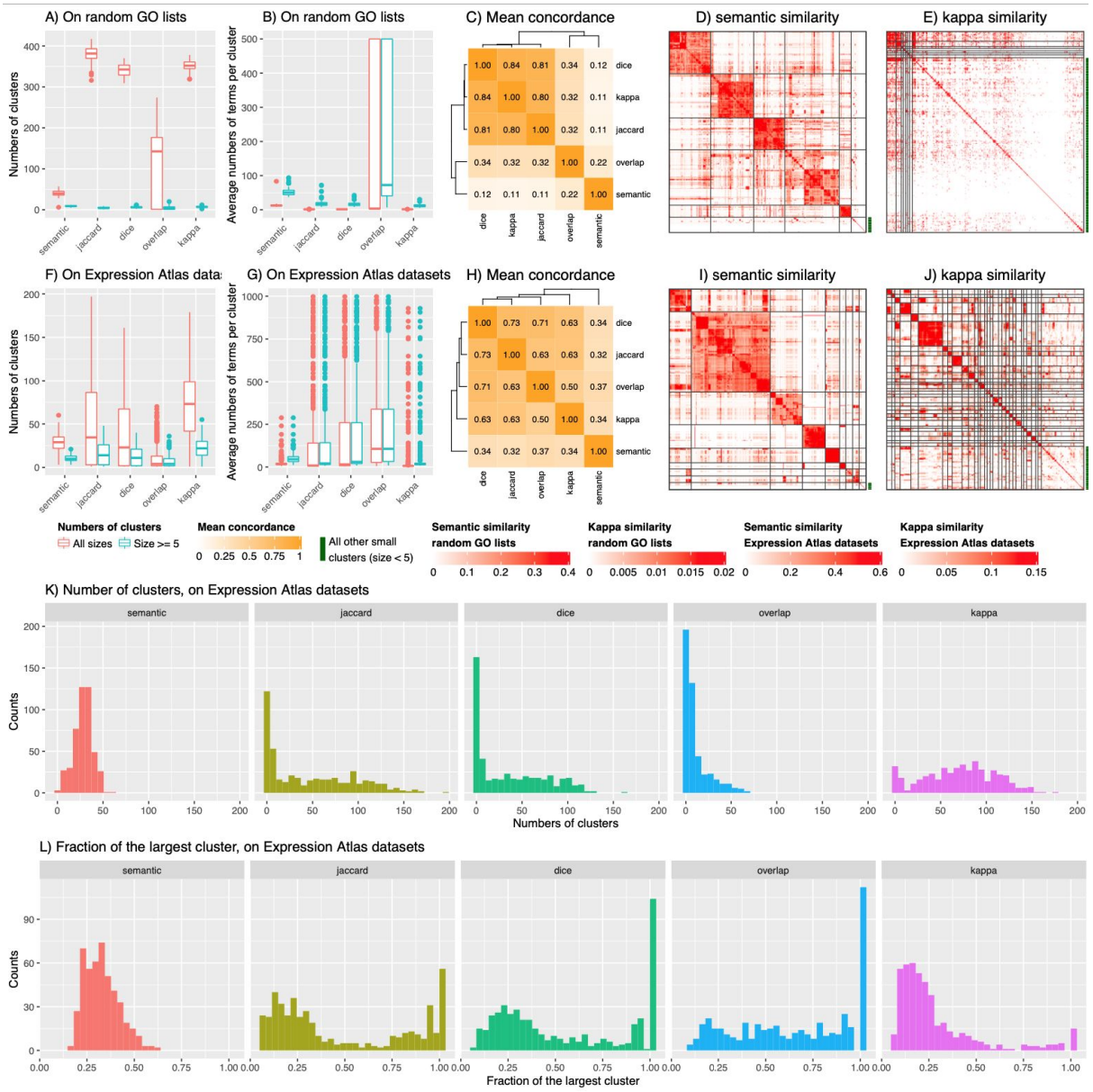


Figure 6. Compare clusterings on similarity matrices by different similarity measurements. A) Numbers of clusters. B) Average numbers of terms per cluster. C) Mean concordance of the clusterings. The calculation of concordance can be found in Supplementary file 10. The analysis was applied on 100 random GO lists with 500 BP terms. D-E) Examples of binary cut clustering on two similarity matrices from the same list of random GO terms by semantic and kappa measurements. F-J) Similar as Figure A-E, but on the functional enrichment results from Expression Atlas datasets. Figure I and J were based on a same dataset. K) Distributions of cluster sizes on similarity matrices from different similarity measurements. L) Distributions of the fraction of the largest cluster. Figure K and L were based on Expression Atlas datasets.

When GO terms were randomly generated, with similarities measured by Jaccard coefficient, Dice coefficient and kappa coefficient, 500 terms were split into large numbers of clusters (Figure 6A, red boxes), where on average, only less than two terms were in a single cluster (Figure 6B, red boxes). The numbers of clusters dropped dramatically if only counting clusters with size ≥ 5 (Figure 6A, blue boxes). For the clusters with size ≥ 5, the average numbers of GO terms per cluster were 20 under Jaccard coefficient, 17 under Dice coefficient and 12 under kappa coefficient, while 53 under semantic measurement (Figure 6B, blue boxes). And for the clusters with size < 5, Jaccard coefficient generated 375 tiny

clusters (covered 84.5% of all terms), Dice coefficient generated 335 tiny clusters (81.4% of all terms) and kappa coefficient generated 345 tiny clusters (84.9% of all terms), while as a comparison, semantic measurement only generated 31 tiny clusters (7.8% of all terms). This implied binary cut generated many tiny clusters on the similarity matrices of Jaccard coefficient, Dice coefficient and kappa coefficient and these three coefficients were not efficient for clustering. Examples of the clusterings under different similarity measurements can be found in Figure 6E. The overlap coefficient performed differently from the other three gene overlap measurements (32.3% agreement of the clusterings, Figure 6C). In 30% of all 100 random GO lists, the 500 GO terms could not even be separated and the numbers of clusters for them were only one (Supplementary file 5: Figure S5.2-S5.4). The individual heatmaps of overlap similarities showed that in most cases only one major cluster was generated with a marginal pattern where a small amount of terms showed high similarities to most of the other terms and no diagonal block pattern was observable (Supplementary file 5: Table S5.1). This is due to the form of overlap coefficient where the score is normalized by the size of the smaller gene set. For a term located at the downstream of the GO tree, it has the same overlap coefficients to all its ancestor terms because parent terms include all genes of their child terms. The heatmaps for individual datasets (Supplementary file 5: Table S5.1) showed that, for the randomly sampled GO lists, similarity values calculated by gene overlap were very weak and noisy, which resulted that in most cases, terms could not be clustered with other terms. As a comparison, the semantic similarity matrices generated reasonable numbers of clusters and had clear diagonal block patterns (Figure 6D).

