BMC Bioinformatics

SOFTWARE **Open Access**

Coral: an integrated suite of visualizations for comparing clusterings

Darya Filippova[1*], Aashish Gadani[3] and Carl Kingsford[1,2]

Abstract

Background: Clustering has become a standard analysis for many types of biological data (e.g interaction networks, gene expression, metagenomic abundance). In practice, it is possible to obtain a large number of contradictory clusterings by varying which clustering algorithm is used, which data attributes are considered, how algorithmic parameters are set, and which near-optimal clusterings are chosen. It is a difficult task to sift though such a large collection of varied clusterings to determine which clustering features are affected by parameter settings or are artifacts of particular algorithms and which represent meaningful patterns. Knowing which items are often clustered together helps to improve our understanding of the underlying data and to increase our confidence about generated modules.

Results: We present Coral, an application for interactive exploration of large ensembles of clusterings. Coral makes all-to-all clustering comparison easy, supports exploration of individual clusterings, allows tracking modules across clusterings, and supports identification of core and peripheral items in modules. We discuss how each visual component in Coral tackles a specific question related to clustering comparison and provide examples of their use. We also show how Coral could be used to visually and quantitatively compare clusterings with a ground truth clustering.

Conclusion: As a case study, we compare clusterings of a recently published protein interaction network of *Arabidopsis thaliana*. We use several popular algorithms to generate the network's clusterings. We find that the clusterings vary significantly and that few proteins are consistently co-clustered in all clusterings. This is evidence that several clusterings should typically be considered when evaluating modules of genes, proteins, or sequences, and Coral can be used to perform a comprehensive analysis of these clustering ensembles.

Background

Collections of protein interactions, gene expression vectors, metagenomic samples, and gene sequences containing thousands to hundreds-of-thousands of elements are now being analyzed routinely. Clustering is often used to condense such large datasets into an understandable form: it has been successfully used on protein-protein interaction (PPI) networks to discover protein complexes and predict protein function, e.g. [1]; on gene expression data to find patterns in gene regulation and essential cell processes, e.g. [2]; and on metagenomic samples to identify new species, compare them to existing clades, evaluate the diversity of a population, and suggest interdependencies between them [3,4]. In other words, clustering has become a ubiquitous part of analysis for large biological datasets.

There are many clustering algorithms available for numerical and network data, e.g. [5-12]. Each algorithm, and choice of its parameters, results in different clusterings. Sometimes, clustering algorithms must resolve ties when generating modules or may be randomized. Consequently, a single clustering algorithm may produce diverse partitions on the same data [13]. Clusterings may also change when the underlying data becomes increasingly noisy or displays variation under different conditions (such as varying gene expression levels). In addition, considering many optimal and near-optimal partitions has been shown to improve the understanding of module dynamics and the strength of relationships between individual items [14-17]. Such clusterings may offer drastically

*Correspondence: dfilippo@cs.CMU.edu
[1]Lane Center for Computational Biology, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Full list of author information is available at the end of the article

different perspectives on the data, where assessing the commonalities and differences is of great interest.

There are several ways in which the problem of diverse clusterings has been addressed. Some tools rely on a single clustering only and focus on module quality assessment, e.g. [18,19]. Comparing two or more clusterings at a time is usually done by computing a single metric, such as the Jaccard or Rand index [20], to compare clusterings side-by-side [21] or in a dendrogram [22]. These approaches can easily compare a pair of clusterings, but are not extendable to greater number of clusterings. Another approach is to aggregate multiple partitions into a consensus clustering [23,24] without delving into the differences between individual clusterings and, thus, disregarding possibly important information about the clusterings. Finally, some approaches have made steps towards visual examination of multiple clusterings: King and Grimmer [25] compare clusterings pairwise and project the space of clusterings onto a plane to visualize a clustering landscape, and Langfelder et al. [26] investigate ways to compare individual modules across multiple conditions. However, none of these approaches offer a platform for a multi-level analysis of ensembles of diverse clusterings.

In the present study, we describe Coral—a tool that allows for interactive and comprehensive comparison of multiple clusterings at different levels of abstraction. Coral allows users to progressively move from an overview to analysis of relationships between individual data items. Users may start by examining statistics on the data and individual clusterings, or by assessing the overall homogeneity of a dataset. Users can then compare any two partitions in the ensemble in greater detail. Larger scale trends become pronounced when groups of items that co-cluster often are automatically identified and highlighted. We also extend parallel sets plot [27] to show how individual items switch modules across clusterings. Coral's visualizations are interactive and are coordinated so that users can track the same group of data items across multiple views.

Implementation

In Coral's design, we followed the visualization mantra coined by Shneiderman [28]: overview, zoom-and-filter, details-on-demand. At the overview level, Coral displays dataset statistics and highlights the most similar and dissimilar clusterings; at the mid-level, "zoomed-in," analysis explains similarities between clusterings through module comparison; the low-level analysis compares co-clustering patterns at the level of individual data items: the genes, proteins, or sequences. The displays are coordinated [29] so selecting data in one of the views highlights the corresponding items in the other views (see Figure 1).

Coral works with modules — groups of items closely related to one another according to some metric or property. For example, modules can constitute a collection of genes that get co-expressed together or proteins forming a complex. A clustering is a collection of modules and usually is an output of a clustering algorithm. Users may also choose to group data according to attributes that come with the data such as cellular component or molecular function GO terms and use that partition as a clustering. Users may combine data from different experiemnts and across species so long as the data items that the user treats as homologous have the same IDs across the dataset.

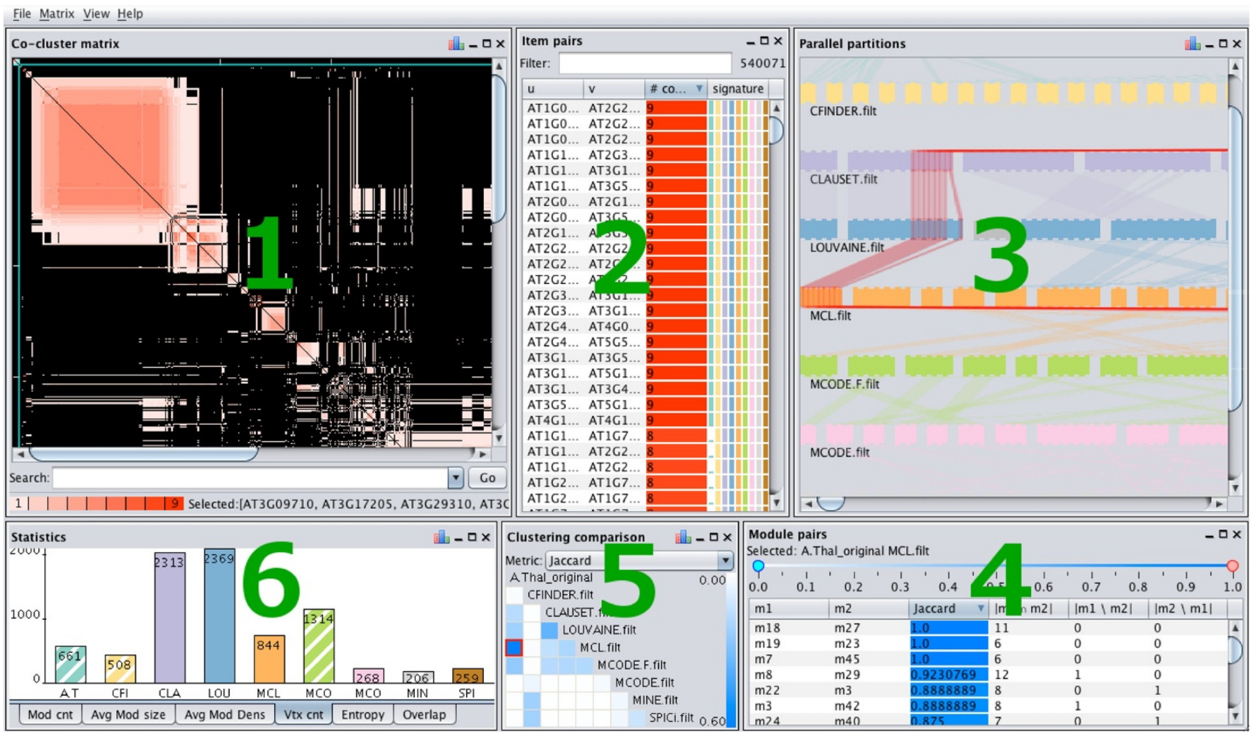
Coral takes as an input the module files where each file represents a single clustering, and each line in the file contains a list of data items (proteins, genes, or sequence ids) from a single module, e.g. as produced by MCL, the clustering algorithm by van Dongen [5]. Coral aggregates and visualizes these data through several connected displays, each of which can be used to answer specific questions about the clusterings. Below, we examine a few such questions and describe how Coral's visualizations help to answer them.

