OPEN ACCESS Freely available online

PLoS one

Jerarca: Efficient Analysis of Complex Networks Using Hierarchical Clustering

Rodrigo Aldecoa, Ignacio Marín*

Instituto de Biomedicina de Valencia, Consejo Superior de Investigaciones Científicas (IBV-CSIC), Valencia, Spain

Abstract

Background: How to extract useful information from complex biological networks is a major goal in many fields, especially in genomics and proteomics. We have shown in several works that iterative hierarchical clustering, as implemented in the UVCluster program, is a powerful tool to analyze many of those networks. However, the amount of computation time required to perform UVCluster analyses imposed significant limitations to its use.

Methodology/Principal Findings: We describe the suite Jerarca, designed to efficiently convert networks of interacting units into dendrograms by means of iterative hierarchical clustering. Jerarca is divided into three main sections. First, weighted distances among units are computed using up to three different approaches: a more efficient version of UVCluster and two new, related algorithms called RCluster and SCluster. Second, Jerarca builds dendrograms based on those distances, using well-known phylogenetic algorithms, such as UPGMA or Neighbor-Joining. Finally, Jerarca provides optimal partitions of the trees using statistical criteria based on the distribution of intra- and intercluster connections. Outputs compatible with the phylogenetic software MEGA and the Cytoscape package are generated, allowing the results to be easily visualized.

Conclusions/Significance: The four main advantages of Jerarca in respect to UVCluster are: 1) Improved speed of a novel UVCluster algorithm; 2) Additional, alternative strategies to perform iterative hierarchical clustering; 3) Automatic evaluation of the hierarchical trees to obtain optimal partitions; and, 4) Outputs compatible with popular software such as MEGA and Cytoscape.

Citation: Aldecoa R, Marín I (2010) Jerarca: Efficient Analysis of Complex Networks Using Hierarchical Clustering. PLoS ONE 5(7): e11585. doi:10.1371/journal.pone.0011585

Editor: Carl Kingsford, University of Maryland, United States of America

Received April 29, 2010; **Accepted** June 20, 2010; **Published** July 14, 2010

Funding: This research was supported by grant BIO2008-05067 (Programa Nacional de Biotecnologia; Ministerio de Ciencia e Innovacion, Spain). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

* E-mail: imarin@ibv.csic.es

Introduction

There are many types of data, both biological and non-biological, which can be represented as undirected graphs. Examples in biology are networks based on protein-protein interaction data, those based on shared protein domains, genetic interaction networks or coexpression networks. Developing heuristic strategies to extract useful information from them is an active field of research (reviewed in [1–3]). A typical problem is how to generate partitions of a network in order to establish clusters, groups of tightly connected units. There are two basic general strategies to perform such a task. One option is to search for densely connected modules, for instance using a local evaluation function that measures when adding or eliminating units leads to a significant decrease of the average density of connections within a group (see e. g. refs. [4–9]). A second possibility is to generate complete partitions of the graph, assigning each unit to a cluster. This requires global parameters to evaluate the quality of the alternative partitions [10–12]. Although both methods have advantages and drawbacks, the latter should be considered preferable on theoretical grounds, given that it allows classifying all the units of the network.

To classify data, hierarchical clustering has several advantages over other procedures. First, it is a fully unsupervised method. In the case of networks, this allows to cluster all units without having to specify a priori the number of clusters present. In addition, the generation of a hierarchical tree provides not only partitions of the network (either by how units are grouped in agglomerative clustering, or by how the units are divided into groups, in divisive clustering), but also allows to visualize how the basic, first-order clusters are combined into higher-level groups. However, the development of hierarchical clustering strategies to analyze networks is problematic. Particularly, clustering unweighted undirected graphs (e. g. networks of interacting units) is seriously hampered by the ''ties in proximity'' problem (discussed in [12]). In this type of networks, the distance between two units is defined as the minimal number of edges that must be walked to connect them. Then, in typical biological networks – large and with small-world properties – the number of tied distances is astronomical. This makes it impossible to directly obtain a reasonable hierarchical tree based on the distances among units. The problem caused by the ties is that in each step of the clustering process a large number of alternative agglomerations (or divisions) are possible. Several authors attempted to solve this problem by using

measures of proximity among units different from their distances [13–15]. However, to justify the usage of any of these alternative parameters is difficult. A few years ago, we devised a valid strategy to solve the ties in proximity problem [12]. The first step consists in generating a large number of alternative, mathematically equivalent partitions of the network using the distances among the units (*primary distances*, according to our nomenclature) and conventional (e. g. average linkage) hierarchical clustering. The results are then averaged to obtain a weighted distance measure for each pair of units (*secondary distance*). This distance corresponds to the fraction of alternative partitions in which two units are assigned to different clusters. Finally, a dendrogram is obtained from the matrix of secondary distances. This strategy, which we called iterative cluster analysis, has already empirically demonstrated its usefulness. High-quality dendrograms have been obtained from complex networks derived from different types of biological data [16–18]. However, performing iterative hierarchical clustering has been so far hampered by the intrinsic slowness of obtaining a representative set of partitions. For example, our original program, UVCluster [12], runs in $O(n^3)$ time, n being the number of nodes. For this reason, the largest analysis published so far corresponds to a network with just 632 units [18].