Analysis on the Expression Atlas datasets with GO BP ontology also showed that binary cut on semantic similarity matrix generated very different clusterings from the four gene overlap similarity matrices (Figure 6F-J). The cluster sizes from the matrices measured by Jaccard coefficient, Dice coefficient and kappa coefficient had very broad ranges where many datasets had more than 100 clusters (on average 16.7% of all datasets, Figure 6K, an example is in Figure 6J). On the other hand, Jaccard coefficient, Dice coefficient and overlap coefficient also generated very few clusters (size < 5) for many datasets (on average 49.7% of all datasets, Figure 6K). As a comparison, semantic measurements generated numbers of clusters with moderate range (14 to 40 for the 10th to the 90th percentile). Moreover, it showed that in 30.6% datasets the largest cluster having more than 80% of all terms under Jaccard coefficient, in 33.8% datasets under Dice coefficient and in 39.7% datasets under overlap coefficient (Figure 6K-L). This implied these three coefficients cannot effectively separate terms in many datasets. For overlap coefficient, it showed almost no obvious marginal pattern as the random GO lists, because it is relatively difficult for a deeper GO term to be significant due to the small number of genes in the GO term (Supplementary file 6). Overall, semantic similarity matrices worked better than gene overlap similarities on the real-world datasets.

4. Discussion

Simplifying functional enrichment results is necessary for easy interpretation without missing any important biological term. The essential step to simplify enrichment results is to cluster the functional terms, later *post hoc* analysis can be applied to summarize the major biological functions in each cluster, such as extracting the most significant terms in each

cluster (Yu *et al.*, 2012) or looking for a minimal subset of terms where the genes cover genes from all terms in the cluster (Liao *et al.*, 2019). A proper clustering method should identify groups of terms that provide more distinct information between clusters and reduce a huge number of terms into an amount that is easier to read and analyze. In this work, we developed the *simplifyEnrichment* package, which proposed a new method of binary cut to efficiently cluster functional terms and to summarize the general functions in clusters by word cloud visualizations.

Binary cut worked well when the diagonal block patterns were observable in the similarity matrix of functional terms. In the benchmark, the similarity matrices of GO terms by semantic measurement showed clear diagonal block patterns, even for a list of random GO terms. In Supplementary 3, we compared the five semantic similarity measurements supported in the *GOSemSim* package and we found the methods of "Rel" (Schlicker *et al.*, 2006), "Resnik" (Resnik, 1999) and "Lin" (Lin, 1998) produced similar and clear diagonal block patterns, thus they are more suggested to work with binary cut clustering. These three semantic measurements are all information content (IC)-based methods with similar forms which rely on the IC of the most informative common ancestor of two terms in the GO tree. A GO term has higher IC if it has less offspring terms. Thus, in general, the deeper the ancestor of two terms is in the GO tree, the higher IC the ancestor will provide and the higher similarity the two terms will have. Uniformly sampling the GO terms selects more terms locating in the downstream of the GO tree due to its hierarchical structure, therefore, it is more probable for two terms to reach an ancestor node relatively deeper in the tree to contribute a higher IC value, which makes that even random GO lists had clear diagonal block patterns in the benchmark. An analysis of all GO BP terms showed that with semantic similarity, globally they had diagonal block patterns where the clusters correspond to several top functional categories (Supplementary file 7). This also ensures that a list of random GO terms also show the diagonal patterns that are uniformly sampled from a highly modularized global similarity matrix.

In each iteration step of the recursive splitting on the similarity matrix, the current submatrix is partitioned into two groups using PAM as default. In Supplementary file 8, we compared binary cut clustering on 100 random GO BP lists with the following three partitioning methods: PAM, *k*-means++ and hierarchical clustering with "ward.D2" method. *k*-means++ is an optimized *k*-means clustering by pre-selecting proper initial centroids (Vassilvitskii, 2007; Leisch, 2006). The results showed the three partitioning methods performed very similarly, *i.e.*, with similar difference scores, numbers of clusters and block mean values among 100 random GO lists. The final clusterings of GO terms are highly agreed (on average 89% agreement, Supplementary file 8: Figure S8.1) between the three partitioning methods. The three methods are all supported in *simplifyEnrichment* and they can be selected specifically for individual datasets.

In the binary cut clustering process, a score is calculated based on the two-group partitioning on the current submatrix and it is compared against a cutoff to decide whether the corresponding terms are treated as a single group or should be further split. In most cases, the default cutoff is robust and works well if the diagonal pattern in the similarity heatmap is observable. Nevertheless, there are still cases where manual adjustment on the cutoff needs

to be applied, such as a large cluster which is composed of several small and dense clusters. A typical example is the clustering on the semantic similarity matrices of Disease Ontology terms (Supplementary file 4). The cutoff can be fine-tuned according to what levels of similarities users want to keep. *simplifyEnrichment* provides a `select_cutoff()` function which tries a list of cutoffs and compares difference scores, numbers of clusters and block mean values for the clusterings to decide an optimized cutoff value. In Supplementary file 9, we demonstrated the use of `select_cutoff()` function and how it helps the selection for a proper cutoff.

We mainly benchmarked GO enrichment with BP ontology, which is the major ontology category (65.9%) in GO. In Supplementary file 4, we also applied binary cut to random GO lists from Cellular Component (CC) and Molecular Function (MF) ontologies. The results were very similar as BP terms where binary cut outperformed other clustering methods on the semantic similarity matrices. Besides GO gene sets, we also applied other ontologies to Express Atlas datastes, *i.e.*, gene sets from DO, KEGG, Reactome and MsigDB, only with gene overlap as the similarity measurements (Supplementary file 4). We only found the similarity matrices based on gene overlap coefficients of Reactome and MsigDB C4 gene sets can be used for binary cut clustering while the similarity matrices from other gene sets generally showed low consistency between terms and did not have clear diagonal block patterns, thus, they are not recommended to work with binary cut.

5. Conclusions

We described a new clustering algorithm, binary cut, for clustering similarity matrices of functional terms. Through comprehensive benchmarks on both simulated and real-world datasets, we demonstrated that binary cut can efficiently cluster functional terms where the terms showed more consistent similarities within clusters and were more mutually exclusive between clusters. We implemented the algorithm into an R package *simplifyEnrichment* which additionally provides functionalities for visualizing, summarizing and comparing the clusterings. We believe *simplifyEnrichment* will be a useful tool for researchers to fast explore their results and obtain key biological findings.

