SPICI: a fast clustering algorithm for large biological networks
Peng Jiang$^{1,2}$ and Mona Singh$^{1,2,\ast}$
$^1$Lewis-Sigler Institute for Integrative Genomics and $^2$Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

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 uncover, 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 18, 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. CFinfer (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., 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).

\textsuperscript{\ast}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

<table>
<thead>
<tr>
<th>Biogrid</th>
<th>STRING</th>
<th>Biogrid</th>
<th>STRING</th>
<th>Bayesian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yeast</td>
<td>Yeast</td>
<td>Human</td>
<td>Human</td>
<td></td>
</tr>
<tr>
<td>Vertices</td>
<td>5361</td>
<td>6371</td>
<td>7498</td>
<td>18</td>
</tr>
<tr>
<td>Edges</td>
<td>85</td>
<td>866</td>
<td>311</td>
<td>765</td>
</tr>
</tbody>
</table>

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.

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: Ts, the support threshold and Td, the density threshold. (See Fig. 1 for a simplified example.)

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 Al-Rahim et al. (2006). At each step, we have a current vertex \( S \) 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 \( S \). 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 \( Tsd \), 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, \( \text{DegreeQ} \), to pick the seed proteins
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 their weighted degree. Once a cluster is built and output, its proteins are removed from 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(V).

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., 2006) 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 SPIC (Blatt et al., 1996), MCL (Enright et al., 2002), MCODE (Badrawi and Hogue, 2003), RNSC (King et al., 2004), Cfinder (Palla et al., 2005), NetworkBLAST (Sharan et al., 2005), DPClus (Ahituv-U-Amin et al., 2006), MCUPGMA (Loewenstein et al., 2008) and DME (Georgii et al., 2009). We briefly highlight the main features of these algorithms. SPIC 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 interactions, 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 hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical 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 minimum density threshold.

are well connected to other nodes in the cluster and that maintains cluster density. MCUPGMA is a memory-efficient average-link hierarchical clustering algorithm. DME finds all clusters that satisfy a user-defined minimum density threshold.
Table 2. Running time and memory usage of clustering approaches

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Biogrid (Yeast)</th>
<th>Biogrid (Human)</th>
<th>STRING (Yeast)</th>
<th>STRING (Human)</th>
<th>Bayesian (Yeast)</th>
<th>Bayesian (Human)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPICi</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>MCU/GMA</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCL</td>
<td>336</td>
<td>114</td>
<td>645</td>
<td>4926</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>1904</td>
<td>427</td>
<td>7848</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPC</td>
<td>183</td>
<td>215</td>
<td>219</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCODE</td>
<td>101</td>
<td>49</td>
<td>7848</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPClus</td>
<td>1602</td>
<td>2113</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNSC</td>
<td>172</td>
<td>17</td>
<td>1325</td>
<td>23,448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFinder</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DME</td>
<td>Memory</td>
<td>Biogrid (Yeast)</td>
<td>Biogrid (Human)</td>
<td>STRING (Yeast)</td>
<td>STRING (Human)</td>
<td>Bayesian (Yeast)</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
<td>1.5</td>
<td>15.1</td>
<td>90.5</td>
<td>1143.0</td>
<td></td>
</tr>
<tr>
<td>SPICi</td>
<td>259.1</td>
<td>259.1</td>
<td>259.1</td>
<td>259.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCL</td>
<td>73.3</td>
<td>24.9</td>
<td>111.7</td>
<td>357.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>61.9</td>
<td>60.5</td>
<td>72.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPC</td>
<td>220.5</td>
<td>430.3</td>
<td>311.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MCODE</td>
<td>375.6</td>
<td>306.1</td>
<td>606.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPClus</td>
<td>140.2</td>
<td>202.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNSC</td>
<td>25.9</td>
<td>9.8</td>
<td>82.3</td>
<td>349.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CFinder</td>
<td>23.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 MCU/GMA’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, MCU/GMA, 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, MCU/GMA 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, MCU/GMA and MCL

<table>
<thead>
<tr>
<th>Network Algorithm</th>
<th>BP Density</th>
<th>Jaccard</th>
<th>PR</th>
<th>CC Density</th>
<th>Jaccard</th>
<th>PR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biogrid SPICi</td>
<td>0.368</td>
<td>0.214</td>
<td>0.183</td>
<td>0.379</td>
<td>0.167</td>
<td>0.141</td>
</tr>
<tr>
<td>Yeast MCL</td>
<td>0.414</td>
<td>0.200</td>
<td>0.160</td>
<td>0.444</td>
<td>0.147</td>
<td>0.115</td>
</tr>
<tr>
<td>Human MCL</td>
<td>0.284</td>
<td>0.208</td>
<td>0.156</td>
<td>0.324</td>
<td>0.171</td>
<td>0.125</td>
</tr>
<tr>
<td>Biogrid SPICi</td>
<td>0.254</td>
<td>0.183</td>
<td>0.159</td>
<td>0.271</td>
<td>0.097</td>
<td>0.078</td>
</tr>
<tr>
<td>Human MCU/GMA</td>
<td>0.319</td>
<td>0.179</td>
<td>0.150</td>
<td>0.348</td>
<td>0.096</td>
<td>0.074</td>
</tr>
<tr>
<td>MCL</td>
<td>0.348</td>
<td>0.177</td>
<td>0.141</td>
<td>0.388</td>
<td>0.120</td>
<td>0.091</td>
</tr>
</tbody>
</table>

<sup>GO: GO analysis values (see text) for SPICi, MCU/GMA 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.</sup>

For clusters with at most five proteins, MCU/GMA 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 MCU/GMA. 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 MCU/GMA (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.
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 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 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 GM071508 to David Botstein, in part). National Institute of Health Center of Excellence (grant P50 in part); National Institutes of Health (GM076275 to M.S., in part); National Science Foundation (ABI-0850063 to M.S., in part); National Institutes of Health (GM071508 to David Botstein, in part). We thank Curtis Huttenhower and Olga Troyanskaya for providing GM071508 to David Botstein, in part). National Institute of Health Center of Excellence (grant P50 in part); National Institutes of Health (GM076275 to M.S., in part); National Science Foundation (ABI-0850063 to M.S., in part); National Institutes of Health (GM071508 to David Botstein, in part). We thank Curtis Huttenhower and Olga Troyanskaya for providing GM071508 to David Botstein, in part). National Institute of Health Center of Excellence (grant P50 in part); National Institutes of Health (GM076275 to M.S., in part); National Science Foundation (ABI-0850063 to M.S., in part); National Institutes of Health (GM071508 to David Botstein, in part).

REFERENCES