How many and what size modules do clusterings have?

To gain a quick overview of their collection of clusterings, Coral users may start the analysis by reviewing basic statistics about their data: number of modules per clustering, average module size, number of items that were clustered, clustering entropy [30], and percentage of data items that ended up in the overlapping modules. Questions such as "Do clusterings have the same number of modules?" and "Are module sizes evenly distributed?" can be easily answered through these statistics. Each statistic is shown as a bar chart, and each clustering is associated with a unique color hue that is used consistently to identify the clustering throughout the system. If a clustering contains overlapping modules, the corresponding bar in the chart is striped as opposed to a solid bar for the clusterings containing no overlapping modules (see Figure 1).

Which clusterings are the most and least similar?

Coral computes similarity scores between all clusterings and visualizes the lower triangle of the similarity matrix in a ladder widget (Figure 2). The ladder compactly displays similarity scores for every pair of clusterings in the ensemble allowing for precise comparisons. Coral offers a choice of several similarity measures to compare partitions: Jaccard coefficient, Mirkin metric, Rand index, Folkes-Mallow metric, mutual information, variation of information, purity and inverse purity, and the F-measure [30]. The ladder is color-coded as a heatmap with more intense blue cells corresponding to higher similarity scores and paler cells corresponding to low scores.



Item pairs

Filter: 540071

u	v	# co...	signature
AT1G0...	AT2G2...	9	
AT1G0...	AT2G2...	9	
AT1G0...	AT2G2...	9	
AT1G1...	AT2G3...	9	
AT1G1...	AT3G1...	9	
AT1G1...	AT3G5...	9	
AT2G0...	AT2G1...	9	
AT2G0...	AT3G5...	9	
AT2G1...	[illegible]	9	
AT2G2...	AT2G2	9	
AT2G2...	AT2G...	9	
AT2G2...	[illegible]	9	
AT2G3...	AT3G1...	9	
AT2G3...	AT3G1...	9	
AT2G4...	AT4G0...	9	
AT2G4...	AT5G5...	9	
AT3G1...	AT3G5...	9	
AT3G1...	AT5G1...	9	
AT3G1...	AT3G4...	9	
AT3G5...	AT5G1...	9	
AT4G1...	AT4G1...	9	
AT1G1...	AT1G7...	8	
AT1G1...	AT2G2...	8	
AT1G1...	AT2G2...	8	
AT1G2...	AT1G7...	8	
AT1G2...	AT1G7...	8	

Statistics

A.T	CFI	CLA	LOU	MCL	MCO	MCO	MIN	SPI
661	508	2313	2369	844	1314	268	206	259

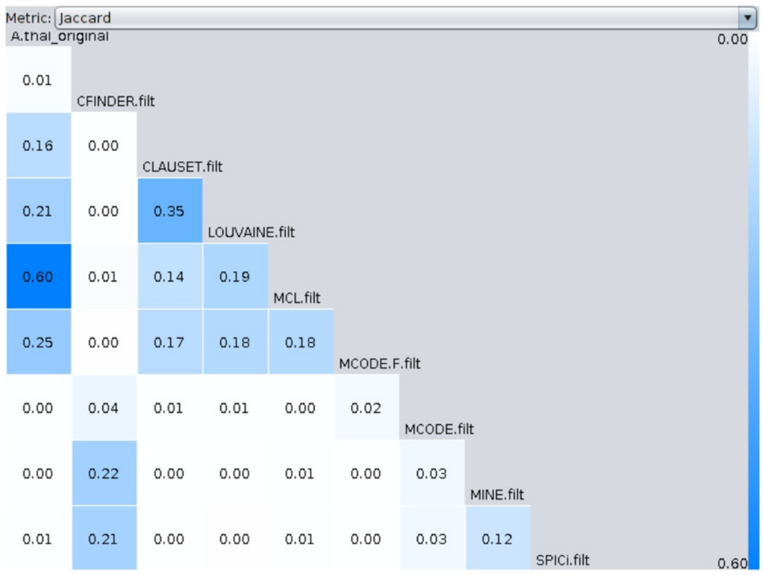
Module pairs

Selected: A.Thal_original MCL.filt

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

m1	m2	Jaccard	|m1 ∩ m2|	|m1 \ m2|	|m2 \ m1|
m18	m27	1.0	11	0	0
m19	m23	1.0	6	0	0
m7	m45	1.0	6	0	0
m8	m29	0.9230769	12	1	0
m22	m3	0.8888889	8	0	1
m3	m42	0.8888889	8	1	0
m24	m40	0.875	7	0	1

Figure 1 Coral overview. Coral views in a clockwise direction: co-cluster matrix (1), pairs table (2), parallel partitions plot (3), module-to-module table (4), ladder (5), overview statistics (6). Users may rearrange the individual views or close them to focus on fewer visualizations at a time. Data: *A. thaliana* clusterings.


Metric: Jaccard

	A.thal_original	CFINDER.filt	CLAUSET.filt	LOUVAINE.filt	MCL.filt	MCODE.F.filt	MCODE.filt	MINE.filt
CFINDER.filt	0.01							
CLAUSET.filt	0.16	0.00						
LOUVAINE.filt	0.21	0.00	0.35					
MCL.filt	0.60	0.01	0.14	0.19				
MCODE.F.filt	0.25	0.00	0.17	0.18	0.18			
MCODE.filt	0.00	0.04	0.01	0.01	0.00	0.02		
MINE.filt	0.00	0.22	0.00	0.00	0.01	0.00	0.03	
SPICi.filt	0.01	0.21	0.00	0.00	0.01	0.00	0.03	0.12

Figure 2 All-to-all clustering comparison in a ladder widget. The ladder represents a lower triangle of an all-to-all matrix where each cell (i, j) holds a score for similarity between clusterings K_i and K_j. Users can choose between several comparison metrics by toggling a dropdown above the ladder. Every cell is color-coded, with darker colors indicating more similarity between the pair.

Clicking a cell updates the contents of a module-to-module comparison widget (see next subsection).

Which modules are the most similar between the two clusterings?

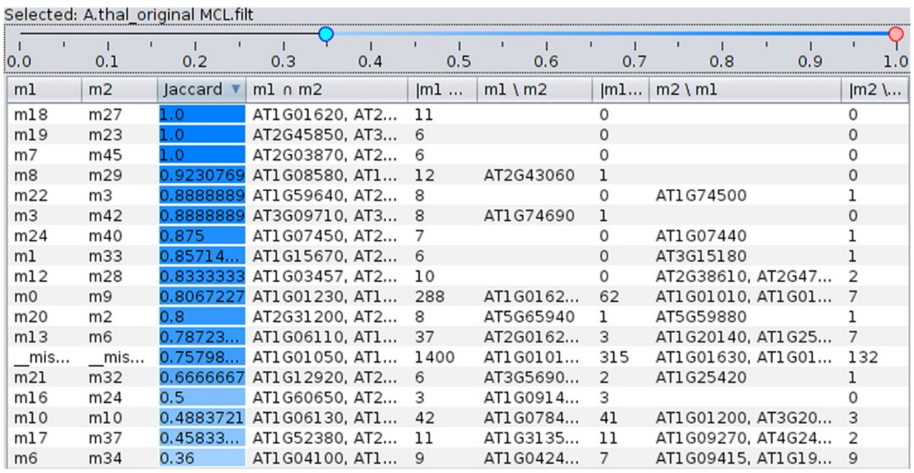
A natural follow-up to finding a highly similar pair of clusterings is to review the similarities between their individual modules. Is a group of interacting genes preserved between the two stages in the cell life cycle? Is there a match for a given protein complex in the PPI network of another species? Module-to-module comparison answers these questions and explains the origins of clustering similarity at a "zoomed-in" module level.