Bibliography

Ayllon-Benitez,A. *et al.* (2020) GSAn: an alternative to enrichment analysis for annotating gene sets. *NAR Genom. Bioinform.*, **2**.

Bodenhofer,U. *et al.* (2011) APCluster: an R package for affinity propagation clustering. *Bioinformatics*, **27**, 2463–2464.

Csardi,G. and Nepusz,T. (2006) The igraph software package for complex network research. *InterJournal*, Complex Systems, 1695.

Hahsler,M. *et al.* (2019) dbscan : Fast Density-Based Clustering with *R. J. Stat. Softw.*, **91**.

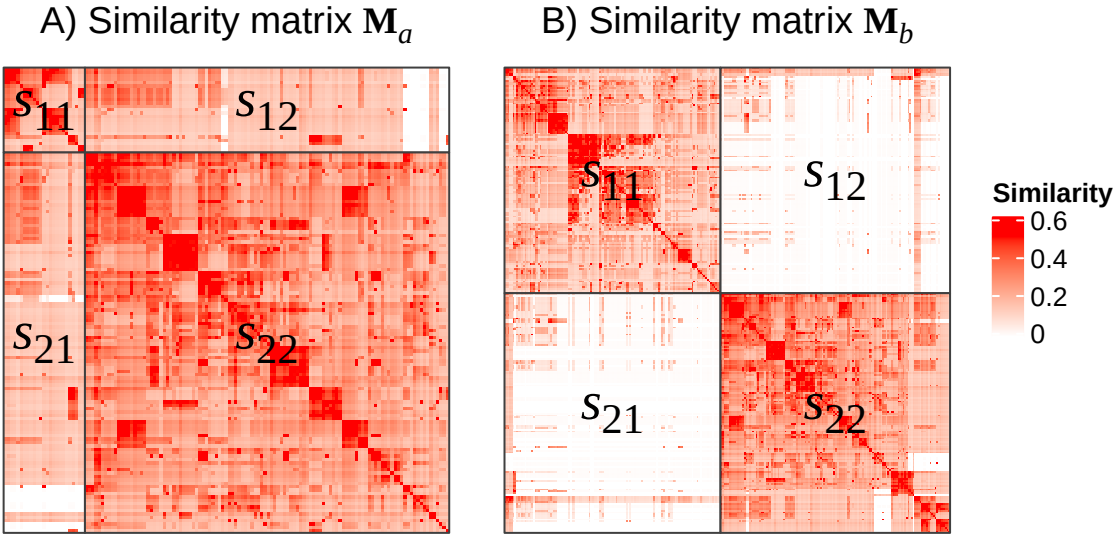
Huang,D.W. *et al.* (2007) The DAVID Gene Functional Classification Tool: a novel biological module-centric algorithm to functionally analyze large gene lists. *Genome Biol.*, **8**, R183.

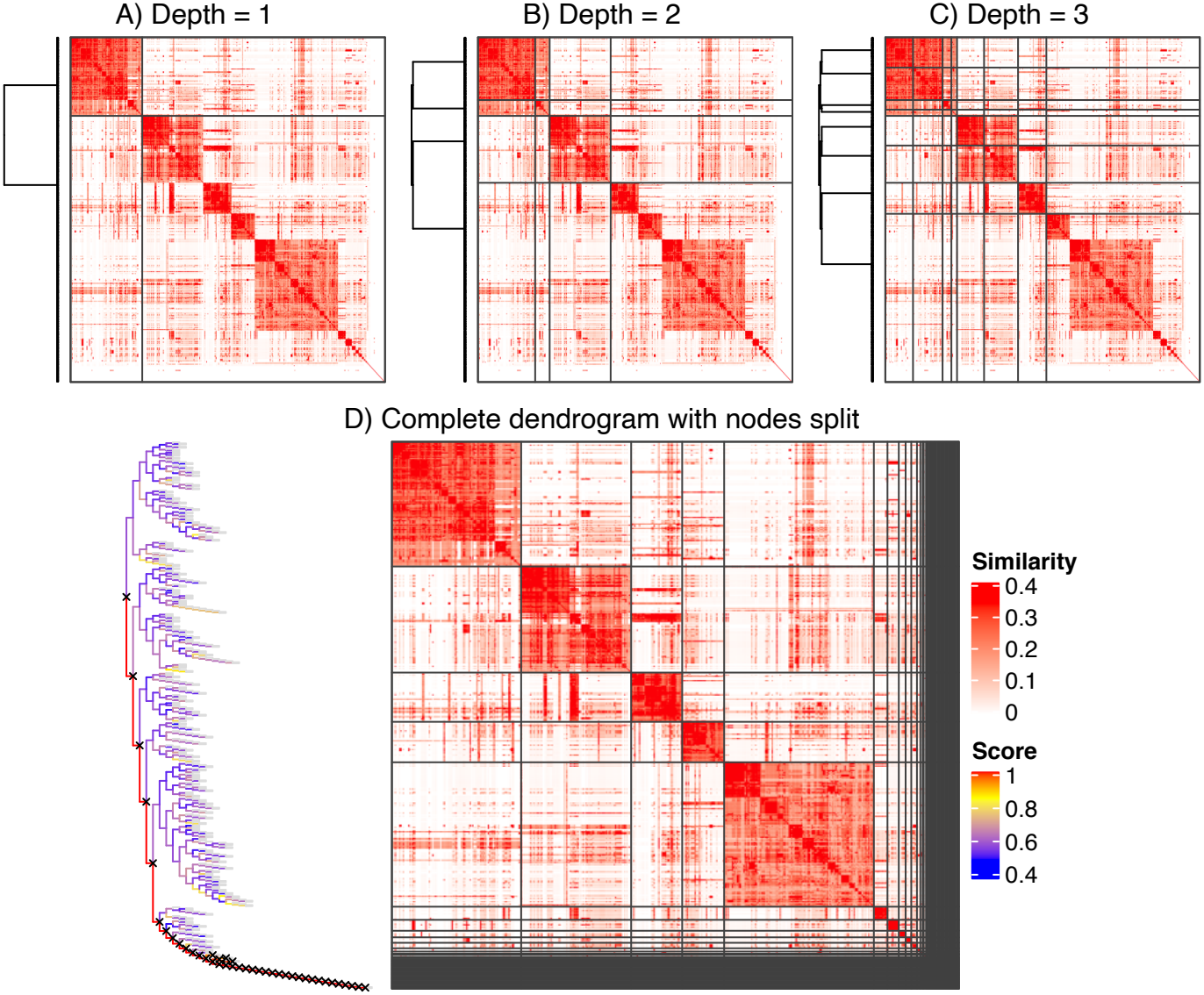
Hung,J.-H. *et al.* (2012) Gene set enrichment analysis: performance evaluation and usage guidelines. *Brief. Bioinformatics*, **13**, 281–291.

