BIOINFORMATICS ORIGINAL PAPER

Vol. 26 no. 8 2010, pages 1105–1111
doi:10.1093/bioinformatics/btq078

Data and text mining

Advance Access publication February 24, 2010

SPICi: a fast clustering algorithm for large biological networks

Peng Jiang[1,2] and Mona Singh[1,2,*]

[1]Lewis-Sigler Institute for Integrative Genomics and [2]Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Clustering algorithms play an important role in the analysis of biological networks, and can be used to uncover functional modules and obtain hints about cellular organization. While most available clustering algorithms work well on biological networks of moderate size, such as the yeast protein physical interaction network, they either fail or are too slow in practice for larger networks, such as functional networks for higher eukaryotes. Since an increasing number of larger biological networks are being determined, the limitations of current clustering approaches curtail the types of biological network analyses that can be performed.

Results: We present a fast local network clustering algorithm SPICi. SPICi runs in time $O(V \log V + E)$ and space $O(E)$, where V and E are the number of vertices and edges in the network, respectively. We evaluate SPICi's performance on several existing protein interaction networks of varying size, and compare SPICi to nine previous approaches for clustering biological networks. We show that SPICi is typically several orders of magnitude faster than previous approaches and is the only one that can successfully cluster all test networks within very short time. We demonstrate that SPICi has state-of-the-art performance with respect to the quality of the clusters it uncovers, as judged by its ability to recapitulate protein complexes and functional modules. Finally, we demonstrate the power of our fast network clustering algorithm by applying SPICi across hundreds of large context-specific human networks, and identifying modules specific for single conditions.

Availability: Source code is available under the GNU Public License at http://compbio.cs.princeton.edu/spici

Contact: mona@cs.princeton.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on September 25, 2009; revised on February 17, 2010; accepted on February 19, 2010

1 INTRODUCTION

High-throughput experimental technologies, along with computational predictions, have resulted in large-scale biological networks for numerous organisms. In recent years, much research effort has focused on analyzing these biological networks in order to obtain hints about cellular organization and functioning. Clustering is perhaps the most common approach for global network analysis, and is frequently applied to uncover functional modules and protein complexes, and to infer protein function (Bader and Hogue, 2003; Hartwell *et al*., 1999; Pereira-Leal *et al.*, 2004; Rives and Galitski, 2003; Spirin and Mirny, 2003). As a result, numerous clustering algorithms for biological networks have been developed (e.g. Altaf-Ul-Amin *et al*., 2006; Bader and Hogue, 2003; Blatt *et al*., 1996; Brun *et al.*, 2003; Chen and Yuan, 2006; Colak *et al*., 2009; Enright *et al*., 2002; Georgii *et al*., 2009; King *et al*., 2004; Loewenstein *et al*., 2008; Navlakha *et al.*, 2009; Palla *et al*., 2005; Samanta and Liang, 2003; Sharan *et al*., 2005).

Previous methods for clustering biological networks work well on networks of moderate size. However, the size and number of biological networks continue to grow. For example, by extensive data integration, proteome-scale functional networks have been built for hundreds of organisms across the evolutionary spectrum (Jensen *et al*., 2009). Recently, by additionally considering specific biological processes (BPs) of interest, hundreds of context-specific functional networks for human have been built (Huttenhower *et al*., 2009). Moreover, in the near future, biological networks will include numerous additional biological entities such as non-coding RNAs as well as a wider range of interaction types.

Large networks present considerable challenges for existing clustering approaches. Here, we develop a new efficient network clustering algorithm SPICi ('spicy', Speed and Performance In Clustering). SPICi builds clusters greedily, starting from local seeds that have high weighted degree, and adding nodes that maintain the density of the clusters and are adjacent to a suitable fraction of nodes within them. The intuition underlying SPICi is similar to that of DPClus (Altaf-Ul-Amin *et al*., 2006). However, SPICi exploits a simpler cluster expansion approach, uses a different seed selection criterion and incorporates interaction confidences. Approaches based on enumeration have also been developed; these aim to uncover all clusters with specific density requirements. CFinder (Palla *et al*., 2005) finds clusters such that each consists of a maximal connected component of adjacent cliques of size k where two cliques are adjacent if they share $k-1$ nodes. An alternate approach relaxes the requirement of complete cliques and instead finds all subsets of nodes with high density (Colak *et al*., 2009; Georgii *et al*., 2009). While these approaches guarantee that they output all clusters with a particular property, they are computationally intensive. In contrast, SPICi takes a heuristic approach with respect to the clusters it outputs, but guarantees a runtime of $O(V \log V + E)$, where V and E are the number of vertices and edges in the network.

We demonstrate SPICi's excellent runtime and its state-of-the-art performance via several analyses. First, we compare SPICi to nine previous network clustering algorithms (Altaf-Ul-Amin *et al*., 2006; Bader and Hogue, 2003; Blatt *et al*., 1996; Enright *et al*.,

*To whom correspondence should be addressed.

© The Author(s) 2010. Published by Oxford University Press.
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.5), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Table 1. Test set of biological networks

	Biogrid Yeast	STRING Yeast	Biogrid Human	STRING Human	Bayesian Human
Vertices	5361	6371	7498	18 670	24 433
Edges	85 866	311 765	23 730	1 432 538	298 473 526

Five test networks are considered, two for yeast and three for human (see text). For each network, the number of vertices and edges are given.