For a given pair K_1, K_2 of clusterings, Coral calculates the Jaccard similarity $J = |m_i^1 \bigcap m_j^2| / |m_i^1 \bigcup m_j^2|$ between every module $m_i^1 \in K_1$ and $m_j^2 \in K_2$ thus capturing the amount of overlap between the two modules. For every such module pair, Coral displays the pair's Jaccard score and items in the union, intersection, left and right set differences. All module pairs are organized in a sortable table (see Figure 3). The slider above the table allows the user to filter out module pairs for which the Jaccard score is outside the slider's range allowing users to focus on highly similar (or dissimilar) modules. Although module-to-module analysis is possible with the parallel partitions plot (discussed below), the table offers a sortable and filterable view of the same data while supplying additional information (e.g. Jaccard index). The module-to-module table shows only the module pairs with some overlap and easily scales to hundreds of modules, thereby offering a more compact and easily navigable alternative to a confusion matrix (e.g. as used in [26]).

Does this module appear in other clusterings?

The ability to track individual items and whole modules across multiple clusterings provides a high level of abstraction in clustering analysis: modules may split and merge as users switch from clustering to clustering. To afford an exploration at the module level, we have developed a parallel partitions plot — an extension of a parallel sets plot used in the analysis of categorical multivariate data [27]. The parallel partitions plot represents each clustering as a horizontal band. The blocks comprising each bands represent modules, with the width of a block proportional to the number of items in that module. Semi-transparent parallelograms between clusterings connect data items with the same name. That is, each item in a clustering will be connected to its copy in the band immediately below it (see Figure 4).

The parallel partitions plot allows users to track individual items and whole modules across all partitions. To see whether a module splits or merges in other clusterings, users can select a module with a mouse while holding a shift key to highlight its members in every clustering in the plot (see red traces in Figure 4). Similarly, users may select individual items and trace them through every clustering band. The selections made in the parallel partitions plot propagate to other views making it easy to track the same group of items throughout the application. The plot is zoomable — users may zoom in to focus on a few items at a time or zoom out to see global trends across the ensemble. When the zoom level permits it, the plot displays the item labels.



m1	m2	Jaccard ▼	m1 ∩ m2	|m1 ...	m1 \ m2	|m1...	m2 \ m1	|m2 \...
m18	m27	1.0	AT1G01620, AT2...	11		0		0
m19	m23	1.0	AT2G45850, AT3...	6		0		0
m7	m45	1.0	AT2G03870, AT2...	6		0		0
m8	m29	0.9230769	AT1G08580, AT1...	12	AT2G43060	1		0
m22	m3	0.8888889	AT1G59640, AT2...	8		0	AT1G74500	1
m3	m42	0.8888889	AT3G09710, AT3...	8	AT1G74690	1		0
m24	m40	0.875	AT1G07450, AT2...	7		0	AT1G07440	1
m1	m33	0.85714...	AT1G15670, AT2...	6		0	AT3G15180	1
m12	m28	0.8333333	AT1G03457, AT2...	10		0	AT2G38610, AT2G47...	2
m0	m9	0.8067227	AT1G01230, AT1...	288	AT1G0162...	62	AT1G01010, AT1G01...	7
m20	m2	0.8	AT2G31200, AT2...	8	AT5G65940	1	AT5G59880	1
m13	m6	0.78723...	AT1G06110, AT1...	37	AT2G0162...	3	AT1G20140, AT1G25...	7
__mis...	__mis...	0.75798...	AT1G01050, AT1...	1400	AT1G0101...	315	AT1G01630, AT1G01...	132
m21	m32	0.6666667	AT1G12920, AT2...	6	AT3G5690...	2	AT1G25420	1
m16	m24	0.5	AT1G60650, AT2...	3	AT1G0914...	3		0
m10	m10	0.4883721	AT1G06130, AT1...	42	AT1G0784...	41	AT1G01200, AT3G20...	3
m17	m37	0.45833...	AT1G52380, AT2...	11	AT1G3135...	11	AT1G09270, AT4G24...	2
m6	m34	0.36	AT1G04100, AT1...	9	AT1G0424...	7	AT1G09415, AT1G19...	9

Figure 3 Module-to-module comparison for two clusterings. When users decide to focus on a pair of clusterings, they may explore all pairs of modules in a sortable table. Each module pair is shown against its Jaccard score, and lists of items in the module intersection, left and right differences. Users can filter the table rows by Jaccard score to only show rows within a given similarity range by adjusting the slider above the table. Cells holding the Jaccard scores are color-coded to indicate similarity.

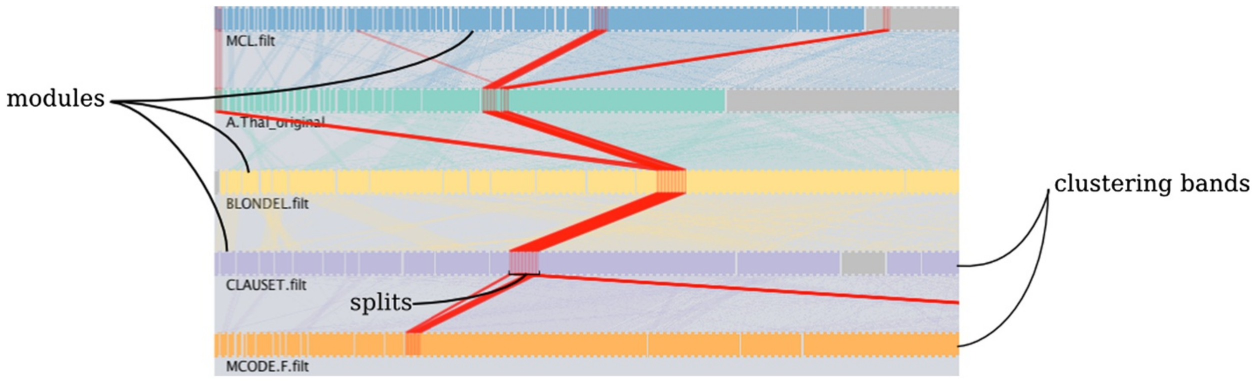


Figure 4 Parallel partitions maps modules between clusterings. Horizontal bands represent partitions, and modules are separated by blank spaces. Semi-transparent bands connect the same items from different clusterings. Red traces highlight selected items across all partitions and show how modules split and merge. Here, the user has selected a group of 215 proteins that belong to the largest core in the ensemble of clusterings for *A. thaliana* PPI network. The `Louvaine`, `Clauset`, `MCL`, and `MCODE.F` clusterings assign all of the selected proteins to a single module. For other clusterings, most of the selected proteins are placed in a grab bag region of items that were not contained in any module (shown in gray).

The order of items in the clustering bands matches the order of items in the co-cluster matrix (discussed below) as closely as possible, while at the same time attempting to minimize the amount of crossover between the parallelograms connecting items in the consecutive clusterings. However, the items in the bands must be placed inside their respective modules. We discuss an algorithm that finds a good ordering of items in the clustering bands in the Methods section.

What other items are in the same module as a given item *u*?

A single clustering assigns a data item v to a module defining its *cohort* — a set of items in the same module as v. Knowing the item's module helps in assigning function to unknown proteins [6] and novel genes [2]; knowing that the item's cohort is consistent across many clusterings increases the confidence of such predictions.