Kanehisa,M. *et al.* (2017) KEGG: new perspectives on genomes, pathways, diseases and

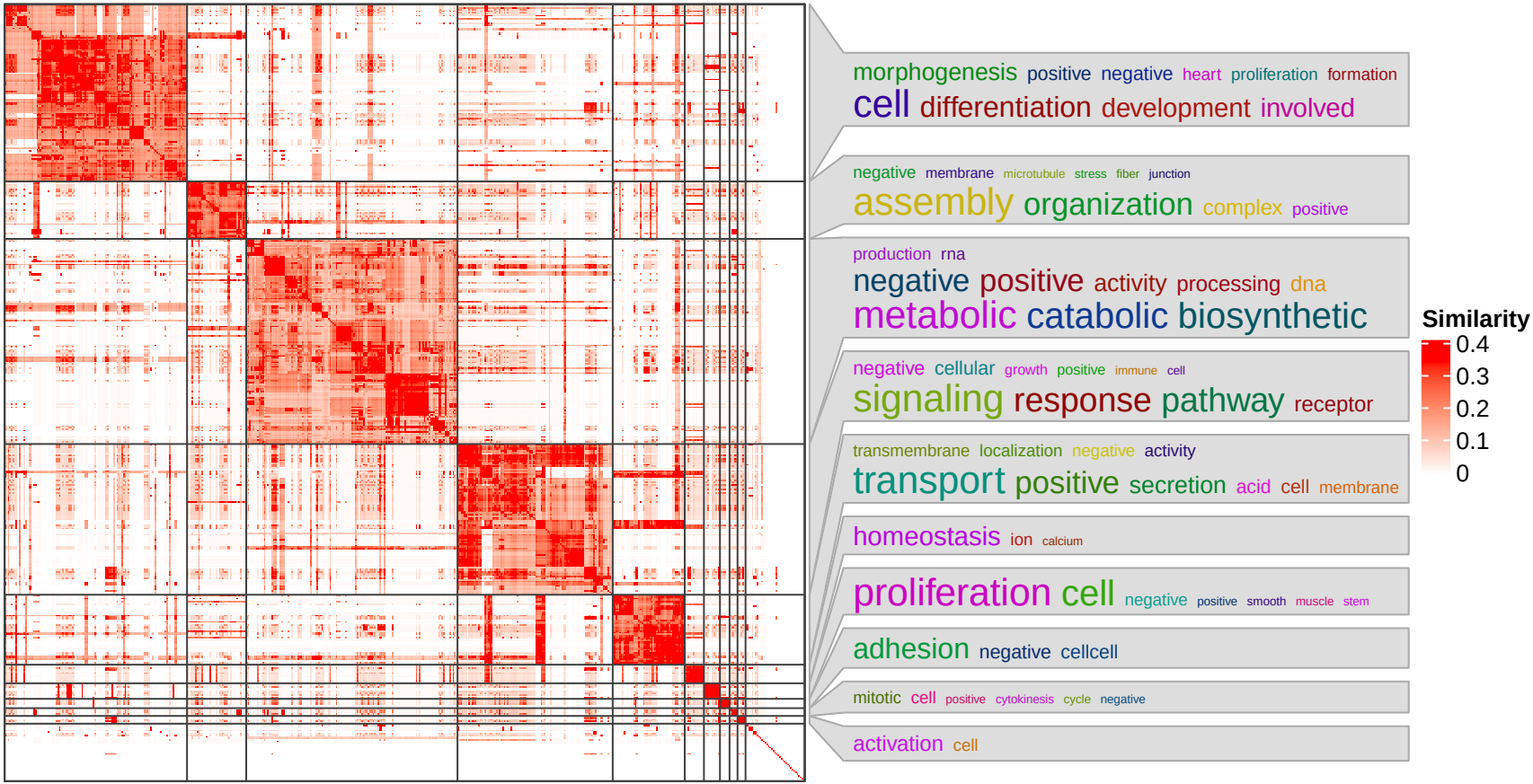
drugs. *Nucleic Acids Res.*, **45**, D353–D361.
Kapushesky,M. *et al.* (2010) Gene expression atlas at the European bioinformatics institute. *Nucleic Acids Res.*, **38**, D690-8.
Khatri,P. and Drăghici,S. (2005) Ontological analysis of gene expression data: current tools, limitations, and open problems. *Bioinformatics*, **21**, 3587–3595.
Langfelder,P. *et al.* (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *Bioinformatics*, **24**, 719–720.
Leisch,F. (2006) A toolbox for -centroids cluster analysis. *Comput. Stat. Data Anal.*, **51**, 526–544.
Liao,Y. *et al.* (2019) WebGestalt 2019: gene set analysis toolkit with revamped UIs and APIs. *Nucleic Acids Res.*, **47**, W199–W205.
Liberzon,A. *et al.* (2015) The Molecular Signatures Database (MSigDB) hallmark gene set collection. *Cell Syst.*, **1**, 417–425.
Lin,D. (1998) An Information-Theoretic Definition of Similarity. In, *Machine Learning Proceedings 1998*, ICML '98. Morgan Kaufmann, San Francisco, CA, USA, p. 592.
Mazandu,G.K. *et al.* (2017) Gene Ontology semantic similarity tools: survey on features and challenges for biological knowledge discovery. *Brief. Bioinformatics*, **18**, 886–901.
McLean,C.Y. *et al.* (2010) GREAT improves functional interpretation of cis-regulatory regions. *Nat. Biotechnol.*, **28**, 495–501.
Merico,D. *et al.* (2010) Enrichment map: a network-based method for gene-set enrichment visualization and interpretation. *PLoS ONE*, **5**, e13984.
Resnik,P. (1999) Semantic Similarity in a Taxonomy: An Information-Based Measure and Its Application to Problems of Ambiguity in Natural Language. *J. Artif. Int. Res.*, **11**, 95–130.
Schlicker,A. *et al.* (2006) A new measure for functional similarity of gene products based on Gene Ontology. *BMC Bioinformatics*, **7**, 302.
Schriml,L.M. *et al.* (2012) Disease Ontology: a backbone for disease semantic integration. *Nucleic Acids Res.*, **40**, D940-6.
Scrucca,L. *et al.* (2016) mclust 5: Clustering, Classification and Density Estimation Using Gaussian Finite Mixture Models. *R J.*, **8**, 289–317.
Subramanian,A. *et al.* (2005) Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci USA*, **102**, 15545–15550.
The Gene Ontology Consortium (2019) The Gene Ontology Resource: 20 years and still GOing strong. *Nucleic Acids Res.*, **47**, D330–D338.
Van Dongen,S. (2008) Graph clustering via a discrete uncoupling process. *SIAM. J. Matrix Anal. & Appl.*, **30**, 121–141.
Vassilvitskii,S. (2007) K-means : the advantages of careful seeding. *IN PROCEEDINGS OF THE 18TH ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS*.
Wang,G. *et al.* (2020) GOMCL: a toolkit to cluster, evaluate, and extract non-redundant associations of Gene Ontology-based functions. *BMC Bioinformatics*, **21**, 139.
Wang,J.Z. *et al.* (2007) A new method to measure the semantic similarity of GO terms. *Bioinformatics*, **23**, 1274–1281.
Yu,G. *et al.* (2012) clusterProfiler: an R package for comparing biological themes among gene clusters. *OMICS*, **16**, 284–287.
Yu,G. *et al.* (2015) DOSE: an R/Bioconductor package for disease ontology semantic and