In this work, we describe a suite of programs called Jerarca (Spanish for hierarch), which contains new, efficient algorithms to perform iterative hierarhical cluster analyses. One of them is basically a faster implementation of the UVCluster program. The other two, RCluster and SCluster, provide alternative ways to obtain the matrices of secondary distances from a graph. In addition, for the conversion of the matrix of distances into a dendrogram, two well-known phylogenetic algorithms, UPGMA and Neighbor-Joining [19–21] have been included in Jerarca. Finally, Jerarca also includes two different mathematical criteria to determine the best partition of the dendrogram into clusters. The first one is a parameter called modularity (Q) [10], which has been extensively used to measure community structure in networks. As an alternative, we include a modification of a hypergeometric distribution-based index suggested in one of our previous works [12]. Several output files, useful to edit and visualize the results, are generated by the program. All these options make Jerarca much more efficient and versatile than our original UVCluster program.

Methods

General features of the program

The Jerarca suite has been written in C++. Both the source code and compiled versions for Windows and Linux platforms are freely available at http://jerarca.sourceforge.net. Figure 1 details the control flow structure of the code. To perform a round of analyses, the user must execute the program from a command window, writing four parameters in the following order: 1) the name of a text file that describes the list of edges of the graph. The names of two linked nodes, separated by a tab or space, must be written in each line of the file; 2) the algorithm(s) chosen to iteratively calculate the matrix(ces) of secondary distances; 3) the algorithm(s) that will be used to obtain the dendrogram; and, 4) the number of iterations to be performed. Therefore, a typical Jerarca input has the following structure (parameters are indicated in brackets):

jerarca [Name of the file] [Name of the iterative algorithm: (*uv*, *r*, *s*, *all*)] [Name of the tree algorithm: (*u*, *nj*, *all*)] [Number of iterations]

For the iterative algorithm, four options are valid: *uv* (UVCluster), *r* (RCluster), *s* (SCluster) and *all*. This last option will produce three parallel solutions, one for each available algorithm. For the tree algorithm, three options are valid: *u* (UPGMA), *nj* (Neighbor-joining) and *all*. This last option again will produce two solutions, one for each algorithm.

A typical Jerarca analysis is shown in Figure 2. In summary, the program reads the input file and creates the adjacency matrix A of the graph: $A_{ij} = 1$ if vertices i and j are connected and $A_{ij} = 0$ otherwise. Then, it applies the iterative algorithm(s) selected as many times as the number of iterations specified. To calculate the matrix of secondary distances, the algorithm saves, for each pair of nodes, the number of iterations in which they have been clustered separately, and the secondary distances between each two units are calculated by dividing those values by the number of iterations. After creating the matrix of secondary distances, the program uses the phylogenetic algorithm(s) chosen to build a dendrogram. The program finally evaluates, using the two indices implemented, each level of the dendrogram and saves the optimal partition of the tree for each index (see below). Several convenient output files (described also in detail below) are generated.

Details of the iterative algorithms

We recently developed several novel ideas that are the basis of Jerarca. We first thought a way to notably improve the speed of the UVCluster program. UVCluster contained a parameter called *Affinity Coefficient* (*AC*), which sets how permissive the clustering process is, in such a way that the lower the *AC* value, the larger the average distances among clustered units can be (see [12] for a detailed explanation). The maximum value of $AC = 100$ implies that only units that are directly connected in the graph are clustered together. Very significantly, this value was the only used in all our subsequent works [16–18]. Not a single useful application for other values has ever been found. This has an important consequence, given that, if we fix $AC = 100$, UVCluster-based iterative hierarchical clustering can be performed using the adjacency matrix of the network instead of the matrix of primary distances. This avoids computing the primary distances among all units using Floyd's algorithm, whose time complexity is $O(n^3)$. Once noticed that important point, we decided to generate a new version of UVCluster implementing this new approach. It turns out that this improved version is qualitatively faster than our former program, running in $O(n^2)$ time.