In Coral, pairwise co-cluster memberships are aggregated in a *co-cluster matrix* [24]. Given a single clustering K, $n = |K|$, we define an $n \times n$ matrix A^K to be K's co-cluster matrix where its entries a^K_{ij} are:

$$a^K_{ij} = \begin{cases} 0 & v_i \text{ and } v_j \text{ are in different modules in } K \\ 1 & v_i \text{ and } v_j \text{ are in the same module in } K. \end{cases}$$

For some item pairs, co-clustering may be an artifact of a tie-breaking policy or a choice of an algorithm parameter: such item pairs may only co-cluster in few clusterings. On the other hand, we would like to know whether there were item pairs that co-clustered across most partitions in the ensemble. These cohort dynamics stand out if we sum up the co-cluster matrices to form a single matrix:

$$A^+ = \sum_{t=1}^{k} A^{K_t},$$

where A^{K_t} is a co-cluster matrix for a clustering K_t and k is the number of clusterings. Here, the a^+_{ij} entries equal k (the number of clusterings) for item pairs (v_i, v_j) that have co-clustered in all partitions suggesting a strong relationship between the items, and the low a^+_{ij} values correspond to pairs that co-clustered in only a few clusterings and are more likely to have been assigned to the same module by chance. The cells are colored according to their values and vary from white (low values) to red (high values). Users may zoom in and out on the matrix to focus on areas of interest.

The co-cluster matrix is hard to read unless similar rows and columns are placed near each other. Reordering the rows and columns of A^+ brings non-zero entries closer to the diagonal and exposes any modular structure. When clusterings are highly similar, the reordered matrix will consist of blocks along the diagonal with high a^+_{ij} values (Figure 5). Clusterings that are very dissimilar produce reorderings similar to Figure 6 — the diagonal blocks mostly contain low a^+_{ij} values (colored white or light pink) with many entries away from the diagonal.

Are there whole groups of items that co-cluster together often?

Groups of items that end up in the same module across many clusterings are of a particular interest because they represent the robust subclusters in the data. Such commonly co-clustered sets are called *cores*. Items in cores

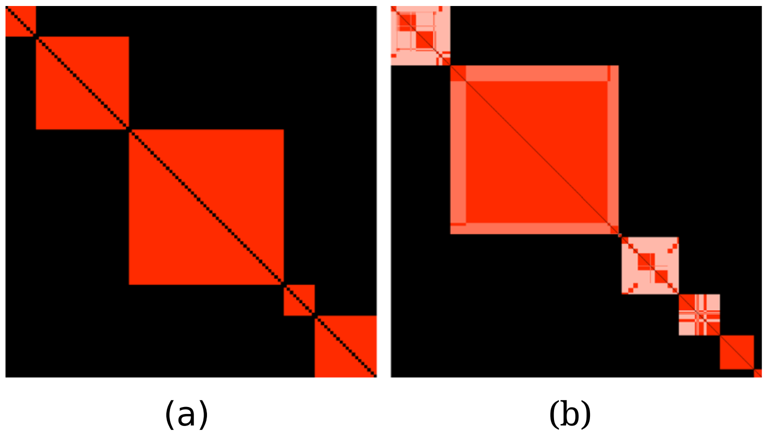


Figure 5 (a) A co-cluster matrix of three identical decompositions forms blocks on the diagonal with only red cells indicating that all three clsuterings agreed (synthetic data). (b) Big modules were broken up into smaller modules to form new clusterings (four `Hybrid` decompositions from the Langfelder study [16]).

form the most trustworthy modules and indicate particularly strong ties between data items, increasing, for example, the confidence in protein complex identification [32] and gene annotation [33].

In a co-cluster matrix, cores translate to contiguous blocks of high-value entries. Coral finds the cores using a fast dynamic programming algorithm and highlights them within the co-cluster matrix (inset, Figure 6a). When users provide clusterings derived from a network, Coral can augment cores with an overlay showing each core's cohesion — the ratio $E_{\text{in}}/E_{\text{out}}$ where E_{in} is the number of edges within the core and E_{out} is the number of edges that have one endpoint inside the core and another endpoint outside of it [34]. When a core's cohesion is low, the blue overlay is smaller indicating that the core shares many connections with the rest of the network (Figure 6b). Cores for which cohesion is high are more isolated from the rest of the network — these cores are distinguishable by the blue overlays that almost cover the core.

Do items within a ground-truth clustering often co-cluster in other clusterings?

When validating new protein complexes or co-expressed gene modules, users may want to see how well their results match ground-truth clusterings such as protein complexes from MIPS [35], or sets of co-regulated genes from RegulonDB [36]. In Coral, users may designate a single clustering as a *base* — a set of trusted modules with which other clusterings are expected to agree. When in this mode, Coral will only highlight those cells in the co-cluster matrix that are within the modules of the base and gray out all other non-zero matrix cells to bring users' attention to the clustering in question. Figure 7 shows an example of a co-cluster matrix with the base set to be the *A. thaliana* modules reported in [31].

In which clusterings do particular items co-cluster?

The co-cluster matrix displays the total number of times any two items were co-clustered, and the tooltips that appear after hovering over a matrix cell show a list of clusterings in which a given pair has been co-clustered. To facilitate sorting and search for particular item pairs, Coral provides a companion table where each row represents a pair of data items and displays the number of times the items co-clustered along with the pair's *signature*. The signature is a k-long vector where the t^{th} element is 1 when both data items, say, proteins, have been placed in the same module in clustering K_t. If the pair's items were not in the same module in K_t, the t^{th} element is set to 0.

Visually, the signature's elements that are 1 are drawn as tall solid columns and zeros are represented by the short stumps using the same color for each clustering as used in the overview statistics and in the parallel partitions plot. Figure 8 shows an example of two such pairs that have different co-cluster signatures suggesting that the relationship between the last two *A. thaliana* proteins is stronger than that of the first pair. Users can sort the rows by either the item name, the number of shared clusterings, or by the co-clustering signature. Users can also filter by the signatures to display only the rows matching a user's pattern.

Reordering the co-cluster matrix

The order of rows and columns in the co-cluster matrix is critical to extracting meaningful information from it. Finding an optimal matrix reordering is NP-complete for almost any interesting objective. Algorithms for the optimal linear arrangement [37] and bandwidth minimization [38] problems have been used to reorder matrices with considerable success; however, both approaches perform