enrichment analysis. *Bioinformatics*, **31**, 608–609.
Yu,G. *et al.* (2010) GOSemSim: an R package for measuring semantic similarity among GO terms and gene products. *Bioinformatics*, **26**, 976–978.
Zhao,C. and Wang,Z. (2018) GOGO: An improved algorithm to measure the semantic similarity between gene ontology terms. *Sci. Rep.*, **8**, 15107.

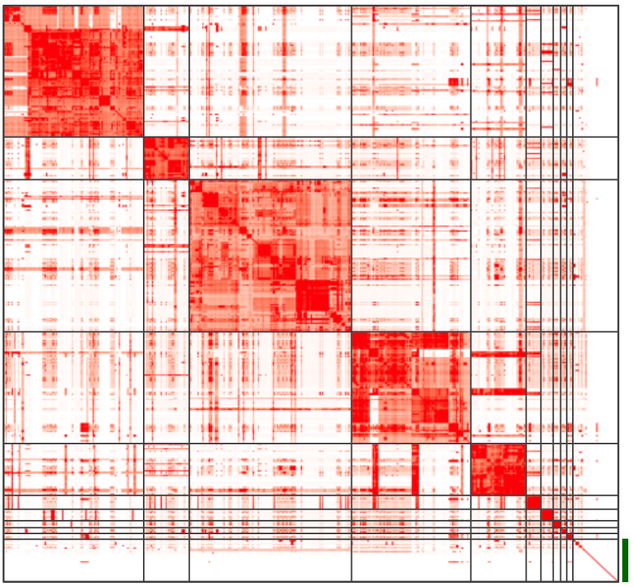

A) Similarity matrix $\mathbf{M}_a$
s_{11}
s_{12}
s_{21}
s_{22}
B) Similarity matrix $\mathbf{M}_b$
s_{11}
s_{12}
s_{21}
s_{22}
Similarity
0.6
0.4
0.2
0


A) Depth = 1
B) Depth = 2
C) Depth = 3
D) Complete dendrogram with nodes split
Similarity
0.4
0.3
0.2
0.1
0
Score
1
0.8
0.6
0.4

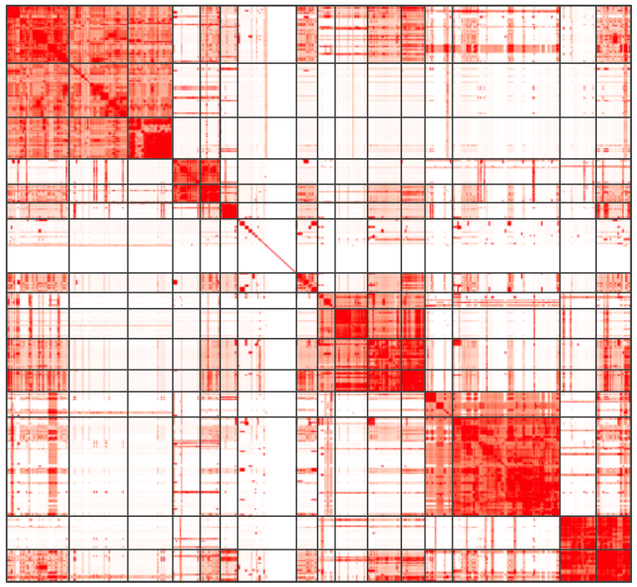
Similarity matrix of 500 random GO terms



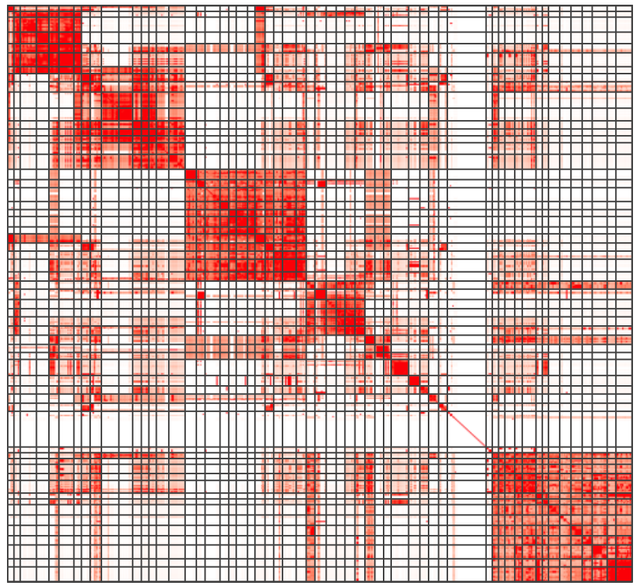
A) 500 terms clustered by 'binary_cut'



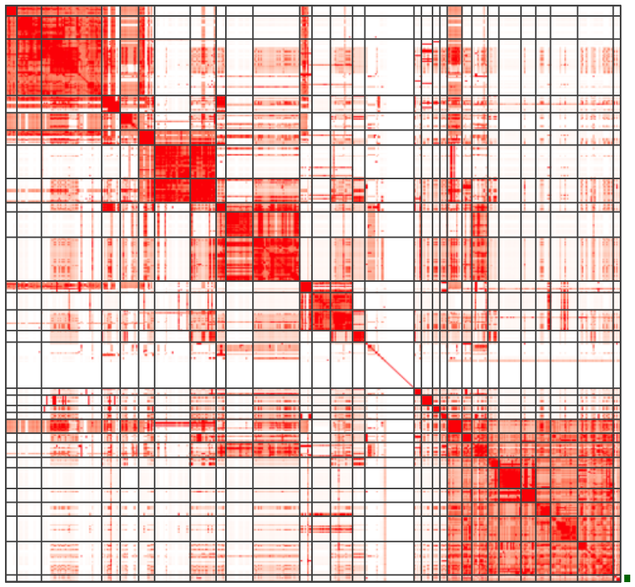
B) 500 terms clustered by 'kmeans'