Two new algorithms, called RCluster and SCluster, described here for the first time, provide alternative ways to establish the matrix of secondary distances, following strategies related to the one implemented in the new version of UVCluster. These programs use alternative methods to select the units to be merged. Figure 3 shows a compact, technical description of their differences. However, we think that the reader may benefit from the following verbal summary of how the three programs work. The differences in the clustering process are as follows:

A) To select which units to merge, UVCluster generates in each iteration a list in which the units are randomly ordered and then proceeds to generate a cluster taking the first unit in that list and searching for all the units that can be merged to that one, according to the provided *AC* parameter. If $AC = 100$ (fixed value in the new version of the program) this means that UVCluster establishes cliques, i. e. groups in which each unit is connected with all the rest of units in the group. Once the largest clique that can be formed from the first selected unit is found, the units of that clique are set apart (i. e. they are considered to form a cluster) and the next unit still available in the list is used to start again the same process. This is a greedy algorithm, which tends to favor finding compact clusters.

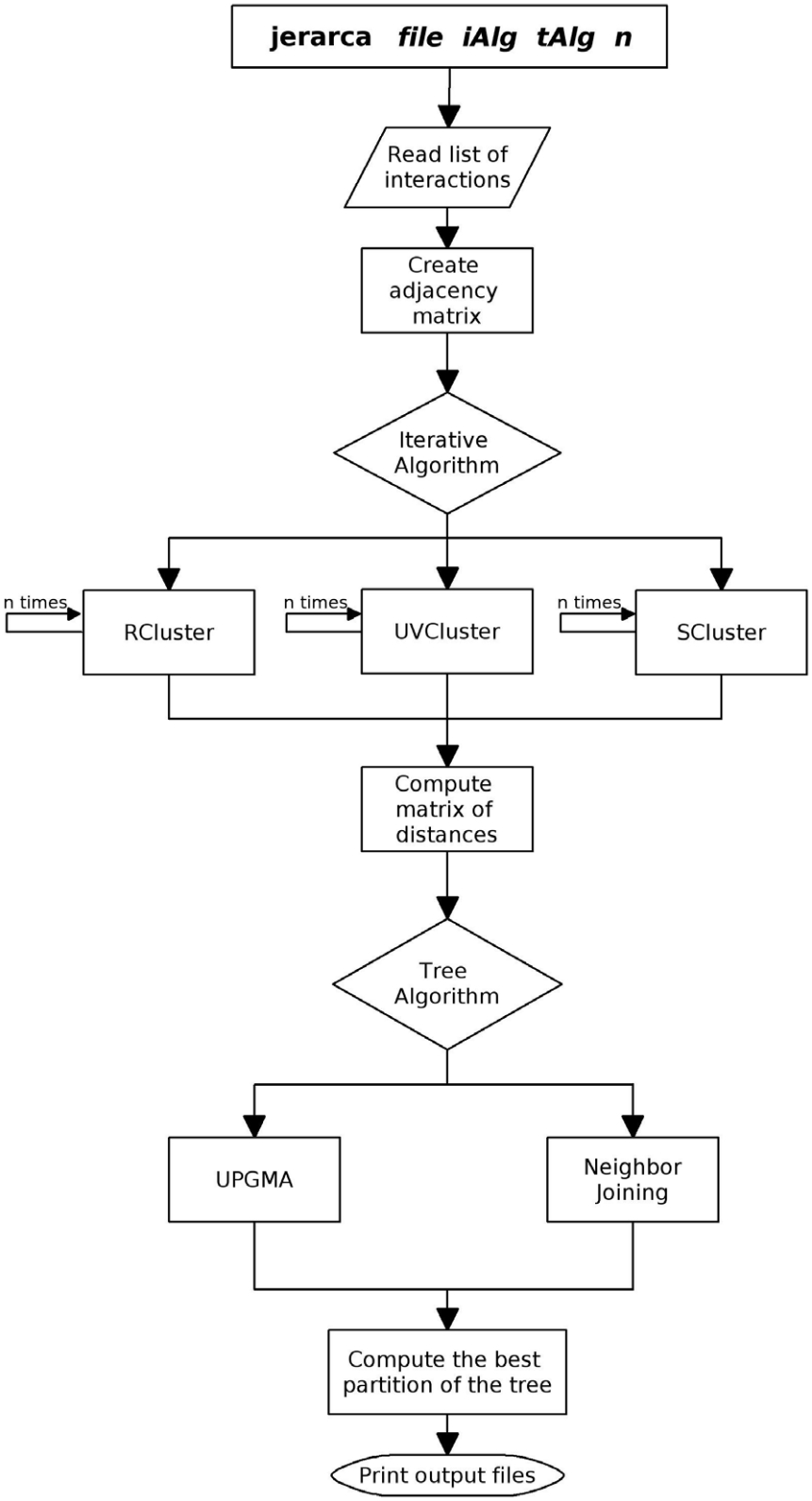

Figure 1. Control flowchart of Jerarca. The four input parameters are *file* (list of interactions that represent the edges of the network), *iAlg* (iterative algorithm to use), *tAlg* (tree algorithm to use) and *n* (number of iterations to perform).
doi:10.1371/journal.pone.0011585.g001