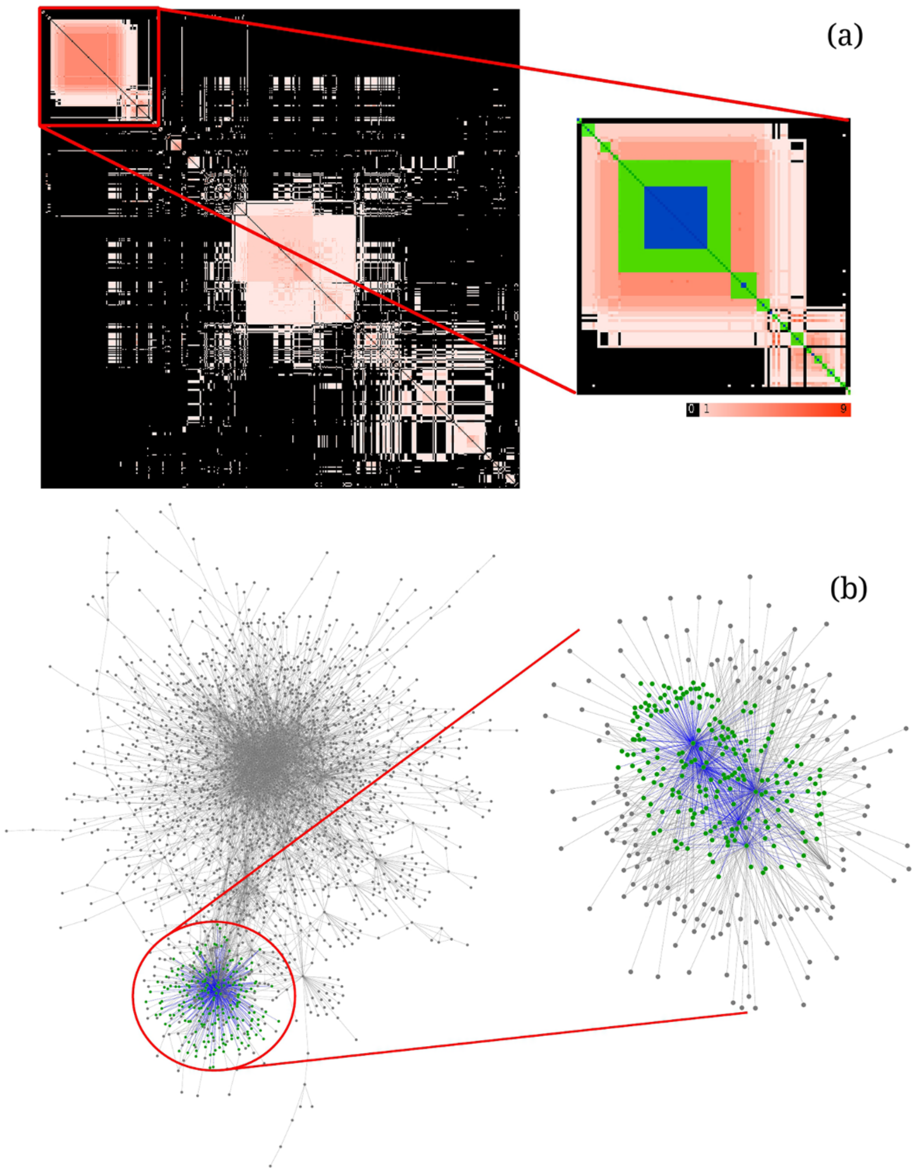


Figure 6 Reordered co-cluster matrix reveals co-clustering patterns. (a) Values in the co-cluster matrix range from 1 (light pink) to 9 (red) for nine clusterings of the *A. thaliana* PPI network with 2402 proteins from [31]. Pink regions represent item pairs that were placed in the same module by very few clusterings while regions of more saturated red represent proteins that co-clustered in most clusterings. Black indicates that the items never co-clustered. On the inset matrix, the matrix items under the green square formed a core. A large blue square overlay suggests that the core was tightly integrated into the rest of the network. **(b)** Left: nodes that formed a core in (a) are colored green, the edges between the nodes within the core are colored blue. The inset to the right shows an isolated view of the core nodes (green), edges between core nodes (blue), and nodes one hop away from the core nodes (gray). Green nodes share many edges with nodes outside of the core which resulted in the core's low cohesion.

poorly for matrices that have many off-diagonal elements. After comparing several reordering algorithms using the bandwidth and envelope metrics, we have chosen the SPIN [39] approach that consistently produced better results on a wide range of matrices.

This approach works as follows: given a matrix A^+, we solve a linear assignment problem (LAP) by mapping A^+'s rows to their optimal positions in the matrix. In other words, given a bipartite graph $G = (R, I, E)$ with R being the set of A's rows, I a set of indices to which the rows will be assigned, and E all possible edges between nodes in R and I, we seek a matching between R and I with in a minimum cost. The edges connecting the row nodes to index nodes are weighted according to how well a row fits a particular index according to a metric that rewards rows that have non-zero entries close to diagonal

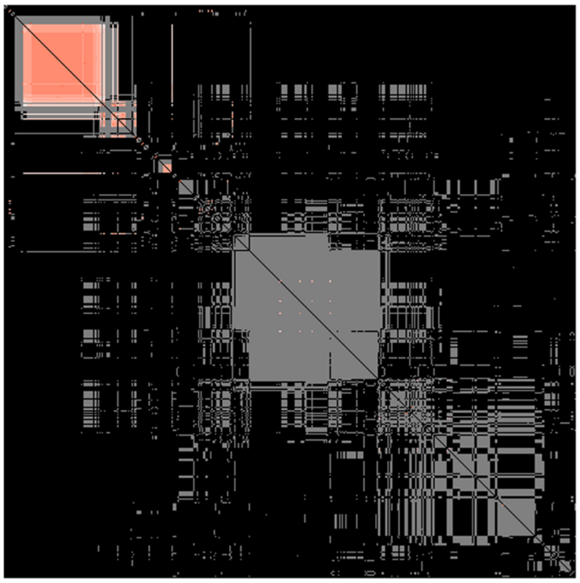


Figure 7 Base clustering mode for the co-cluster matrix. The base clustering highlights only the item pairs that co-clustered within the selected clustering graying out the rest of the matrix. Base clustering helps users focus on comparisons with the selected clustering. In this figure, the colored areas represent the original 26 *A. thaliana* modules; their mostly pink hue indicates that their item pairs co-clustered in few clusterings. Large areas of gray indicate that many novel modules found by other clusterings were not found by the link clustering algorithm [10].

and penalizes those rows that have weight away from diagonal:

$$w(i,\ell) = \sum_{j=1}^{n} a^{+}_{ij}|j-i|,$$

where $w(i,\ell)$ is the weight of assigning i^{th} row to ℓ^{th} position, and the a^{+}_{ij} values are the co-cluster matrix entries. After permuting A^{+}'s rows, the columns of A^{+} must be permuted to match the row order, thus changing the weights $w(i,\ell)$ and making the row assignments found previously no longer optimal, so this process is repeated. In Coral, we use two different solvers for the LAP problem: a fast, greedy solver and the Hungarian algorithm. The greedy solver matches rows to indexes by iteratively selecting the best row-index pair; it quickly finds a starter reordering that can later be improved by the Hungarian algorithm. The Hungarian algorithm solves the linear assignment problem optimally, but because a permutation of rows invalidates the ordering of the columns, the algorithm has to be re-run for several iterations to improve the reordering. We continue iterating LAP until we get no improvement in assignment cost, observe a previously considered permutation, or exceed the maximum number of iterations.

Identifying cores

Given a reordered co-cluster matrix A, we want to find contiguous areas containing high co-cluster values (*cores*). We rely on the notion of region density:

$$d(p,q) = \frac{\sum_{i=p}^{q-1}\sum_{j=i+1}^{q} a^{+}_{ij}}{|q-p|} = \frac{s(p,q)}{|q-p|}, \qquad (1)$$

where a region is a square block on the matrix diagonal between rows p and q, and its density is the sum $s(p,q)$ of all matrix entries within the area divided by the area's width $|q-p|$. Alternatively, we can think of the co-cluster matrix A^{+} as a weighted adjacency matrix of some graph $G(A^{+})$, then $d(p,q)$ is the density of a subgraph S induced by the vertices $p,\ldots,q$: $d(p,q) = |E(S)|/|V(S)|$, where $|E(S)|$ is the sum of edge weights in S and $V(S)$ is a set vertices in S [33].

To find cores, we want to find areas on the diagonal such that the sum of their densities is highest. We do not allow the identified cores to overlap (thus we require disjoint subgraphs). We formulate the problem of finding maximally dense arrangement of areas as a dynamic program with the recurrence:

$$D_{\text{opt}}(j) = \max_{1\le i<j}\{D_{\text{opt}}(i-1) + d(i,j)\}.$$

where $D_{\text{opt}}(j)$ is the optimal area arrangement between 0^{th} and j^{th} item, and $D_{\text{opt}}(n)$ gives the optimal partition of a matrix A^{+} into cores.

Assuming that densities $d(p,q)$ are precomputed and require only a constant time to look up, the dynamic program above takes $O(n^2)$ time (for each i, we solve at most n subproblems, and i ranges from 1 to n). However, a brute force algorithm for computing the block sums $s(p,q)$ (and, hence, the densities) in equation 1 must iterate through

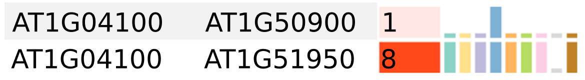

AT1G04100	AT1G50900	1
AT1G04100	AT1G51950	8

Figure 8 Co-cluster signatures help track where two items have co-clustered. Two rows from the pairs table for the *A. thaliana* dataset: each row starts with the two item IDs (here: *A. thaliana* proteins), followed by the number of times these two proteins were co-clustered, followed by a co-cluster signature that tells in which clusterings the two proteins were co-clustered. Clusterings order for this example: *A. thaliana*, `Clauset`, `CFinder`, `Louvain`, `MCL`, `MCODE.F`, `MCODE`, `MINE`, `SPICi`. Proteins AT1G04100 and AT1G51950 co-clustered in 8 clusterings. The two share many specific GO annotations: both are involved in auxin stimulus, localize to the nucleus, and participate in protein binding and sequence-specific DNA binding transcription factor activity. AT1G04100 and AT1G50900 were in the same module just once and shared no GO annotations suggesting that the relationship between these two proteins was of a different nature.

every pair $1 \leq p < n, p < q \leq n$, each time computing a sum of $\binom{|q-p+1|}{2}$ entries, resulting in a runtime of $O(n^4)$. This can be improved because the sums are related. We have:

$$s(p, q+1) = s(p, q) + \sum_{i=p}^{q} a_{i,q+1},$$

making it possible to compute all $s(p, q)$ in $O(n^2)$ time. This reduces the total runtime to find cores to $O(n^2 + n^2) = O(n^2)$.