2002; Georgii *et al*., 2009; King *et al*., 2004; Loewenstein *et al*., 2008; Palla *et al*., 2005; Sharan *et al*., 2005) on a test set of five existing biological networks (Table 1). SPICi is more than 4–1000 times faster than the previous approaches on the networks for which the approaches terminate within 12 h on a standard desktop machine. Moreover, it is the only algorithm of those tested that is able to cluster all the networks within a reasonable amount of time. Second, we show that even though SPICi is much faster than previous clustering approaches, the clusters it uncovers in biological networks recapitulate functional modules just as well. Third, we perform a robustness analysis on synthetic networks, as described by Brohee and van Helden (2006), and show that SPICi has very good performance in recapitulating protein complexes, deteriorating only on extremely incomplete networks. We also find SPICi to be robust to perturbations in dense functional networks. Finally, we use SPICi to cluster 230 large, context-specific human networks (Huttenhower *et al*., 2009) and identify modules specific for single conditions; because of the size and number of networks, this type of analysis was made feasible only by utilizing our new fast clustering approach.

2 METHODS

2.1 Algorithm framework

2.1.1 Preliminaries Given a weighted network, the goal of our algorithm is to output a set of disjoint dense subgraphs. We model the network as a undirected graph $G=(V,E)$ with a confidence score $0<w_{u,v}\leq 1$ for every edge $(u,v)\in E$. For any two vertices u and v without an edge between them, we set $w_{u,v}=0$. Our approach utilizes several measures. For each vertex u, we define its weighted_degree, $d_w(u)$, as the sum of all of its incident edges' confidence values:

$$d_w(u)=\sum_{v:(u,v)\in E} w_{u,v}.$$

For each set of vertices $S\subset V$, we define its density as the sum of the weights of the edges among them, divided by the total number of possible edges (i.e. the density of a set is a measure of how close the induced sub-graph is to a clique, and varies from 0 to 1):

$$\text{density}(S)=\frac{\sum_{u,v\in S} w_{u,v}}{|S|*(|S|-1)/2}.$$

Finally, for each vertex u and set $S\subset V$, we define the support of u by S as the sum of the confidences of u's edges that are incident to vertices in S:

$$\text{support}(u,S)=\sum_{v\in S} w_{u,v}.$$

2.1.2 Algorithm overview We use a heuristic approach to greedily build clusters. SPICi builds one cluster at a time, and each cluster is expanded from an original seed pair of proteins. The unclustered node that has the highest support for the cluster is added if the support is high enough and the density of the cluster remains higher than a user-defined threshold; otherwise, the cluster is output and its nodes are removed from the network. SPICi thus has two parameters: T_s, the support threshold and T_d, the density threshold. (See Fig. 1 for a simplified example.)

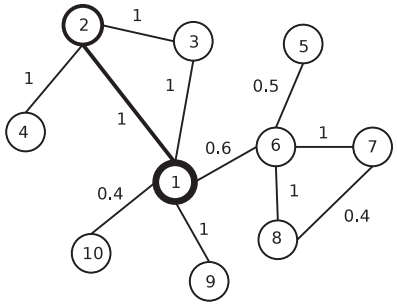

Fig. 1. Example to illustrate the clustering process. This example network has 10 vertices, and every edge has confidence 1 except (1,6), (1,10), (5,6) and (7,8). Suppose the support threshold is $T_s=0.5$. The highest weighted degree vertex, vertex 1 with weighted degree 4, is taken as a seed protein. The highest non-empty bin (0.8,1] for vertex 1 is composed of neighboring vertices 2, 3 and 9. Of these, vertex 2 has weighted degree 3, the largest of this bin, and it is taken as the second seed vertex. In the first step of the density-based search, vertex 3 has the highest support, 2, from the current cluster {1,2}. We add vertex 3 to the cluster and this cluster now has density 1. Then, all the remaining vertices have support less than $density\times cluster\_size\times T_s=1.5$. Thus we stop expanding the cluster and output {1,2,3} as the first cluster. After this, the next search will start from vertex 6 and output {6,7,8} as the next cluster. Vertices 4, 5, 9 and 10 are left as singleton clusters.

2.1.3 Seed selection To select the seed vertices, we first find the vertex u that has the highest weighted degree in the current network. Then, we divide the neighboring vertices of u into five bins based on their edge weights: $(0,0.2]$, $(0.2,0.4]$, $(0.4,0.6]$, $(0.6,0.8]$ and $(0.8,1]$. We search from the highest weight bin $(0.8,1]$ to the lowest weight bin $(0,0.2]$. If our current bin is not empty, we use the vertex v in it with the highest weighted degree as the second seed vertex. We refer to (u,v) as the seed edge. We utilize this heuristic approach for seed selection based on two observations for functional networks. First, there is a positive correlation between the weighted degree of a node and a measure of the overall functional enrichment found among its interacting proteins (data not shown); this suggests that high weighted degree nodes are meaningful starting points for local module searches in functional networks. Second, two vertices are more likely to be in the same module if the weight on the edge between them is higher. This is why we search from the highest weight bin to the lowest weight bin. For vertices in each bin, their edge weights to the first seed vertex are quite similar, and by taking the one with highest weighted degree, we obtain a larger candidate set for continuing the search.

