Uncovering Many Views of Biological Networks Using Ensembles of Near-Optimal Partitions

Geet Duggal*
Dept. of Computer Science,
Center for Bioinformatics and Computational Biology
University of Maryland
College Park, MD, 20740
geet@umd.edu

Saket Navlakha*
Dept. of Computer Science,
Center for Bioinformatics and Computational Biology
University of Maryland
College Park, MD, 20740
saket@cs.umd.edu

Michelle Girvan
Dept. of Physics, Institue for Physical Science and Technology, Institute for Research in Electronics and Applied Physics
University of Maryland
College Park, MD, 20740
girvan@umd.edu

Carl Kingsford
Dept. of Computer Science,
Center for Bioinformatics and Computational Biology
University of Maryland
College Park, MD, 20740
carlk@cs.umd.edu

ABSTRACT

Densely interacting regions of biological networks often correspond to functional modules such as protein complexes. Most algorithms proposed to uncover modules, however, produce one clustering that only reveals a single view of how the cell is organized. We describe two new methods to find ensembles of provably near-optimal modularity partitions that lie within a heuristically constrained search space. We also show how to count the number of solutions in this space that exist within a bounded modularity range. We apply our algorithms to a protein interaction network for *S. cerevisiae* and show how fine-grained differences between near-optimal partitions can be used to define robust communities. We also propose a technique to find structurally diverse near-optimal solutions and show that these different partitions are enriched for different biological functions. Our results indicate that near-optimal solutions can represent alternative and complementary views of the network's structure.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*network problems*; H.2.8 [**Database Management**]: Database Applications—*data mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*clustering*

General Terms

Algorithms, Theory

Keywords

Biological networks, Near-optimal clusterings, Modularity

*Equally contributed to this work.

1. INTRODUCTION

Network clustering has become a very popular tool used to understand the structural organization of complex networks. Although the definition of a meaningful cluster can vary across contexts, a network cluster is typically thought of as a set of tightly interacting nodes with relatively few external edges. These sets of nodes correspond to the network's structural organs, which lay the foundation for many processes occurring over these networks. For example, in social and information networks, clustering has been used to uncover dynamic relationships and communities [21, 15], and to understand how knowledge propagates through the network [25, 39]. In computational biology, network clustering has been used to understand and generate hypotheses about how thousands of genes and proteins interact to carry out important biological processes. In protein-protein interaction (PPI) networks, for example, groups of functionally related proteins (such as protein complexes) often appear as dense subgraphs and therefore correspond well with the notion of a network cluster. Automatically uncovering these functional modules can lead to hypotheses about the cellular role of unannotated proteins [5, 31].

Most graph clustering algorithms proposed only produce a single partition in accordance with some objective function [22, 14, 9, 40, 30, 11]. A growing body of research, however, suggests that multiple complementary network clusterings often exist [16, 2, 36]. Recent work has also empirically shown that near-optimal partitions can be useful for

many tasks [29]: in particular, they can be used to differentiate between core and peripheral community members by looking at which nodes more readily change their cluster-membership [13, 26]; they can be used to find robust communities, i.e. sets of nodes that remain co-clustered across many partitions; they can be used to gauge confidence in the optimal solution based on the quality of nearby solutions; and they can be used to mitigate noise in the network [29].

Here, we consider the popular modularity [34, 33] optimization function that has been used in various network domains [38, 24]. We propose a new approach to find provably near-optimal modularity clusterings that can be derived from a heuristically structured search space (for example, the Fast Greedy (FG) hierarchical clustering tree [7]). By adding constraints when selecting successive solutions, we can systematically explore the solution space and can prove that the i^{th} solution found by our algorithm is an i^{th}-best solution that is compatible with the tree. We also show how to count the number of partitions that have a modularity lying within a bounded range given this tree. Our results on a PPI network for the yeast *S. cerevisiae* show that there exist many nearly identical partitions with similar modularity, which confirms the modularity degeneracy problem [17] for this application. Although these strictly near-optimal partitions can be used to identify robust communities, it is also important to quickly sample highly modular *and structurally dissimilar* partitions. We introduce a repulsion term into the modularity optimization function, which allows us to strike a balance between modularity and diversity of solutions. When tested on the PPI network, we find that the diversified partitions are indeed enriched for different biological functions, and therefore represent alternative and complementary views of the network's organization.

2. RELATED WORK

Several algorithms have been proposed to find high modularity network clusterings [34, 33, 7, 1, 35, 11]. Most of these algorithms return only a single clustering, though there are some exceptions: Navlakha and Kingsford [29] extend the integer linear programming formulation of Agarwal et. al. [1] to incorporate diversity constraints into the modularity maximization function. Their approach, however, requires solving an integer program which does not scale to larger networks. Other approaches find multiple modularity solutions using heuristic techniques [27, 18] (such as simulated annealing) which provide no theoretical guarantee about the optimality of the solutions found in polynomial time. This makes it impossible to pinpoint where exactly they lie in the solution landscape with respect to the (potentially many) undiscovered solutions.