The algorithm finds a series of areas of varying size and density. Some areas are of no interest and were included in the series only because every block contributes a non-zero score to the total sum. To focus on the meaningful regions only, we filter out the cores with density less than the average density. To calculate the average density for a region $p, \ldots, q$, we first compute an average cell value for A^+:

$$w_{\text{avg}} = \frac{s(1, n)}{\bar{z}},$$

where $\bar{z}$ is the number of non-zero cells in A^+. We then define a probability of an edge existing in a graph induced by A^+:

$$P(e) = \frac{\bar{z}}{\binom{n-1}{2}}.$$

Then, for a given ordering of the matrix A^+, let S be a subgraph induced by vertices $p, \ldots, q$. Then $h_{pq} = |q - p + 1|$ is the number of vertices in S and $\binom{h_{pq}}{2}$ is the maximum number of edges S can possibly have. For this block, the expected block density would be:

$$d_{\text{avg}}(p, q) = \frac{w_{\text{avg}} P(e) \binom{h_{pq}}{2}}{h_{pq}} = \frac{s(p, q)}{\bar{z}} \frac{\bar{z}}{\binom{n-1}{2}} \frac{\binom{h_{pq}}{2}}{h_{pq}}$$
$$= \frac{\binom{h_{pq}}{2} s(p, q)}{h_{pq} \binom{n-1}{2}}.$$

The areas that have density higher than their $d_{\text{avg}}(p, q)$ represent groups of data items that have co-clustered together more often than is expected by chance. Hence, Coral displays only these cores.

Ordering in parallel partitions

When ordering clustering bands in the parallel partitions plot, we would like to put similar clusterings next to each other and avoid putting two dissimilar clusterings vertically adjacent. The intuition for such a constraint is that if the two clusterings K_i and K_{i+1} share many similarities, the bands connecting individual items between the clusterings will only cross a few times making it easier to track module changes. We also need to order items within the bands in a way that puts items from the same module next to each other and does not allow items from other modules to interleave.

To find a vertical order for the clustering bands, we apply a greedy algorithm that uses clustering similarity scores. First, we compute the similarity for every pair of clusterings $\text{sim}(K_i, K_j)$ using Jaccard. Next, we find the two most similar clusterings K_1,K_2, add them to a list, and look for a clustering most similar to either K_1 or K_2 (whichever is greater). We proceed by repeatedly picking the clustering that is most similar to the last clustering added to the list. The order in which clusterings were added to the list determines the order of the clustering bands.

We pursue two objectives when ordering items and modules within a single clustering band: items that belong to the same module must be placed next to each other, and the ordering has to be similar to the column ordering in the co-cluster matrix (so as to maintain the user's mental mapping between the two visualizations). To preserve the matrix ordering in clustering bands, each module is best placed in a position where most of its items are close to the matrix columns corresponding to those items. However, the order of the columns in the matrix may be such that two items u and v from the same module are far apart in A^+. We propose a heuristic to solve this ordering problem: given an ordering of the columns in the matrix A^+, for each module m_i in clustering $K = \{m_1, \ldots, m_{k_i}\}$ we compute its rank based on how "displaced" items in the module are relative to the positions of the module's items in the matrix:

$$d(m_j) = \sum_{u \in m_j} i(u),$$

where $i(u)$ is the index of a column in A^+ corresponding to the data item u. Modules that should be placed first in the clustering band would have the lowest rank, so we sort the modules in order of increasing d, and the module's position in the sorted array determines module's position in the clustering band.

Results

Data

Arabidopsis thaliana is a model organism widely used in plant science, but out of its 35,000 predicted proteins one third still lack an assigned function [40]. A recent publication reports a new protein interaction network for *A. thaliana* that covers a part of the plant's proteome not studied previously [31]. We have selected several clustering algorithms that are often used on PPI networks (Table 1) and, for each of the algorithms, we have generated a clustering of the largest connected component of the *A. thaliana*'s network. To test the resulting modules for robustness, we compare this ensemble of clusterings to the modules reported by the authors of [31] who used a link-clustering method by Ahn, Bagrow, and Lehman [10]. Prior to comparison, we filtered the newly generated modules using the same criteria as [31] by removing modules of size smaller than 6 and with partition density < 0. The new modules were tested for GO enrichment with FuncAssoc [41] (see Table 1 for details).

For our comparison, we have focused on the graph clustering algorithms for which the implementations were available (see Table 1). `Louvain` [9] and `Clauset` [7] are two algorithms that search for a network partition with highest modularity [42]. Both tend to find large clusters and usually cover most of the nodes in a network. `CFinder` [8] is a clique percolation method that identifies overlapping communities by continuously rolling cliques of an increasing size. Resulting clusterings usually contain many small modules with a high amount of overlap and cover only a part of the network ignoring graph structures like bridges and stars. `MCL` [5] is a fast, flow-based clustering algorithm that uses random walks to separate high-flow and low-flow parts of the graph. Its modules tend to be small and usually cover most of the input network. `MCODE` [6] algorithm finds modules in biological networks by expanding communities around vertices with high clustering coefficient. "Fluff" and "haircut" options for `MCODE` allow to add singleton nodes connected to the module by just one edge and to remove nodes weakly connected to the module correspondingly. `MINE` [12] is closely related to `MCODE`, but uses a modified weighting scheme for vertices which results in denser, possibly overlapping modules. `SPICi` [11] grows modules around vertices with high weighted degree by greedily adding vertices that increase module's density. The partitions contain many dense modules, but usually cover only a part of the network.

Table 1 Clustering algorithms used on *A. thaliana* network

Algorithm	Proteins	Modules	Enriched
`Louvain` [9]	2369	23	21
`CFinder` [8]	508	666	180
`Clauset` [7]	2313	20	18
`MCL` [5]	844	46	33
`MCODE` [6]	268	20	16
`MCODE.F` [6]	1314	20	19
`MINE` [12]	206	57	29
`SPICi` [11]	259	46	27

Statistics on clusterings of the largest connected component of the *A. thaliana* network. All algorithms were run using their default settings. `MCODE` was run without "haircut" and no "fluff," `MCODE.F` included "fluff." The table reports the number of proteins that were assigned to at least one module, the number of modules after filtering according to procedure used in Vidal et al. [31], and the number of modules FuncAssoc [41] reported as enriched for at least one GO annotation.

Applying Coral to *A. thaliana* clusterings

To get an overview of the data, we review various statistics on the generated clusterings. For the majority of the clusterings, modules that remained after filtering covered only a portion of the network. The two clusterings produced by the modularity-based methods, `Louvain` and `Clauset`, were the only clusterings that included more than 95% of all proteins into their modules. The number of modules per clustering varied significantly from 20 to 666 (Table 1). The average module size was highest for `Clauset` (115.65), `Louvaine` (103.00), and the `MCODE.F` (82.05) clusterings significantly exceeding the average module size among all other clusterings (3.02-26.31 items per module). For the original 26 *A. thaliana* modules [31], 3% of the proteins were assigned to more than one module; in the `CFinder` clustering over half of the clustered proteins (59%) participated in multiple modules.

The nine *A. thaliana* clusterings are highly dissimilar: most cells in the ladder widget (Figure 2) are white or pale blue, and the majority of pairwise Jaccard similarity scores are below 0.07. `MCL` yielded the partition most similar to *A. thaliana* modules reported in [31] (`A.Thal original`) with Jaccard similarity of 0.60. Surprisingly, the 26 modules generated by link clustering [31] shared very little similarity with `CFinder`, the only other algorithm in the ensemble designed to produce overlapping modules.