2.1.4 Cluster expansion After obtaining two seed nodes with an edge between them, we grow each cluster in a procedure similar to that of Altaf-Ul-Amin *et al*. (2006). At each step, we have a current vertex set S for the cluster, which initially consists of the two seed vertices. We search for the vertex u with maximum value of $\text{support}(u,S)$ amongst all the unclustered vertices that are adjacent to a vertex in S. If $\text{support}(u,S)$ is smaller than a threshold, we stop expanding this cluster and output it. If not, we put vertex u into S and update the density value. If the density value is smaller than our density threshold T_d, we do not include u in the cluster and output S. We repeat this procedure until all vertices in the graph are clustered.

2.1.5 Implementation and runtime We implement our algorithm using two critical data structures, described in more detail in the next paragraph. The first data structure is a priority queue, **DegreeQ**, to pick the seed proteins

Search

Initialize **DegreeQ** to be V
While **DegreeQ** is not empty

1. Extract u from **DegreeQ** with largest weighted degree
2. If u has adjacent vertices in **DegreeQ** then
   - Find from u's adjacent vertices the second seed protein v (see text)
   - $S = \textbf{Expand}(u, v)$

   else $S = \{u\}$
3. $V = V - S$
4. Delete all vertices in S from **DegreeQ**
5. For each vertex t in **DegreeQ** that is adjacent to a vertex in S, decrement its weighted degree by $support(t, S)$

Expand(u,v)

Initialize the cluster $S = \{u, v\}$
Initialize **CandidateQ** to contain vertices neighboring u or v

While **CandidateQ** is not empty

1. Extract t from **Candidate** with highest **support**(t, S)
2. If $support(t, S) \geq T_s * |S| * density(S)$ and $density(S \cup \{t\}) > T_d$ then
   - $S = S + \{t\}$
   - Increase the **support** for vertices connected to t in **CandidateQ**
   - For all unclustered vertices adjacent to t, insert them into **CandidateQ** if not already present

   else break from loop

return S

Fig. 2. Pseudocode of the algorithm. The **Search** procedure iteratively finds seed proteins and calls **Expand** to build a cluster from them.

from which clusters are built. Initially, all proteins are organized based on their weighted degree. Once a cluster is built and output, its proteins are removed from **DegreeQ** and the weighted degrees of all proteins adjacent to these are decreased to reflect their connectivity to other unclustered proteins. Thus, in addition to extracting the maximum degree node, **DegreeQ** also needs to support deletions and decrease key operations. The second data structure, **CandidateQ**, is used to expand clusters. It is also a priority queue, where each element is a node u adjacent to one of the nodes in the cluster S being built, and is prioritized based on support(u, S). In addition to extracting the max element, **CandidateQ** needs to support insertions, as neighbors of nodes added to S are added to it, and increase key operations, since as S grows, the supports of all vertices with respect to S increase. We describe SPICi with these data structures in the pseudocode given in Figure 2.

The specific data structures we use are now described. For **CandidateQ**, we need efficient **Insert**, **ExtractMax** and **IncreaseKey** operations. We use a Fibonacci heap (Fredman and Tarjan, 1987), as amortized analysis gives a time complexity of $O(1)$ for **Insert** and **IncreaseKey** and of $O(\log n)$ for **ExtractMax**, where n is the number of possible items in the heap. We now count the overall time cost for all **Expand** operations. There are at most $O(V)$ **ExtractMax** operations, so the overall cost for them is $O(V \log V)$. For **IncreaseKey**, each operation is associated with an edge connected to that vertex, so the overall cost for these operations is $O(E)$. For **Insert**, the overall cost is $O(E)$ since each insertion is associated with an edge from S.

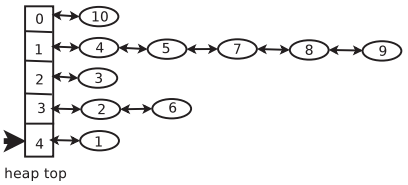

Fig. 3. Integer heap data structure corresponding to the example in Figure 1. Vertices are put into the data structure based on their weighted degree. For example, Vertex 6 has weighted degree 3.1. It is rounded to 3 in slot 3. Vertex 5 has weighted degree 0.5. It is rounded to 1 in slot 1. Each slot is organized as a doubly linked list, so we can delete and insert an element in O(1) time.

So, the overall time cost for all **Expand** operations is $O(V \log V + E)$. For the Fibonacci heap, the space complexity is $O(n)$, so we get a space complexity of $O(V)$.

For **DegreeQ**, we need to support **ExtractMax**, **Delete** and **DecreaseKey**. As an optimization, we round off each vertex's weighted degree to an integer, and utilize an extremely fast data structure, which we refer to as an Integer heap, with time complexity of $O(1)$ for **Delete** and **DecreaseKey**, and an amortized time complexity of $O(1)$ for **ExtractMax**. More specifically, since every element in the heap is an integer, we use an array as its backbone, and at every slot in the array, we use a doubly linked list for all vertices with weighted degree rounded to that slot index. Since each edge has confidence ≤ 1, the number of slots is $O(V)$, and the total space necessary to handle all vertices is $O(V)$. (See Fig. 3 for a schematic of our Integer heap data structure.) For insertion and deletion, we require the procedure call to provide the pointer to the doubly linked list node, and so these two operations can be performed in $O(1)$ time. For **DecreaseKey**, we first disconnect the node and then reconnect it to a new slot with $O(1)$ cost. Note that we store the initial weights per edge, and then round each time we perform **DecreaseKey**. For **ExtractMax**, we just need to pop out a value at the top slot of our array. If any array slot becomes empty, we need to search down the array until we reach a new non-empty slot. The total number of all down searches is $V - 1$, which is the maximum length of the array. Thus, if there are a total of V operations, the amortized time for each operation is $O(1)$. The time complexity of procedure **Search** without considering the time spent for **Expand** is $O(V + E) = O(E)$, as there are at most V **ExtractMax** and **Delete** operations, E **DecreaseKey** operations, and each edge is considered at most once when finding the second seed vertex. Thus, considering **Search** and **Expand** together, SPICi has time complexity $O(V \log V + E)$ and space complexity $O(E)$.