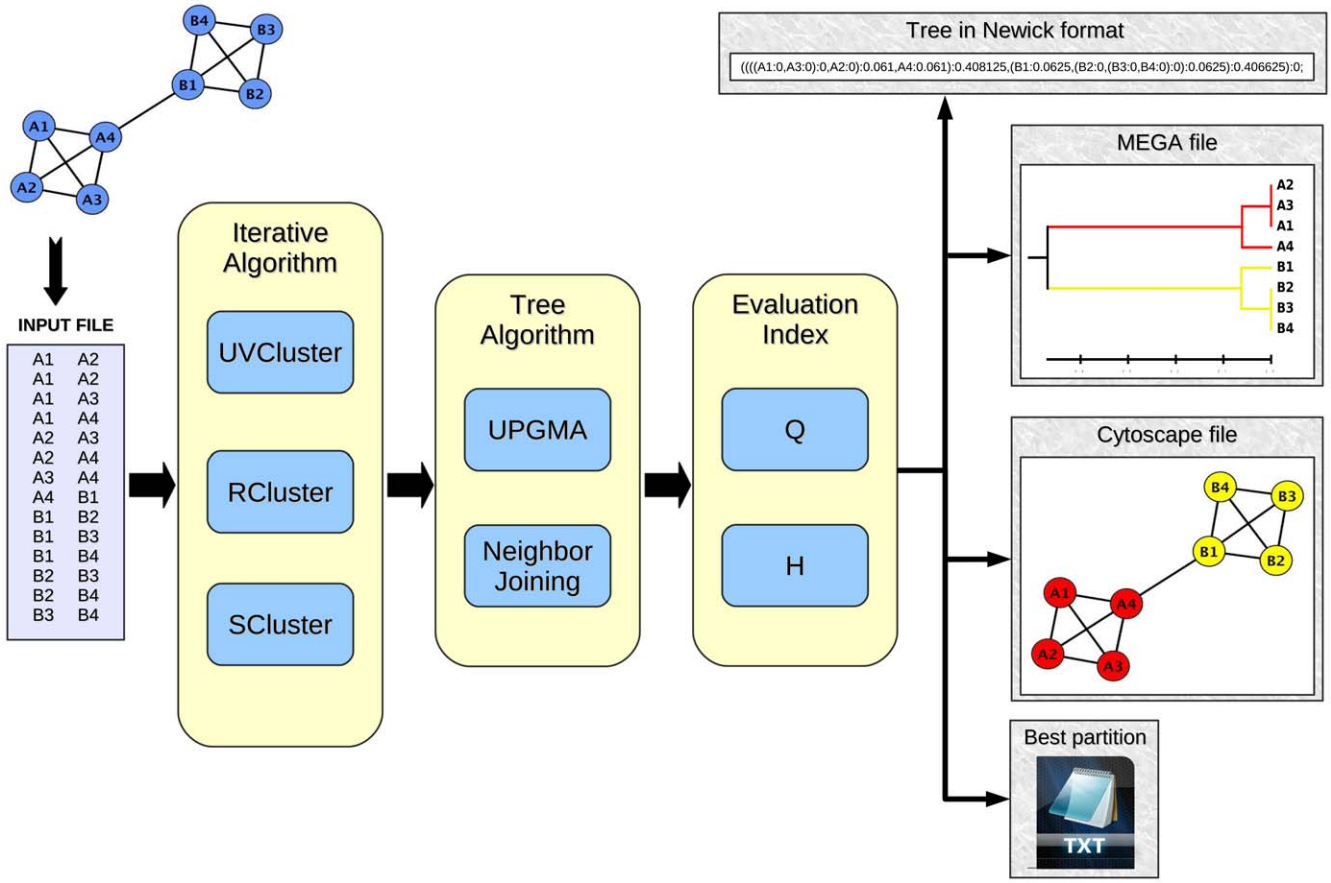

Figure 2. A typical analysis with Jerarca. The user specifies the input file where the graph is represented. It is analyzed by the program through diverse algorithms returning four different outputs: the tree in Newick format, a MEGA-compatible file, a file with attributes for Cytoscape and a text file containing the optimal partition of the tree.
doi:10.1371/journal.pone.0011585.g002