Low pairwise similarity scores between so many pairs of clusterings is easily explained using the module-to-module table: clusterings with Jaccard similarity below 0.07 overlap by a few small modules or no modules at all. The similarity of 0.60 between `MCL` and `A.Thal` (Figure 4) may be attributed to the two big modules that are largely replicated between the two clusterings: the module `m9` from `MCL` and the module `m0` from *A. thaliana* (highlighted row) overlap by 288 proteins with Jaccard similarity 0.8. Several smaller modules (shown at the top of the table) are duplicated exactly between the two clusterings.

The co-clustering matrix for *A. thaliana* clusterings contains several large regions of co-clustered proteins along the diagonal (Figure 6), however, most cells are pale indicating that they were co-clustered by only a few clustering algorithms; very few matrix cells are close to the saturated red. Indeed, 65.25% of all co-clustered pairs of *A. thaliana* proteins have co-clustered just once across all of the nine clusterings used in the analysis and only 6.34% of protein pairs were co-clustered in 5 or more partitions. This low number of protein pairs that were assigned to the same cluster means that the clusterings in the ensemble mostly disagreed.

The dynamic program for identifying cores found 249 areas in the *A. thaliana* network in which proteins co-clustered more often than could be expected by chance, with the largest core containing 215 proteins and with the average number of proteins per core of 10.38 proteins. Most cores, including the largest core, had low cohesion values indicating that the proteins forming the cores had many connections to proteins outside of the cores (see Figure 6). This finding is correlated with the fact that the clusterings did not agree in general and only small sets of proteins were consistently clustered together across the ensemble.

Finally, setting `A.thal original` to be the base clustering shows that the modules found by [31] covered only a fraction of modules found by other methods, although they included the largest core. The majority of `A.thal original` modules were colored pale pink indicating that modules found by the link clustering were found by no more than 3 other clustering methods. We trace the largest core in the parallel partitions plot (Figure 4): the proteins in the core are co-clustered by `A.thal original`, `Clauset`, `Louvaine`, `MCL`, and `MCODE.F` while `SPICi`, `MINE`, and `MCODE` ignored the majority of core's proteins completely. `CFinder`, with its many overlapping modules of size 3, 4, and 5, clusters some of the core's proteins and puts a large part of the core in the grab bag group representing unclustered proteins.

Discussion

Clustering algorithms may generate wildly varying clusterings for the same data: algorithms optimize for different objectives, may use different tie breaking techniques, or only cluster part of the data items. A popular technique for optimizing modularity has been shown to suffer from a resolution limit [43] and multiple decompositions may have the same modularity value [14]. When a true decomposition is available, the clustering quality can be quantified using the similarity score and the true positive and true negative rates. However, when there is no true clustering, it is hard to decide which clustering is better than the others. We propose that researchers generate several clusterings by either using different clustering algorithms or varying algorithm parameters. Coral can help compare such clusterings and identify cores in which the data items co-cluster across multiple clusterings.

Most views and statistics in Coral work for both non-overlapping and overlapping clusterings. All overview statistics extrapolate well for overlapping modules except for entropy which assumes that no two modules overlap and therefore may overestimate the actual entropy. The co-cluster matrix naturally adapts to overlapping modules by allowing their corresponding blocks to overlap. Currently, if a pair of data items co-occur in more than one module within a single clustering, their co-cluster value is set to 1 and is not weighted higher relative to other pairs. The parallel partitions plot assumes that the modules in individual clusterings do not overlap. However, if there are overlapping modules, parallel partitions will still lay out the modules in a line by duplicating the co-occurring element in every module in which it occurs.

Although the examples we use in this paper are based on network clustering, Coral does not require its input data to be a network partition and can be used with equal success on gene expression or classification data. In particular, if users would like to compare several classification results, they can do so in the same manner as we have demonstrated for the *A. thaliana* clusterings. The similarity measures of purity, inverse purity, and the F-measure implemented in Coral are helpful in comparing classifications to the available truth. The module-to-module table is a more flexible alternative to the confusion matrix that is often used to evaluate classification results.

Coral has been used to analyze clusterings of up to 4115 items. The runtime varies with the number of clusterings, number of modules and data items per clustering, and the size of the modules. The startup operations — parsing the input clusterings, computing dataset statistics and all-to-all clustering similarities, as well as rendering the views — take from under a second to 11 seconds for clusterings from 33 to 4115 data items. Matrix reordering is the single biggest performance bottleneck for Coral. Reordering the co-cluster matrix for 2376 *A. thaliana* proteins took, on average, 29 seconds when to reorder using the greedy heuristic and 70 seconds when to reorder using the Hungarian algorithm. However, both the greedy heuristic and the Hungarian algorithm find good orderings after very few iterations and the reordering only needs to be computed once before analysis. Solutions for LAP computed with the Hungarian algorithm improve with every iteration and usually converge on a good reordering fast.

Conclusions

Coral offers a comprehensive array of visualizations that allow users to investigate modules from various viewpoints inlcuding several novel views. Coral guides users from overview statistics implemented as familiar bar charts to detailed cluster-to-cluster comparison in a table. The ladder widget, a lower triangle of the comparison matrix, helps users pick the most similar (or dissimilar) pair of clusterings and to judge how similar clusterings in the dataset are overall. A color-coded co-cluster matrix shows how often any pair of items in the dataset have been placed in a module together. A novel adaptation of parallel coordinates, parallel partitions plot, makes tracking a group of items across clusterings easy with intuitive selection techniques. These views combined create a powerful tool for a comprehensive exploration of an ensemble of clusterings. Coral can help users generate questions and hypotheses about the data that could be later definitively answered with the help of additional experiments.

Availability and requirements

- **Project name:** Coral
- **Project home page:** http://cbcb.umd.edu/kingsford-group/coral/
- **Operating systems:** platform-independent
- **Programming language:** Java
- **Other requirements:** Java 1.6 and 1Gb of RAM
- **License:** GNU GPL

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
CK and DF conceived the project, DF is the designer and developer for the application. CK provided design considerations and worked on developing some of the algorithms. AG implemented reordering algorithms for co-cluster matrix. DF and CK contributed to final manuscript. All authors read and approved the final manuscript.

Acknowledgements
This work was partially supported by NSF grants EF-0849899, IIS-0812111, and CCF-1053918 and NIH grant 1R21AI085376 to C.K. The authors thank Megan Riordan for helpful programming contributions, and Geet Duggal and Rob Patro for their helpful insights on matrix reordering.

Author details
[1]Lane Center for Computational Biology, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. [2]Center for Bioinformatics and Computational Biology, University of Maryland, College Park, MD, USA. [3]Department of Computer Science, University of Maryland, College Park, MD, USA.