2.2 Network datasets

We concentrate our initial analysis on two networks for yeast and three networks for human (Table 1). The two Biogrid (Breitkreutz *et al*., 2008) networks consist of experimentally determined physical and genetic interactions. The two STRING (Jensen *et al*., 2009) networks and the human Bayesian network (Huttenhower *et al*., 2009) consist of functional associations between proteins that are derived from data integration. For Biogrid, we extract all non-redundant interaction pairs, including all protein physical and genetic interactions. For STRING (Jensen *et al*., 2009), we use all weighted interactions. For the Bayesian human network, we use the global network from Huttenhower *et al*. (2009); this network is not tuned toward any specific BP. In subsequent analysis, we also use the 229 context-specific human networks from Huttenhower *et al*. (2009); here each context is a BP from the Gene Ontology (GO; Ashburner *et al*., 2000), and the training set is altered according to the specific BP context so that the network will better represent that specific context. None of the networks are further processed. For functional module discovery and/or protein function prediction, it may be beneficial in practice to remove high-degree nodes and/or otherwise prune the networks; since different processing may be necessary for different networks,

and our primary goal is to test SPICi's ability to cluster large networks, such variations are not explored here.

2.3 Computational experiments

All experiments are run on an Intel 2 GHz dual core computer with 2 GB memory. We compare our approach to *SPC* (Blatt *et al*., 1996), *MCL* (Enright *et al*., 2002), *MCODE* (Bader and Hogue, 2003), *RNSC* (King *et al*., 2004), *Cfinder* (Palla *et al*., 2005), *NetworkBLAST* (Sharan *et al*., 2005), *DPClus* (Altaf-Ul-Amin *et al*., 2006), *MCUPGMA* (Loewenstein *et al*., 2008) and *DME* (Georgii *et al*., 2009). We briefly highlight the main features of these algorithms. *SPC* associates a 'spin' with each node, and spin–spin correlations are used to partition the network. *MCL* is a global clustering approach based on modified random walks on networks. *MCODE* is one of the first approaches specifically geared for clustering interactomes, and greedily grows clusters from a seed node. *CFinder* finds a set of k-clique percolation clusters, each of which consists of a maximal connected component of adjacent cliques of size k where two cliques are adjacent if they share $k-1$ nodes. *NetworkBLAST*, designed for comparing multiple protein networks but applicable for clustering a single protein network, greedily builds small 'dense' clusters. *DPClus* is a greedy approach that grows clusters based on adding nodes that are well connected to other nodes in the cluster and that maintains cluster density. *MCUPGMA* is a memory-efficient average-link hierachical clustering algorithm. *DME* finds all clusters that satisfy a user-defined minimum density threshold.

For *MCL*, we set the inflation factor to 1.8, as this has been found to yield the best performance in clustering biological networks (Brohee and van Helden, 2006). For *RNSC*, we use the parameters given in the sample README file that comes with the software (-e3 -n2 -N100 -D40 -d10 -c300 -t15 -T2). For SPICi, we set both T_s and T_d to 0.5. We use the default parameters for the other approaches. For *MCUPGMA*, the distance between two nodes u and v is set to $1-w_{u,v}$. By default, *MCUPGMA* allocates a fixed amount of memory corresponding to a prespecified limit on the number of edges that are allowed into memory in each clustering iteration. Without this limit, *MCUPGMA* runs out of memory on the large Bayesian human networks. While we carefully experimented with changing this memory limit for the human functional networks, the running time of *MCUPGMA* on these networks still exceeded our time limits and thus for simplicity, we ran the program with its default parameter. We also note that *MCUPGMA* outputs a hierarchical clustering dendogram without a split into clusters. We obtain clusters using an inconsistency coefficient (Jain and Dubes, 1988); this is a standard procedure in MATLAB's statistics toolbox for processing hierarchical clustering dendograms to obtain clusters. The inconsistency coefficient for each merge of the the dendogram is computed by taking its height in the dendogram and subtracting the average height of all merges considered, and dividing this by the standard deviation of these the merges. The higher the value of this coefficient, the more a merge would connect dissimilar nodes. As in MATLAB's default parameters, we consider only the current merge, and the merges one level below. We use a value of 0.8 to cut the tree, as these values range from 0 to 1.2 in the considered dendograms and there are only two peaks in the distribution (data not shown), one corresponding to merges with inconsistency value <0.1, (i.e. the initial merges) and the other corresponding to merges in the range of 0.7 and 0.8.

All reported runtimes are wall clock times for running the clustering portion of the programs only. We do not report CPU times, as some of the clustering algorithms are designed to be run within a user interface, making strict system timing calls difficult.