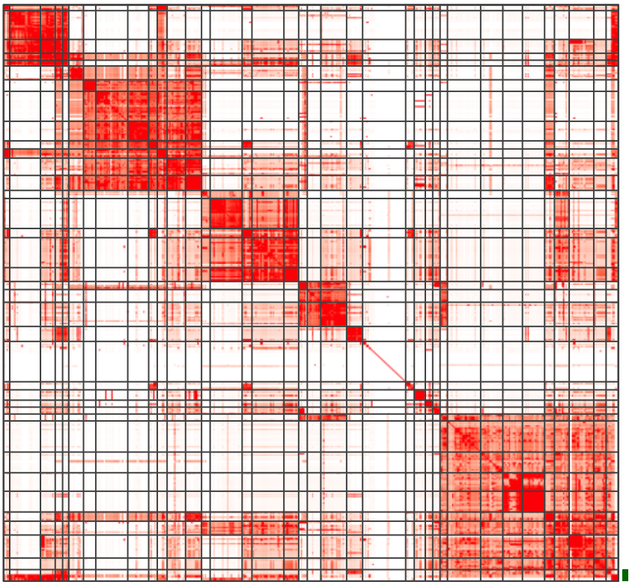
C) 500 terms clustered by 'dynamicTreeCut'



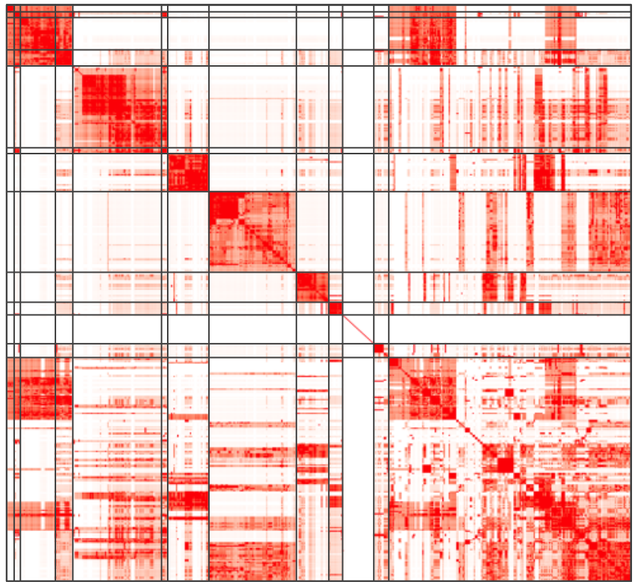
D) 500 terms clustered by 'mclust'



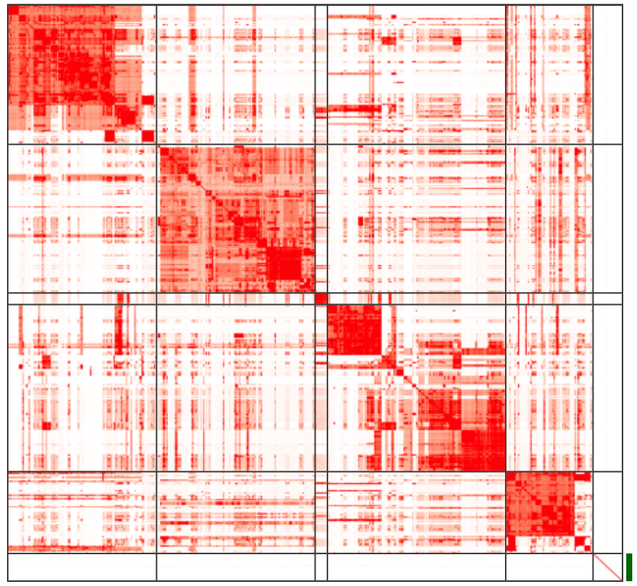
E) 500 terms clustered by 'apcluster'



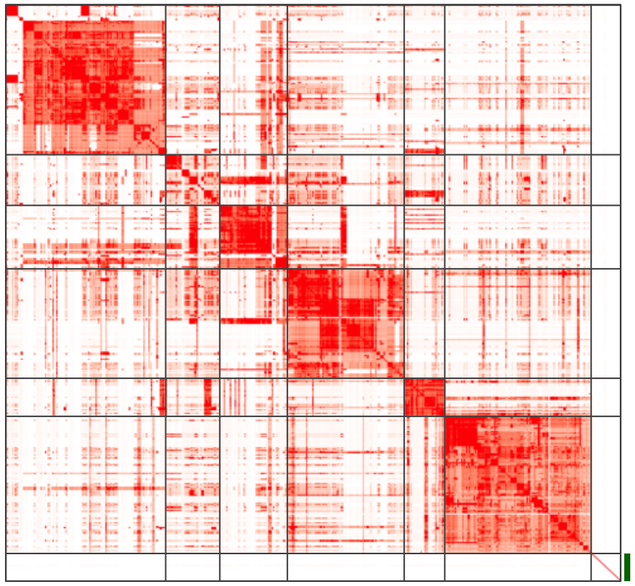
F) 500 terms clustered by 'hdbscan'



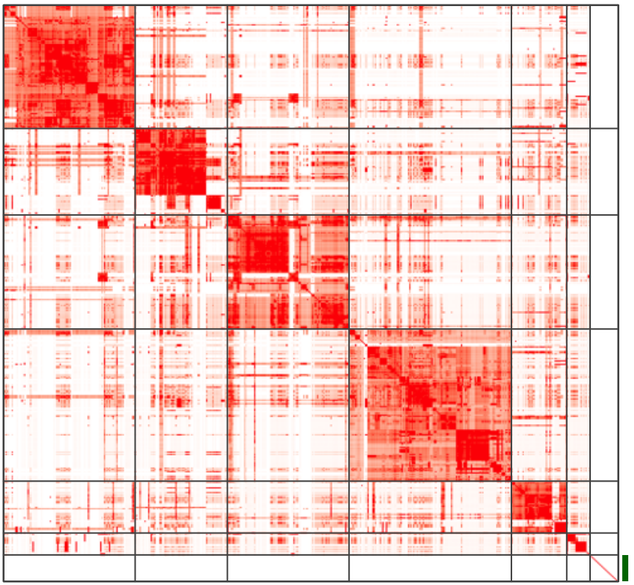
G) 500 terms clustered by 'fast_greedy'