Received: 6 April 2012 Accepted: 9 October 2012
Published: 29 October 2012

References

1. Sharan R, Ulitsky I: **Network-based prediction of protein function.** *Mol Syst Biol* 2007, **3:**88.
2. Ulitsky I, Maron-Katz A, Shavit S, Sagir D, Linhart C, Elkon R, Tanay A, Sharan R, Shiloh Y, Shamir R: **Expander: from expression microarrays to networks and functions.** *Nat Protoc* 2010, **5**(2):303–322.
3. Chatterji S, Yamazaki I, Bai Z, Eisen JA: **CompostBin: A DNA composition-based algorithm for binning environmental shotgun reads.** Tech. rep., arXiv 2007.
4. White JR, Navlakha S, Nagarajan N, Ghodsi MR, Kingsford C, Pop M: **Alignment and clustering of phylogenetic markers — implications for microbial diversity studies.** *BMC Bioinf* 2010, **11:**152.
5. van Dongen S: **Graph clustering by flow simulation** *PhD thesis*, University of Utrecht; 2000.
6. Bader G, Hogue C: **An automated method for finding molecular complexes in large protein interaction networks.** *BMC Bioinf* 2003, **4:**2.
7. Clauset A, Newman MEJ, Moore C: **Finding community structure in very large networks.** *Physical Rev E* 2004, **70**(066111):6.
8. Adamcsek B, Palla G, Farkas IJ, Derényi I, Vicsek T: **CFinder: locating cliques and overlapping modules in biological networks.** *Bioinformatics* 2006, **22**(8):1021–1023.
9. Blondel VD, Guillaume JL, Lambiotte R, Lefebre E: **Fast unfolding of communities in large networks.** *J Stat Mech: Theory and Experiment* 2008, **2008**(10):P10008.
10. Ahn YY, Bagrow JP, Lehmann S: **Link communities reveal multiscale complexity in networks.** *Nat Lett* 2010, **466**(5):761–765.
11. Jiang P, Singh M: **SPICi: a fast clustering algorithm for large biological networks.** *Bioinformatics* 2010, **26**(8):1105–1111.
12. Rhrissorrakrai K, Gunsalus KC: **MINE: module identification in networks.** *BMC Bioinformatics* 2011, **12**(192):1–10.
13. Navlakha S, Kingsford C: **Exploring biological network dynamics with ensembles of graph partitions.** In *Proceedings of the 15th Pacific Symposium on Biocomputing (PSB)*. Hawaii; 2010:166–177 .
14. Duggal G, Navlakha S, Girvan M, Kingsford C: **Uncovering many views of biological networks using ensembles of near-optimal partitions.** In *Proceedings of the 1st International Workshop on Discovering, Summarizing and Using Multiple Clusterings (KDD MultiClust)*. Washington, DC; 2010:9.
15. Lewis AC, Jones NS, Porter MA, Charlotte DM: **The function of communities in protein interaction networks at multiple scales.** *BMC Syst Biol* 2010, **4:**100.
16. Langfelder P, Zhang B, Horvath S: **Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R.** *Bioinformatics* 2008, **24**(5):719–720.
17. Hopcroft J, Khan O, Kulis B, Selman B: **Tracking evolving communities in large linked networks.** *Proc Nat Acad Sci USA* 2004, **101**(Suppl 1):5249–5253.
18. Yu Z, Wong HS, Wang H: **Graph-based consensus clustering for class discovery from gene expression data.** *Bioinformatics* 2007, **23**(21):2888–2896.
19. Hibbs MA, Dirksen NC, Li K, Troyanskaya OG: **Visualization methods for statistical analysis of microarray clusters.** *BMC Bioinf* 2005, **6:**115.
20. Thalamuthu A, Mukhopadhyay I, Zheng X, Tseng GC: **Evaluation and comparison of gene clustering methods in microarray analysis.** *Bioinformatics* 2006, **22**(19):2405–2412.
21. Seo J, Gordish-Dressman H: **Exploratory data analysis with categorical variables: an improved rank-by-feature framework and a case study.** *Int J Human-Comput Interact* 2007, **23**(3):287–314.
22. Laderas T, McWeeney S: **Consensus framework for exploring microarray data using multiple clustering Methods.** *OMICS: J Integr Biol* 2007, **11:**116–128.
23. Strehl A, Ghosh J: **Cluster ensembles — A knowledge reuse framework for combining multiple partitions.** *J Machine Learning Res* 2002, **3:**583–617.
24. Monti S, Tamayo P, Mesirov J, Golub T: **Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data.** *Machine Learning* 2003, **52:**91–118.
25. Grimmer J, King G: **General purpose computer-assisted clustering and conceptualization.** *Proc Nat Acad Sci USA* 2011, **108**(7):2643–2650.
26. Langfelder P, Luo R, Oldham MC, Horvath S: **Is my network module preserved and reproducible?** *PLoS Comput Biol* 2011, **7:**e1001057.
27. Kosara R, Bendix F, Hauser H: **Parallel sets: interactive exploration and visual analysis of categorical data.** *IEEE Trans Visualization Comput Graphics* 2006, **12**(4):558–568.
28. Shneiderman B: **The eyes have it: a task by data type taxonomy for information visualizations.** In *Proceedings 1996 IEEE Symposium on Visual Languages*. Boulder, CO: IEEE Comput Soc Press; 1996:336–343.
29. North C, Shneiderman B: **Snap-together visualization: a user interface for coordinating visualizations via relational schemata.** In *Proceedings of the Working Conference on Advanced Visual Interfaces - AVI '00*. New York: ACM Press; 2000:128–135.
30. Meilă M: **Comparing clusterings by the variation of information.** In *Proceedings of the 16th Conference on, Learning Theory and Kernel Machines: 7th Kernel Workshop*. Edited by Schölkopf B, Warmuth MK. Berlin: Springer-Verlag; 2003:173–187. Lecture, Notes in Computer Science, vol. 2777.
31. Arabidopsis Interactome Mapping Consortium: **Evidence for Network Evolution in an Arabidopsis Interactome Map.** *Science* 2011, **333**(6042):601–607.
32. Luo F, Li B, Wan XF, Scheuermann RH: **Core and periphery structures in protein interaction networks.** *BMC Bioinf* 2009, **10**(Suppl 4):S8.
33. Saha B, Hoch A, Khuller S, Raschid L, Zhang X: **Dense subgraphs with restrictions and applications to gene annotation graphs.** In *Proceedings of the 14th Annual international conference on Research in Computational Molecular Biology (RECOMB'10)*. Edited by Berger B. Berlin: Springer-Verlag; 2010:456–472. Lecture Notes in Computer Science, vol. 6044.
34. Bailey TA, Dubes RC: **Cluster validity profiles.** *Pattern Recognit* 1982, **15**(2):61–83.
35. Mewes HW, Ruepp A, Theis F, Rattei T, Walter M, Frishman D, Suhre K, Spannagl M, Mayer KFX, Stümpflen V, Antonov A: **MIPS: curated**

databases and comprehensive secondary data resources in 2010. *Nucleic Acids Res* 2011, **39**(Database issue):D220—D224.

36. Gama-Castro S, Salgado H, Peralta-Gil M, Santos-Zavaleta A, Muñiz Rascado L, Solano-Lira H, Jimenez-Jacinto V, Weiss V, García-Sotelo JS, López-Fuentes A, Porrón-Sotelo L, Alquicira-Hernández S, Medina-Rivera A, Martínez-Flores I, Alquicira-Hernández K, Martínez-Adame R, Bonavides-Martínez C, Miranda-Ríos J, Huerta AM, Mendoza-Vargas A, Collado-Torres L, Taboada B, Vega-Alvarado L, Olvera M, Olvera L, Grande R, Morett E, Collado-Vides J: **RegulonDB version 7.0: transcriptional regulation of Escherichia coli K-12 integrated within genetic sensory response units (Gensor Units).** *Nucleic Acids Res* 2011, **39**(Database issue):D98—D105.
37. Mueller C, Martin B, Lumsdaine A: **A comparison of vertex ordering algorithms for large graph visualization.** In *Proceedings of the 6th International Asia-Pacicifc Symposium on Visualization (APVIS'07)*. Edited by Hong S-H, Ma K-L. Piscataway: IEEE; 2007:141–148.
38. Lai Y, Williams K: **A survey of solved problems and applications on bandwidth, edgesum, and profile of graphs.** *J Graph Theory* 1999, **31**(2):75–94.
39. Tsafrir D, Tsafrir I, Ein-Dor L, Zuk O, Notterman DA, Domany E: **Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices.** *Bioinformatics* 2005, **21**(10):2301–2308.
40. Kerrien S, Aranda B, Breuza L, Bridge A, Broackes-Carter F, Chen C, Duesbury M, Dumousseau M, Feuermann M, Hinz U, Jandrasits C, Jimenez RC, Khadake J, Mahadevan U, Masson P, Pedruzzi I, Pfeiffenberger E, Porras P, Raghunath A, Roechert B, Orchard S, Hermjakob H: **The IntAct molecular interaction database in 2012.** *Nucleic Acids Res* 2011, **40**(D1):D841—D846.
41. Berriz GF, Beaver JE, Cenik C, Tasan M, Roth FP: **Next generation software for functional trend analysis.** *Bioinformatics* 2009, **25**(22):3043–3044.
42. Newman MEJ: **Modularity and community structure in networks.** *Proc Nat Acad Sci USA* 2006, **103**(23):8577–8582.
43. Fortunato S, Barthelemy M: **Resolution limit in community detection.** *Proc Nat Acad Sci USA* 2007, **104:**36–411.

doi:10.1186/1471-2105-13-276
Cite this article as: Fillippova *et al.*: Coral: an integrated suite of visualizations for comparing clusterings. *BMC Bioinformatics* 2012 **13**:276.