2.4 GO analysis

The quality of clusters obtained from all algorithms are evaluated using the framework described in Song and Singh (2009). We use GO to build our reference set of functional modules, with all IEA (inferred from electronic annotation) terms removed, as suggested in Rhee *et al*. (2008). For each organism, only GO terms that annotate at most 1000 proteins are considered. For a given GO annotation **A**, we define the 'functional module set' G_A to consist of all genes annotated with **A**. Song and Singh (2009) utilize the following three measures to measure the overlap between computationally derived cluster and the GO functional modules:

(1) Jaccard: for each cluster **C**, its Jaccard value with each GO derived functional module group G_A is computed as $\frac{|C \cap G_A|}{|C \cup G_A|}$. The Jaccard measure for cluster **C** is the maximum Jaccard value over all considered GO terms **A**.

(2) PR (precision–recall): for each cluster **C**, its **PR** value with a GO derived functional module G_A is computed as $\frac{|C \cap G_A|}{|G_A|}\frac{|C \cap G_A|}{|C|}$. The **PR** measure for **C** is the maximum PR value over all considered GO terms **A**.

(3) Semantic density: for each cluster, the average semantic similarity between each pair of annotated proteins within it is computed. In particular, for proteins p_1 and p_2 with annotations $A(p_1)$ and $A(p_2)$ respectively, the semantic similarity of their GO annotations is defined as:

$$\frac{2 * \min_{a \in A(p1) \bigcap A(p2)} \log(p(a))}{\min_{a \in A(p1)} \log(p(a)) + \min_{a \in A(p2)} \log(p(a))},$$

where $p(a)$ is the fraction of annotated proteins with annotation a in the organism (Lord *et al.*, 2003; Song and Singh, 2009). Note that more specific annotations a have smaller values of $p(a)$, and $\log(p(a)) \leq 0$. For our semantic density calculations here, all GO terms are considered, even those annotating more than 1000 proteins.

Each of these three measures varies from 0 to 1, with higher values indicating better agreement of the uncovered clusters with functional modules corresponding to GO. We consider the GO BP and cellular component (CC) ontologies separately. For each cluster, we calculate these three measures separately for both ontologies, and these measures are assigned to all proteins within the cluster. Genes in singleton clusters are penalized by having Jaccard, PR and semantic density values of 0. Finally, for each of the six measures (three BP and three CC), we compute its average value over all proteins in the network; this is equivalent in the case of non-overlapping clusters to taking a weighted average over all clusters, where each cluster is weighted by its size. For more details, please see Song and Singh (2009). Some of the tested approaches output overlapping clusters. In this case, if a protein is found in more than one cluster, then each of its six measures is obtained by averaging over the values obtained from each of its clusters.

2.5 Robustness analysis

In order to characterize SPICi's robustness to changes in the network, we use the procedure of Brohee and van Helden (2006) to characterize how well MIPS complexes are recapitulated from synthetic test network data. In particular, networks are initially created for each of the 104 *Saccharomyces cerevisiae* MIPS (Mewes *et al*., 2004) complexes that are not determined from high-throughput experiments. A node is included in the network for each protein in one of these complexes, and an edge is included in the network between any two proteins in the same complex. This network with $|E|$ edges is then modified as follows. For an edge addition rate p_a and an edge deletion rate p_d, first $p_a \cdot |E|$ edges are added to the network, and then $p_d \cdot |E|$ edges are chosen uniformly at random for deletion. Brohee and van Helden (2006) utilize two measures, **Accuracy** and **Separation**, for evaluating clusterings. **Accuracy** measures how well the clustering recovers the gold standard MIPS complexes. **Separation** measures how specifically the clustering can be mapped to the MIPS gold standard set without cross-complex contamination. [See Brohee and van Helden (2006) for precise definitions of these two measures.]

Table 2. Running time and memory usage of clustering approaches

Running Time (s)	Biogrid Yeast	Biogrid Human	STRING Yeast	STRING Human	Bayesian Human
SPICi	**1**	**1**	**2**	**7**	**1111**
MCUPGMA	5	4	9	33	
MCL	336	114	645	4926	
NetworkBLAST	1904	427	7848		
SPC	183	215	219		
MCODE	101	49	7848		
DPClus	1602	2113			
RNSC	172	17	1325	23 448	
CFinder		25			
DME					
Memory (MBs)	**Biogrid Yeast**	**Biogrid Human**	**STRING Yeast**	**STRING Human**	**Bayesian Human**
SPICi	**1.2**	**1.5**	**15.1**	**90.5**	**1143.0**
MCUPGMA	259.1	259.1	259.1	259.1	
MCL	73.3	24.9	111.7	357.0	
NetworkBLAST	61.9	60.5	72.8		
SPC	220.5	430.3	311.0		
MCODE	375.6	306.1	606.9		
DPClus	140.2	202.1			
RNSC	25.9	9.8	82.3	349.4	
CFinder		23.0			
DME					

Running time and peak memory usage of each algorithm on each network. For running time, clock times, rounded to the second, are reported. Peak memory usage is given in megabytes. Note that *MCUPGMA*'s memory usage is preallocated with a default limit and is thus constant for these networks (Section 2). Blank entries in the table indicate that the approach did not successfully cluster the network within 12 h. In each column, bold entries indicate the smallest memory usage and the fastest running time obtained on the given network.