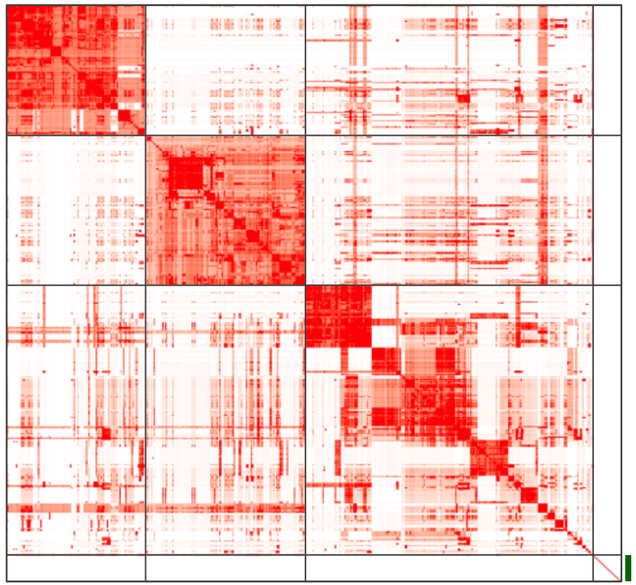
H) 500 terms clustered by 'leading_eigen'



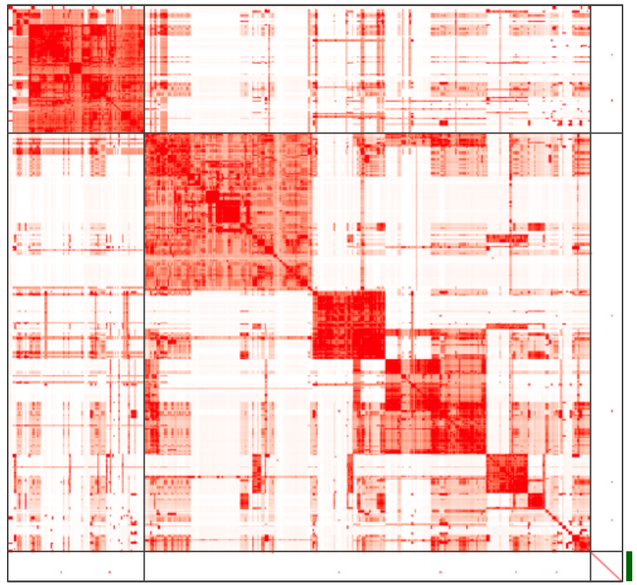
I) 500 terms clustered by 'louvain'



J) 500 terms clustered by 'walktrap'



K) 500 terms clustered by 'MCL'