B) Our second algorithm, RCluster (R meaning random), also establishes cliques but, instead of using a starting unit and greedily making a particular cluster to grow from it, RCluster in each step randomly merges two clusters, provided that all their units are connected (i. e. they form, after being merged, a clique). The program follows a hybrid strategy to select the clusters. To start with, the program simply randomly picks up two clusters, establishes whether they can be merged or not and, if indeed it is possible to merge them, puts all the units together into a single, new cluster. While there are many clusters that can be merged, this simple strategy is very efficient and it has the big advantage of not requiring to recalculate the adjacency matrix in each merging step, something that is very time consuming for large graphs. However, as the merging process progresses, the likelihood of finding mergeable clusters just by randomly picking up two of them gets smaller. It is then convenient to shift to a second strategy, which is indeed based on generating in each step of the merging process an adjacency matrix, in which a value $A_{ij} = 1$ means that the units of the two clusters (i, j) form a clique. This second strategy is implemented in two steps: 1) The program generates an adjacency matrix and then randomly searches for a $A_{ij} = 1$ value in that matrix to merge two clusters; 2) It recalculates the adjacency matrix. Logically, for the newly formed cluster, it assigns a value of "1" with another cluster only when all the units in both clusters are connected. These two processes are repeated until no clusters can be merged. The transition from the first to the second strategy occurs when n random picks, n being the number of nodes of the network, have failed to find two mergeable clusters. Empirical analyses have shown this to be a convenient cutoff. Notice that, in RCluster, and differently from what occurs in UVCluster, multiple clusters grow at the same time. However, the process of choosing a random pair of clusters to merge in each iteration makes the program slower than the current version of UVCluster. We found that it runs in $O(n^2 \log n)$ time.

C) Finally, the third alternative is our novel SCluster algorithm (S stands for simple), which is both our greediest and our fastest algorithm, running in $O(n \log n)$ time. SCluster just picks up a unit by random and then collapses in a cluster that unit with all the units directly connected to it. These units are removed from the graph and then another unit is randomly chosen and the process is repeated until no further units remain. Notice the difference with UVCluster and RCluster: the units collapsed in a cluster do not have to be all connected among them (forming cliques) but just linked to the initial unit.

Dendrogram algorithms and evaluation of the partitions

Using any/all the algorithms described above, a matrix of secondary distances is obtained from which dendrograms can be

UVCluster	RCluster	SCluster

create the list of nodes
WHILE the list of nodes is not empty:
  create a new cluster C
  select a random node N
  add N to C
  remove N from the list of nodes
  WHILE possible:
  select a node N' connected to every node in C
  add N' to C
  remove N' from the list of nodes
  END WHILE
END WHILE
	assign each node Ni to a cluster Ci
	n = 0
	WHILE n < number_of_nodes:
	  n = n + 1
	  randomly select two clusters Ci, Cj
	  IF every node in Ci is connected to every node in Cj
	  merge Ci and Cj
	  n = 0
	  END IF
	END WHILE
	
	create matrix M (num_of_clusters x num_of_clusters)
	FOREACH pair of clusters Ci, Cj:
	  IF every node in Ci is connected to every node in Cj
	  M(Ci, Cj) = 1
	  ELSE
	  M(Ci, Cj) = 0
	  END IF
	END FOREACH
	
	WHILE exists any M(Ci, Cj) = 1
	  choose a random pair or clusters Ci, Cj for which M(Ci, Cj) = 1
	  merge Ci and Cj
	  remove Ci and Cj from M
	  add the new formed cluster to M
	  recalculate M
	END WHILE
		create the list of nodes
		WHILE the list of nodes is not empty:
		  create a new cluster C
		  select a random node N
		  add N to C
		  add to C every node N' connected to N
		  remove every node in C from the list of nodes
		END WHILE

Figure 3. Main loop of the three iterative clustering algorithms implemented in Jerarca. An iteration defines a partition of the network by assigning the nodes to clusters. These loops are repeated as many times as iterations are specified by the user.
doi:10.1371/journal.pone.0011585.g003

generated. Jerarca implements two well-known phylogenetic algorithms for this task, UPGMA and Neighbor-Joining. The user may run one or both algorithms.

From the dendrogram, partitions of the units into clusters can be obtained. Jerarca establishes partitions by scanning the dendrogram from the root to the external leaves. Starting from the root, each dichotomy in the tree (that increases the number of clusters) generates an alternative partition that can be evaluated. Given that the neighbor-joining method generates unrooted trees, the middle point of the tree is used as root [22]. Jerarca implements two mathematically independent criteria in order to evaluate the community structure of a given partition. The first index is the well-known and broadly used modularity (Q) [10], which measures the distribution of within and between communities links in a certain partition compared to the expected number of connections that should exist given a specific degree distribution [23]. The second index (called H) is based on the cumulative hypergeometric distribution of links, and derives from an index proposed in the paper that described UVCluster [12]. The definition of H is as follows:

$$H = -\log \sum_{j=p}^{Min(M,n)} \frac{\binom{M}{j}\binom{F-M}{n-j}}{\binom{F}{n}},$$

where F is the maximum possible number of direct interactions in the whole network (for a network of k elements, $F = k(k-1)/2$), n is the number of direct interactions actually observed among the k elements of the network, M is the maximum possible number of intracluster direct interactions in a given partition and p is the total number of direct intracluster interactions actually detected in that partition. The parameter H measures the probability of obtaining by chance a given partition assuming a random distribution of intracluster and intercluster connections. The larger the value of H, the better (''more unexpected'') the partition of the tree.