3 RESULTS AND DISCUSSION

3.1 Speed and memory analysis

We run SPICi and nine previous clustering approaches on our five network datasets. Table 2 gives the runtime and memory usage of each approach on each of the datasets. SPICi is the only approach that can cluster each of the five networks within 12 h; indeed it takes <10 s for four of the five networks and takes <20 min on the largest dense functional network. Even on networks that can be clustered by the other approaches, SPICi obtains substantial speed-ups. This decrease in runtime is accompanied by a decrease in memory usage as well. For the human Bayesian functional network, SPICi uses 1.11 GB of memory, which corresponds to the size of the network itself.

3.2 GO analysis

We use the procedure from Song and Singh (2009) to assess the overall quality of the clusters we find. Three approaches (SPICi, *MCUPGMA*, *MCL*) can cluster four of the networks, and we focus on these methods and networks in our analysis in the main body of the article. We find that neither SPICi, *MCUPGMA* nor *MCL* clearly dominates the other approaches (Table 3 and Supplementary Table 1). We observe that these three approaches have complimentary strengths when considering clusters of different sizes on the functional networks (see Supplementary Fig. 1).

Table 3. GO analysis of clusters output by SPICi, MCUPGMA and MCL

		BP			CC		
Network	Algorithm	sDensity	Jaccard	PR	sDensity	Jaccard	PR
Biogrid Yeast	SPICi	0.368	**0.214**	**0.183**	0.379	0.167	**0.141**
	MCUPGMA	**0.414**	0.200	0.160	**0.444**	0.147	0.115
	MCL	0.284	0.208	0.156	0.324	**0.171**	0.125
Biogrid Human	SPICi	0.254	**0.183**	**0.159**	0.271	0.097	0.078
	MCUPGMA	0.319	0.179	0.150	0.348	0.096	0.074
	MCL	**0.348**	0.177	0.141	**0.388**	**0.120**	**0.091**
STRING Yeast	SPICi	0.466	**0.264**	**0.232**	0.450	**0.220**	**0.199**
	MCUPGMA	**0.579**	0.235	0.206	**0.584**	0.187	0.172
	MCL	0.227	0.205	0.194	0.261	0.167	0.143
STRING Human	SPICi	0.316	**0.210**	**0.180**	0.331	0.123	0.103
	MCUPGMA	**0.338**	0.200	0.178	**0.344**	0.088	0.074
	MCL	0.247	0.197	0.159	0.297	**0.163**	**0.125**

sDensity, Jaccard and PR values (see text) for SPICi, MCUPGMA and MCL on four networks, as judged via overlap with functional modules derived from the BP and CC ontologies. Bold entries correspond to the best values obtained for each measure on each network.

For clusters with at most five proteins, *MCUPGMA* has the highest average quality measures. On the other hand, SPICi's clusters of intermediate size (from 6 to 150 proteins) generally have higher quality measures. For clusters with more than 150 proteins, SPICi and *MCL* perform best.

Many of the remaining seven algorithms can cluster the smaller sized networks well, and in some cases may outperform the three approaches here; however, their runtime or memory requirements limit their applicability. We note that while it is possible to reduce the large functional networks into smaller ones by only keeping edges with a weight above a certain threshold, we find that, for all approaches we tested, by keeping all interactions, we find additional 'unique' functionally enriched clusters as well as an increase in the number of proteins in functionally enriched clusters. (Supplementary Material.)

3.3 Robustness analysis

We first apply the procedure of Brohee and van Helden (2006) to compare the robustness of SPICi against that of *MCL* and *MCUPGMA*. We build 10 synthetic test networks edges for each pairwise combination of 10 addition rates and 10 deletion rates. The averaged Separation and Accuracy measures (Brohee and van Helden, 2006) for each addition and deletion rate are shown in Figure 4 (see also Supplementary Table 2). We find that SPICi has better overall performance than *MCUPGMA* (Fig. 4b). When comparing *MCL* and SPICi, it is clear that neither method dominates the other at all noisy edge insertion and deletion rates. For low interaction insertion and deletion rates, the methods perform comparably. For high deletion rates, *MCL* generally outperforms SPICi. For high interaction addition rates and low interaction deletion rates, SPICi has better overall performance than *MCL*. These results suggest that SPICi is less sensitive to noisy edge addition than *MCL*, and is perhaps better suited for dense functional networks such as the STRING networks. Consistent with this, we find that SPICi is quite robust to perturbations of confidence