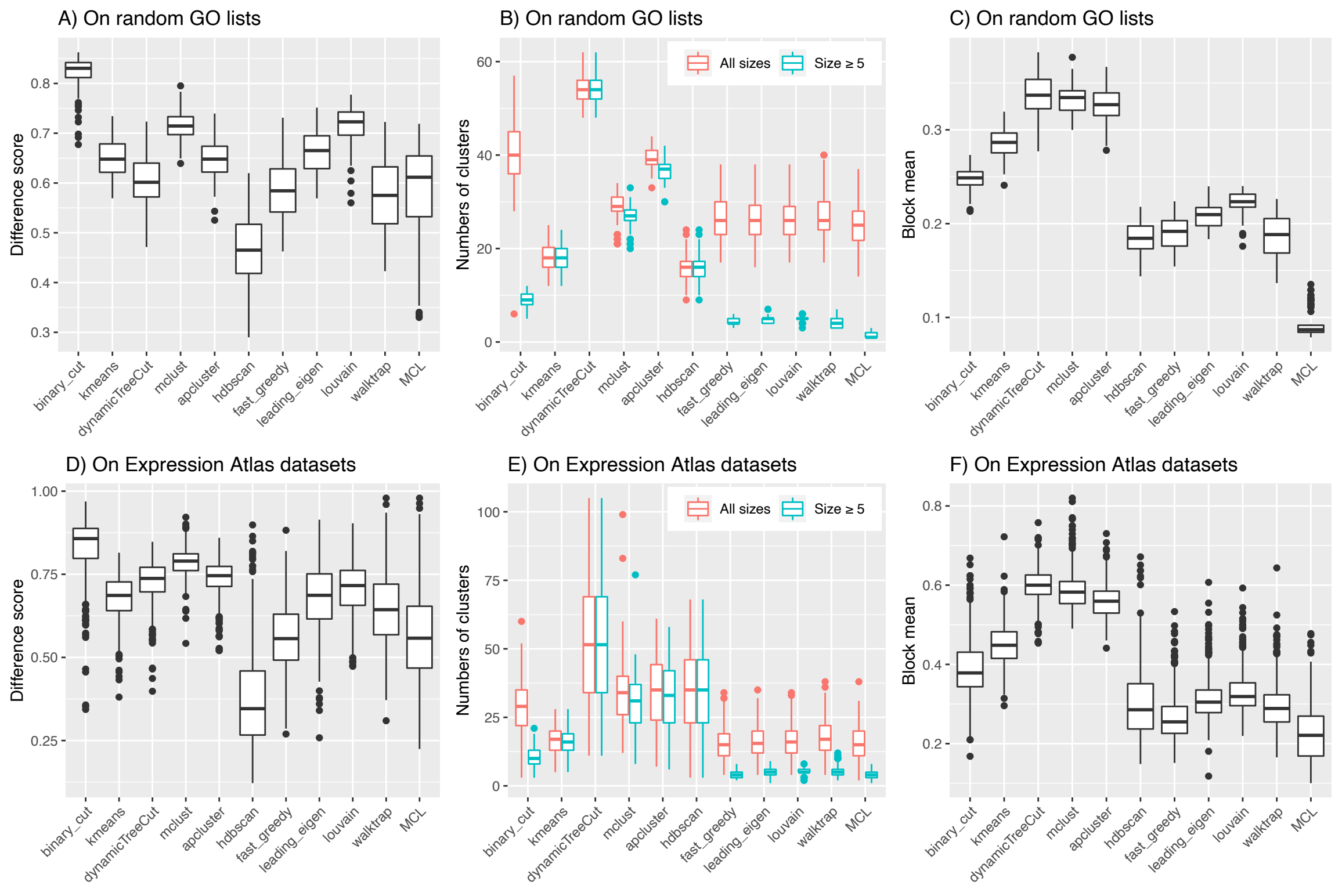


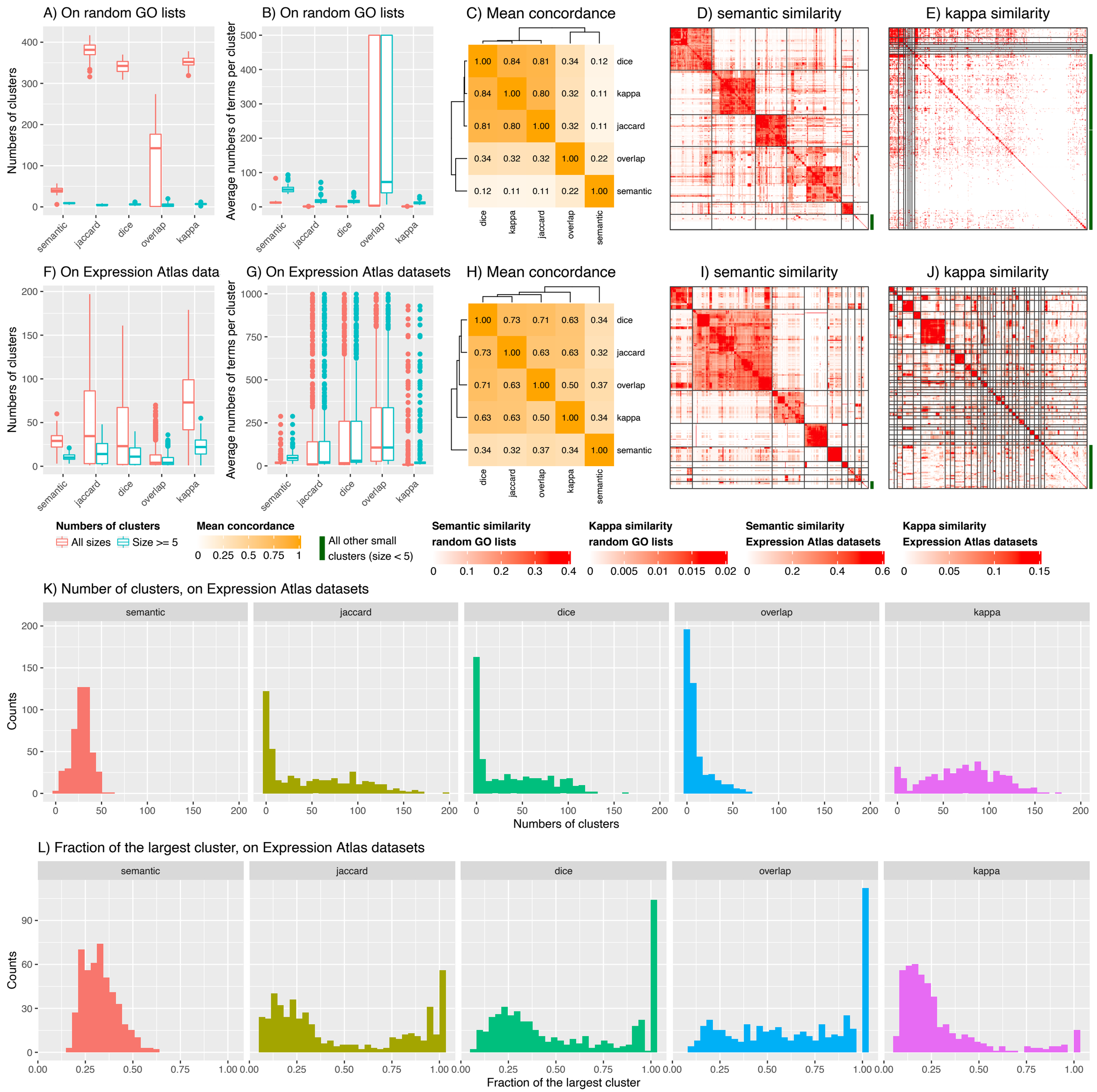
L) Numbers of clusters

Method	All clusters	Large clusters (size ≥ 5)
binary_cut	43	10
kmeans	16	16
dynamicTreeCut	58	58
mclust	31	29
apcluster	39	36
hdbscan	13	13
fast_greedy	29	5
leading_eigen	30	6
louvain	29	6
walktrap	26	3
MCL	28	2

Similarity 0 0.1 0.2 0.3 0.4

All other small clusters (size < 5)


A) On random GO lists
B) On random GO lists
C) On random GO lists
D) On Expression Atlas datasets
E) On Expression Atlas datasets
F) On Expression Atlas datasets
Difference score
Numbers of clusters
Block mean
All sizes
Size ≥ 5
binary_cut
kmeans
dynamicTreeCut
mclust
apcluster
hdbscan
fast_greedy
leading_eigen
louvain
walktrap
MCL


A) On random GO lists
Numbers of clusters
semantic
jaccard
dice
overlap
kappa
B) On random GO lists
Average numbers of terms per cluster
C) Mean concordance
1.00 0.84 0.81 0.34 0.12 dice
0.84 1.00 0.80 0.32 0.11 kappa
0.81 0.80 1.00 0.32 0.11 jaccard
0.34 0.32 0.32 1.00 0.22 overlap
0.12 0.11 0.11 0.22 1.00 semantic
D) semantic similarity
E) kappa similarity
F) On Expression Atlas data
G) On Expression Atlas datasets
H) Mean concordance
1.00 0.73 0.71 0.63 0.34 dice
0.73 1.00 0.63 0.63 0.32 jaccard
0.71 0.63 1.00 0.50 0.37 overlap
0.63 0.63 0.50 1.00 0.34 kappa
0.34 0.32 0.37 0.34 1.00 semantic
I) semantic similarity
J) kappa similarity
Numbers of clusters
All sizes
Size >= 5
Mean concordance
0 0.25 0.5 0.75 1
All other small clusters (size < 5)
Semantic similarity random GO lists
0 0.1 0.2 0.3 0.4
Kappa similarity random GO lists
0 0.005 0.01 0.015 0.02
Semantic similarity Expression Atlas datasets
0 0.2 0.4 0.6
Kappa similarity Expression Atlas datasets
0 0.05 0.1 0.15
K) Number of clusters, on Expression Atlas datasets
Counts
Numbers of clusters
L) Fraction of the largest cluster, on Expression Atlas datasets
Fraction of the largest cluster