Output files

Jerarca produces four types of output files (Figure 2). Their names, automatically generated, include a reference to the algorithms and the evaluation criterion used (e. g. a typical name would be ''Filename_partitionH_SCluster_Upgma.txt''). Moreover, the extension of a file specifies the content of the output:

1) Files with ''.meg'' extension contain the matrix of distances among units and the clusters obtained in the optimal partition of the dendrogram, according to either Q or H. This file can be directly imported into the software MEGA 4 [24] for further analyses.
2) Files with ''.att'' extension contain the assignment of nodes to clusters in the best partition. These files are designed to be imported into Cytoscape (version 2.x) [25] as attributes of the nodes (from the main Cytoscape menu: File – Import – Node attributes).
3) Files with a ''.txt'' extension save the best partition of the dendrogram obtained in text format. They include a description of the optimal partition: number of clusters, value of the index used and the assignment of nodes to each cluster.
4) Finally, the files with ''.nwk'' extension describe the dendrogram structure in standard Newick format, which

can be read by virtually all programs that analyze trees, such as MEGA.

Results

The speed of the programs has been tested in several benchmarks. Here we describe the results for three of them, consisting in an artificial and two real networks:

Benchmark A

We prepared a synthetic graph of known community structure, in which 512 units were divided into 16 clusters of equal size. Within each cluster all units were initially fully connected (for a total of $(k^2-k)/2$ edges, being k the number of units in a cluster). Then, we progressively "degraded" that structure by removing a certain percentage of edges and then randomly shuffling a number of edges among the units. The networks generated are a variation of the connected-caveman graphs defined by Watts [26].

Benchmark B

The proteins (nodes) that constitute 408 different protein complexes described in the yeast *Saccharomyces cerevisiae* were obtained from the CYC2008 database (http://wodaklab.org/cyc2008; [27]). We then downloaded from the BioGRID database [28] the protein-protein interactions (edges) characterized so far for all these proteins. The final graph contained 1604 nodes and 14171 edges.

Benchmark C

The complete set of protein-protein interactions (interactome) of *S. cerevisiae* was obtained from BioGRID. These data generated a network formed by 5735 nodes (proteins) and 51134 edges (protein-protein interactions).

Benchmark A was specifically created for testing the quality of the optimal partitions computed by the algorithms implemented in Jerarca. We generated networks with progressive percentages of degradation. In this context, a percentage of degradation of, say, 10%, means that first, 10% of links were eliminated and, from the rest, 10% shuffled among units. The shuffling process involves the random removal of an edge of the graph and the later addition of a new edge between two nodes, chosen also randomly. We previously suggested using a number of iterations equal to 10 times the number of units [12]. Thus, for each of those networks, we ran 5000 iterations of Jerarca with the parameter *all* for both the iterative and the tree algorithms. This means that 12 analyses (= 3 iterative algorithms×2 tree algorithms×2 partition criteria) were performed for each network. With 0–30% degradation, all algorithms recovered the original community structure of the network without errors. However, starting at 40% degradation, slight errors in recovering the original community structure of the graph began to emerge, so we focused on this case. For each of the six dendrograms constructed by using the three iterative and the two tree algorithms, the optimal partitions given by the two evaluation indexes implemented in Jerarca (Q and H) were exactly the same. In all cases but one, a single unit of the network, different for each combination of programs, was misclassified. Only the combination of SCluster and UPGMA recovered the exact community structure of the original network. Significantly, this particular combination also obtained the highest Q and H values. This example shows that all the programs efficiently recover the original structure, even when it is quite cryptic (40% degradation means that just about a third of the original links remain). On the other hand, it also shows the advantage of using when possible all the programs together, given that some may perform better than others.