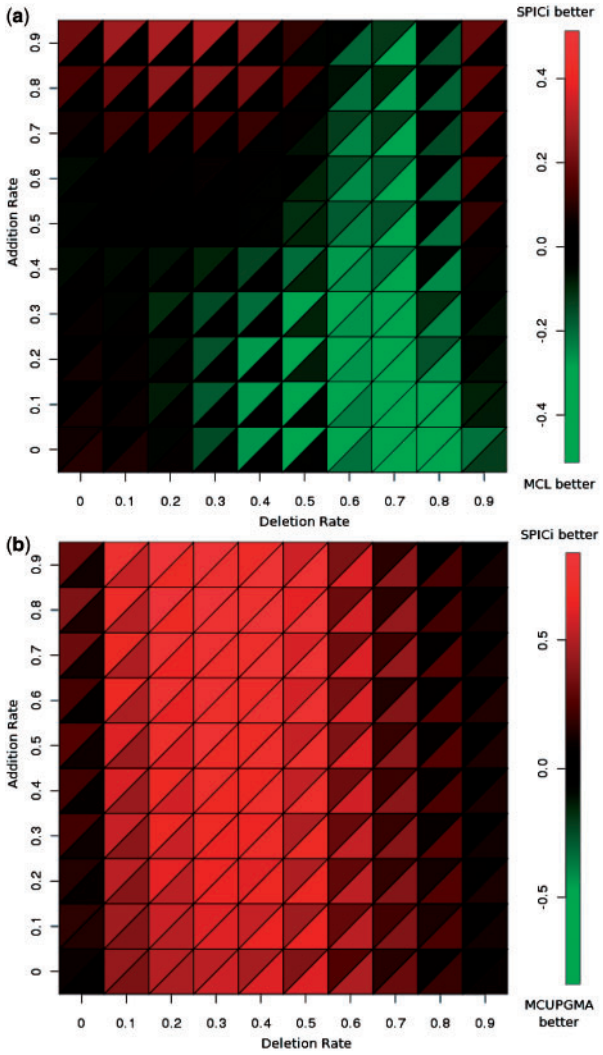

Fig. 4. Robustness analysis comparing SPICi, *MCL* and *MCUPGMA* in their ability to recapitulate MIPS complexes from synthetic networks. Ten edge deletion and insertion edges are considered (from 0.0 to 0.9 in increments of 0.1). The *x*-axis gives the random edge deletion rate, and the *y*-axis gives the noisy edge addition rate. Each cell corresponds to a single insertion and deletion rate combination. In (**a**), the lower triangle within each cell gives the average value of $\log_2$(SPICi Accuracy/MCL Accuracy) over the 10 networks generated for the corresponding insertion and deletion rate combination. The upper triangle within each cell gives the analogous $\log_2$(SPICi Separation/MCL Separation) values. Values greater than 0, shown in red, indicate that SPICi is better. Similarly, values smaller than 0, shown in green, indicate that *MCL* is better. In (**b**), the same data are shown, except SPICi is compared with *MCUPGMA*.

values in both the STRING human and yeast networks, with a steady but relatively modest decrease in average Jaccard, PR and semantic density values as increasing amounts of noise are added (Supplementary Table 3).

3.4 Clustering of numerous context-specific human functional networks

While the Bayesian human network from Huttenhower *et al.* (2009), obtained by global data integration of multiple sources

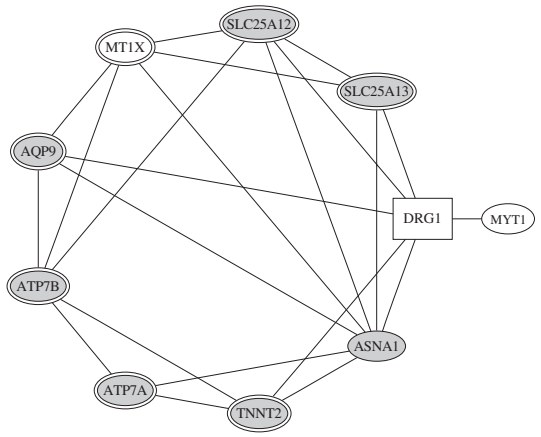

Fig. 5. A context-specific module found in the *response to inorganic substance* network of Huttenhower *et al.* (2009). We only show interactions with weights >0.5. Gray colored nodes correspond to proteins with the GO annotation *Transport*. Double peripheral ellipse nodes correspond to proteins with GO annotation *response to metal ion*. The DRG1 protein, shown in a box node, is discussed further in the text.

of evidence linking proteins, proves to be challenging for all the previous network clustering approaches, the authors actually created 229 additional networks of similar size. Each of these networks corresponds to one of 229 specific BPs. Here, we show the type of analysis SPICi enables by its fast clustering approach—analysis that would not be possible by the previous approaches. In particular, we utilize SPICi to uncover context-specific modules from these context-specific networks.

We use SPICi to cluster all 229+1 human functional networks. Altogether, we get 63 973 clusters of size >5 and density >0.5. We select context-specific modules utilizing the following criteria. For each candidate cluster, we require that:

(1) No uncovered clusters from any other context-specific network can overlap more than half of its proteins.

(2) The density of the cluster's set of proteins is <0.25 in the global network.

(3) Fewer than 10 other context-specific networks contain this set of proteins with a density >0.25.

By applying these three criteria, we attempt to uncover modules that are unique to a certain context. In total, 2088 clusters passed these criteria. As an example, we look at one such cluster, found in the *response to inorganic substance* network (Fig. 5). There are 10 proteins in this cluster. This cluster has very limited overlap (at most two proteins) with clusters found in the other networks. Moreover, all other networks contain this set of proteins with a density <0.25. The cluster is found to be enriched via the hypergeometric distribution with the annotations *response to metal ion* (*P*-value = 1.39E-015, seven proteins annotated) and *transport* (*P*-value = 4.20E-006, seven proteins annotated). An interesting case is the DRG1 protein (also known as developmentally regulated GTP-binding protein 1). It is annotated with GO terms such as *GTP binding* and *transcription factor binding*, but has no known annotations related with *response to metal ion* or *transport*. This uncovered cluster reveals DRG1's potential role in metal ion response and transport.

4 CONCLUSIONS

We have developed a fast, memory-efficient clustering algorithm, SPICi. SPICi is significantly faster than previous clustering algorithms for biological networks, and importantly, enables us to cluster larger networks than previously possible. Moreover, we have demonstrated via several analyses that the clusters uncovered by SPICi are of comparable quality to those found by other state-of-the-art algorithms. In our experience, SPICi is especially well-suited for dense networks, such as functional networks. Within sparser networks, we have found that SPICi also readily identifies dense regions, but for reasonable parameter settings will conservatively leave many proteins unclustered.