We performed speed tests in a PC-compatible computer with an Intel Core 2 Quad Q8200 at 2.33 GHz and 4 GB of RAM, running Linux. The analyses of benchmark A were very fast. The 12 analyses per network described in the previous paragraph (5000 iterations/analysis) required just between 30 and 75 seconds. The least degraded (= more compact) graphs, allow for the fastest analyses. To test the speed of the program in real networks of larger sizes, we used benchmarks B and C. For benchmark B (1604 nodes), 16000 iterations took about 3.25 hours when using the RCluster algorithm, while for UVCluster and SCluster the cost was 2 minutes and less than a minute respectively. This large difference is due to the fact that this network contains densely connected modules (each protein complex was much more tightly connected internally than with the rest of the network), a feature that favors the greedy strategies implemented in UVCluster and SCluster. For benchmark C (5735 nodes), 60000 iterations took 40 minutes with SCluster and about 3 hours with UVCluster. For RCluster, we estimated the analysis to require around 300 hours, so it was not performed in full.

In summary, the new algorithms implemented in Jerarca make possible to analyze large networks. As the times just detailed demonstrate, a single computer may easily cope with problems involving several thousands of units in a reasonable time, using both UVCluster and SCluster. Also, for networks with up to 1000 nodes, the user can test the three programs together, obtaining the results in minutes to a few hours.

Discussion

As the amount of biological information is rapidly increasing, one of the main goals in bioinformatics is the generation of fast programs able to deal with large datasets. For network analyses, the bottleneck of the iterative hierarchical clustering strategy is precisely that the clustering algorithm must be repeatedly used to generate a sufficiently large set of iterations as to be representative of the underlying structure of the graph. The second part of the analysis, the construction of the tree applying a phylogenetic algorithm is performed just once and therefore has little effect in the time complexity of the program. As already indicated in the Introduction, the applications of our UVCluster program were limited by the high amount of time needed for analyzing large networks. An optimization of the iterative clustering method implemented in that program was therefore mandatory. By setting certain restrictions (fixed *AC*), we have qualitatively reduced the time complexity of the UVCluster algorithm. Traditionally limited to analyses below 1000 units, the current algorithm can cope with networks of several thousand units in a few hours. This allows analyzing some very interesting datasets, such as the whole interactome of the eukaryotic species *Saccharomyces cerevisiae* (see benchmark C above).

A second significant advantage of Jerarca is that it also includes two novel algorithms, RCluster and SCluster, which provide alternative ways of computing the secondary distances between the nodes of the graph. RCluster randomly grows multiple clusters at the same time, avoiding the greedy agglomerative process implemented in UVcluster. However, the randomization process required makes the program slower than UVCluster. SCluster is just the opposite: it is the fastest and greediest of the three algorithms. In spite of its simplicity, its performance is also appropriate (See results for benchmark A above). Since Jerarca allows to execute several parallel analyses, we recommend to use the three iterative algorithms for networks with up to 1000 nodes.

A complete analysis of such networks may require less than two hours (see Results). With larger networks, up to 10000 units, both UVCluster and SCluster can be used, the analyses with both programs requiring just a few hours. The inclusion of SCluster, which runs in *O(n log n)* time, allows for the analyses of even larger networks. This may be of interest in fields such as the analysis of coexpression or gene interaction networks, in which the number of nodes (in those cases, corresponding to genes) may be in the tens of thousands. All these considerations obviously refer to analyses using a single computer. However, it is important to take into account that the programs can be very easily parallelized, given that the iterations can be divided into multiple processors and the results added together at the end of the computation.

In addition to UPGMA, already included in the original version of UVCluster, Jerarca also allows the alternative of building the trees using the neighbor-joining algorithm, probably the most frequently used algorithm to generate trees from a distance matrix. We suggest to obtain both trees (which is almost instantaneous), in order to evaluate the congruence of the results. An additional advantage of Jerarca respect to UVCluster refers to the determination of the optimal partitions of the graph according to two statistical parameters (Q and H). We added these options considering that the users may be often not only interested in obtaining a hierarchical representation of the network, but also in how the network can be divided into clusters or communities (see Introduction). The strategy used to obtain the partitions is in fact quite simple, given that the tree is just scanned from root to leaves. Therefore, the number of partitions examined is quite reduced (equal to the number of nodes *n*). More complex methods can be easily envisaged. For example, partitions could be generated at different distances from the root in different sections of the tree. However, although this option may potentially improve the likelihood of obtaining a better partition of the network, it is computationally much more expensive. We plan to explore this possibility in future versions of the suite. A final advantage of Jerarca is the set of outputs that it generates, which is much more complete than the one provided by our original UVCluster program. The possibility to directly export the data to powerful packages such as MEGA and Cytoscape will allow the users both to perform additional analyses that may complement those generated by Jerarca and to obtain sophisticated graphical representations of the results. All these advantages clearly make Jerarca a better tool to perform iterative clustering analyses of network data than our original UVCluster program.

The program, along with the source code is freely available under the GNU General Public License v3 at http://jerarca.sourceforge.net. The modular code structure of Jerarca permits easily including new features to the program. New algorithms, both iterative and for building the trees, as well as new indexes for extracting the optimal partition of the tree, can be easily added.