We have shown that SPICi can be effectively run on hundreds of large human context-specific networks in order to find context-specific modules. In the future, we foresee using SPICi to perform other types of comparative interactomics. For example, protein interaction networks for a single organism can be modified to incorporate information about each protein's tissue- or condition-specific expression, and comparing clusterings across these networks can help to identify modules that are either conserved across numerous conditions or specific to certain conditions. Given the large number of expression datasets, this leads to the possibility of hundreds or even thousands of varying networks across a single organism. SPICi's runtime and memory efficiency enables these new types of analyses, and should be particularly useful as biological networks continue to grow in size and number.

ACKNOWLEDGEMENTS

We thank Curtis Huttenhower and Olga Troyanskaya for providing their human context-specific networks. We thank members of the Singh group, especially Tao Yue and Jimin Song, for helpful discussions on our approach and for comments on our manuscript.

Funding: National Science Foundation (ABI-0850063 to M.S., in part); National Institutes of Health (GM076275 to M.S., in part); National Institute of Health Center of Excellence (grant P50 GM071508 to David Botstein, in part).

Conflict of Interest: none declared.

REFERENCES

Altaf-Ul-Amin,M. *et al.* (2006) Development and implementation of an algorithm for detection of protein complexes in large interaction networks. *BMC Bioinformatics*, **7**, 207.

Ashburner,M. *et al.* (2000) Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat. Genet.*, **25**, 25–29.

Bader,G.D. and Hogue,CW. (2003) An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, **4**, 2.

Blatt,M. *et al.* (1996) Superparamagnetic clustering of data. *Phys. Rev. Lett.*, **76**, 3251–3254.

Breitkreutz,B. *et al.* (2008) The BioGRID Interaction Database: 2008 Update. *Nucleic Acids Res.*, **36**, D637–D640.

Brohée,S. and van Helden,J. (2006) Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, **7**, 488.

Brun,C. *et al.* (2003) Functional classification of proteins for the prediction of cellular function from a protein-protein interaction network. *Genome Biol.*, **5**, R6.

Chen,J. and Yuan,B. (2006) Detecting functional modules in the yeast protein-protein interaction network. *Bioinformatics*, **22**, 2283–2290.

Colak,R. *et al.* (2009) Dense graphlet statistics of protein interaction and random networks. In *Pacific Symposium on Biocomputing*, pp. 178–189.

Enright,A.J. *et al.* (2002) An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.*, **30**, 1575–1584.

Fredman,M.L. and Tarjan,R.E. (1987) Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, **34**, 596–615.

Georgii,E. *et al.*(2009) Enumeration of condition-dependent dense modules in protein interaction networks. *Bioinformatics*, **25**, 933–940.

Jensen,L.J. *et al.* (2009) STRING 8–a global view on proteins and their functional interactions in 630 organisms. *Nucleic Acids Res.*, **37**, D412–D416.

Hartwell,L.H. *et al.* (1999) From molecular to modular cell biology. *Nature*, **402**, 6761.

Huttenhower,C. *et al.* (2009) Exploring the human genome with functional maps. *Genome Res.*, **19**, 1093–1106.

Jain,A. and Dubes,R. (1988) *Algorithms for Clustering Data*, 2nd edn. Prentice-Hall, NJ.

Jensen,L.J. *et al.* (2004) ArrayProspector: a web resource of functional associations inferred from microarray expression data. *Nucleic Acids Res.*, **32**, W445–W448.

King,A.D. *et al.* (2004) An efficient algorithm for large-scale detection of protein families. *Bioinformatics*, **20**, 3013–3020.

Lord,P.W. *et al.* (2003) Semantic similarity measures as tools for exploring the gene ontology. *Pac. Symp. Biocomput.*, **8**, 601.

Loewenstein,Y. *et al.* (2008) Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. *Bioinformatics*, **24**, i41–i49.

Mewes,H.W. *et al.* (2004) MIPS: analysis and annotation of proteins from whole genomes. *Nucleic Acids Res.*, **32**, D41–D44.

Navlakha,S. *et al.* (2009) Revealing biological modules via graph summarization. *J. Comput. Biol.*, **16**, 253–264.

Palla,G. *et al.* (2005) Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, **435**, 814–818.

Pereira-Leal,J. *et al.* (2004) Detection of functional modules from protein interaction networks. *Proteins*, **54**, 49–57.

Rhee,S.Y. *et al.* (2008) Use and misuse of the gene ontology annotations. *Nat. Rev. Genet.*, **9**, 509–515.

Rives,A.W. and Galitski,T. (2003) Modular organization of cellular networks. *Proc. Natl Acad. Sci. USA*, **100**, 1128–1133.

Samanta,M. and Liang,S. (2003) Predicting protein functions from redundancies in large-scale protein interaction networks. *Proc. Natl Acad. Sci. USA*, **100**, 12579–12583.

Sharan,R. *et al.* (2005) Conserved patterns of protein interaction in multiple species. *Proc. Natl Acad. Sci. USA*, **102**, 1974–1979.

Song,J. and Singh,M. (2009) How and when should interactome-derived clusters be used to predict functional modules and protein function? *Bioinformatics*, **25**, 3143–3150.

Spirin,V. and Mirny,L.A. (2003) Protein complexes and functional modules in molecular networks. *Proc. Natl Acad. Sci. USA*, **100**, 12123.