Acknowledgments

The authors would like to thank Vicente Arnau for his contributions in the development of the original UVCluster program, and Antonio Marco for his suggestions about how to implement the calculation of the H parameter.

Author Contributions

Conceived and designed the experiments: IM. Performed the experiments: RA. Analyzed the data: RA. Contributed reagents/materials/analysis tools: RA. Wrote the paper: IM.

References

1. Barabási AL, Oltvai ZN (2004) Network biology: understanding the cell's functional organization. Nat Rev Genet 5: 101–114.
2. Aittokallio T, Schwikowski B (2006) Graph-based methods for analysing networks in cell biology. Brief Bioinform 7: 243–255.
3. Sharan R, Ulitsky I, Shamir R (2007) Network-based prediction of protein function. Mol Syst Biol 3: 88.
4. Spirin V, Mirny LA (2003) Protein complexes and functional modules in molecular networks. Proc Natl Acad Sci U S A 100: 12123–12128.
5. Bader GD, Hogue CWV (2003) An automated method for finding molecular complexes in large protein interaction networks. BMC Bioinformatics 4: 2.
6. Brun C, Herrmann C, Guénoche A (2004) Clustering proteins from interaction networks for the prediction of cellular functions. BMC Bioinformatics 5: 95.
7. Pereira-Leal JB, Enright AJ, Ouzounis CA (2004) Detection of functional modules from protein interaction networks. Proteins 54: 49–57.
8. Pržulj N, Wigle DA, Jurisica I (2004) Functional topology in a network of protein interactions. Bioinformatics 20: 340–348.
9. Altaf-Ul-Amin M, Shinbo Y, Mihara K, Kurokawa K, Kanaya S (2006) Development and implementation of an algorithm for detection of protein complexes in large interaction networks. BMC Bioinformatics 7: 207.
10. Newman MEJ, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69: 026113.
11. King AD, Pržulj N, Jurisica I (2004) Protein complex prediction via cost-based clustering. Bioinformatics 20: 3013–3020.
12. Arnau V, Mars S, Marín I (2005) Iterative cluster analysis of protein interaction data. Bioinformatics 21: 364–378.
13. Rives AW, Galitski T (2003) Modular organization of cellular networks. Proc Natl Acad Sci U S A 100: 1128–1133.
14. Lu H, Zhu X, Liu H, Skogerbø G, Zhang J, et al. (2004) The interactome as a tree – an attempt to visualize the protein-protein interaction network in yeast. Nucl Acids Res 32: 4804–4811.
15. Yip AM, Horvath S (2007) Gene network interconnectedness and the generalized topological overlap measure. BMC Bioinformatics 8: 22.
16. Lucas JI, Arnau V, Marín I (2006) Comparative genomics and protein domain graph analyses link ubiquitination and RNA metabolism. J Mol Biol 357: 9–17.
17. Marco A, Marín I (2007) A general strategy to determine the congruence between a hierarchical and a non-hierarchical classification. BMC Bioinformatics 8: 442.
18. Marco A, Marín I (2009) Interactome and Gene Ontology provide congruent yet subtly different views of a eukaryotic cell. BMC Syst Biol 3: 69.
19. Sokal RR, Michenner CD (1958) A statistical method for evaluating systematic relationships. Univ Kansas Sci Bull 38: 1409–1438.
20. Saitou N, Nei M (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol 4: 406–425.
21. Nei M, Kumar S (2000) Molecular evolution and phylogenetics. Oxford University Press.
22. Farris JS (1972) Estimating phylogenetics trees from distance matrices. Am Nat 106: 645–668.
23. Clauset A, Newman MEJ, Moore C (2005) Finding local community structure in networks. Phys Rev E 72: 026132.
24. Tamura K, Dudley J, Nei M, Kumar S (2007) MEGA4: Molecular Evolutionary Genetics Analysis (MEGA) software version 4.0. Mol Biol Evol 24: 1596–1599.
25. Cline MS, Smoot M, Cerami E, Kuchinsky A, Landys N, et al. (2007) Integration of biological networks and gene expression data using Cytoscape. Nature Protocols 2: 2366–2382.
26. Watts DJ (1999) Small worlds. The dynamics of networks between order and randomness. Princeton University Press.
27. Pu SY, Wong J, Turner B, Cho E, Wodak SJ (2009) Up-to-date catalogues of yeast protein complexes. Nucl Acids Res 37: 825–831.
28. Breitkreutz BJ, Stark C, Reguly T, Boucher L, Breitkreutz A, et al. (2008) The BioGRID interaction database: 2008 update. Nucl Acids Res 36: D637–D640.