Bae and Bailey [2] propose a technique for finding alternative clusterings in hierarchical data structures, but their technique requires that the alternative clusterings have the same number of clusters as the optimal clustering. Moreover, if two nodes are originally in the same cluster, they must not be placed in the same cluster in a subsequent clustering. This latter requirement can be too restrictive in network clustering scenarios when certain sets of nodes (e.g. cliques) should not be disturbed, or if the goal is to look for more subtle differences between solutions.

Gondek and Hofmann [16] use the coordinated conditional information bottleneck to find non-redundant clusterings. However, their approach is probabilistic, not easily adaptable to network clustering problems (such as modularity), and requires the number of clusters to be given *a priori*. Qi and Davidson [36] systematically transform the input data so that the same algorithm applied to the transformed data returns a new clustering. In contrast, we explicitly constrain for quality and diversity in the clustering process itself. Furthermore, like many other approaches, they report only a single alternate solution, whereas we are interested in generating large ensembles.

3. ALGORITHMS

3.1 Generating provably near-optimal modularity partitions

We start with an input network G (e.g. Figure 1A) and a structured reduction of the space of possible partitions in the form of the Fast Greedy (FG) [7] hierarchical clustering of G (Figure 1B). The FG clustering starts with each node in its own cluster, and in each step merges the pair of clusters that yields the largest positive gain in modularity [34]. Once there no longer exists a positive pair, the standard algorithm stops and returns the current set of clusters and its modularity (which ranges from -1 to 1 [4]). We instead construct the entire hierarchical tree decomposition formed by continuing to merge the pair of clusters that yields the least negative modularity.

Each internal tree node x corresponds to a cluster consisting of the leaves of the tree rooted at x (denoted by $L(x)$). $L(\text{Root})$ contains all the network nodes, and for leaves, $L(x) = x$. We define a quality score, $q(x)$, for each tree node x equal to the modularity of the cluster induced by x. In particular:

$$q(x) = \frac{1}{2m} \sum_{u,v \in L(x)} \left(A_{uv} - \frac{k_u k_v}{2m} \right), \tag{1}$$

where A is the adjacency matrix of G, k_u is the degree of node u in G, and m is the number of edges in G [34]. The total modularity Q for a partition is the sum of each cluster's modularity contribution. Any non-overlapping partition of the nodes consistent with the heuristic tree corresponds to selecting a set of nodes in the tree, called a node-cut, such that each leaf has exactly one ancestor among the chosen nodes. Grouping together leaves with the same ancestor in the node-cut K corresponds to a partition of the nodes into $|K|$ non-overlapping clusters. To find an optimal modularity partition from the tree we want to find the node-cut with the highest induced modularity:

$$\operatorname*{argmax}_K \sum_{x \in K} q(x). \tag{2}$$

Despite there being an exponential number of possible node-cuts in the tree, we can find the one representing the highest modularity partition in linear time using dynamic programming [32]. The dynamic programming algorithm considers a tree node x and either chooses it or chooses the best solution of each of the subtrees rooted at the children of x, recursively. In particular, for each node x we set:

$$q^*(x) = \max \begin{cases} q(x) & \text{case I} \\ q^*(x_\ell) + q^*(x_r) & \text{case II.} \end{cases} \tag{3}$$

Here, x_ℓ and x_r indicate the left and right children of x, respectively. When x is a leaf, only case I applies. By back-

tracking from the root node, we can find the nodes chosen in the optimal cut, and can thus produce the optimal modularity clustering compatible with the tree.

To find the second-best tree-compatible solution, the idea is to systematically "rattle" the previous solution so that the subsequent one is forced to have a minimally different modularity. Let the optimal tree-derived partition P_1 consist of r clusters $\{C_1^1, C_1^2, \cdots, C_1^r\}$. To find the next best solution (P_2), we observe that at least one cluster in P_1 must not appear exactly in P_2. That is, at least one cluster of P_1 must be perturbed in some way — either splitting the cluster into multiple clusters, or by merging the cluster with another.

To explore each of these possibilities, we introduce the idea of *forbidden nodes*. Let tree node x be marked as forbidden if x is not allowed to be part of a node-cut (effectively setting $q(x) = -\infty$). By disallowing a node to be part of the next partition, the dynamic programming procedure will be forced to choose the best modification of the partition that does not involve x. To deal with the uncertainty of which of the r clusters to perturb, we try all of them; i.e. we iteratively mark each of the r clusters of P_1 as forbidden, re-cut. Each of the r solutions found by this procedure are placed on a priority queue U, ordered by their modularity. The partition on the top of the queue is a second-best solution and it is dequeued from the priority queue. It is important to note that solutions that were not picked in the current step remain in U because these solutions can emerge later as the next best solution.

Figure 1 shows an example of finding the second-best partition given a network (Figure 1A) and the Fast Greedy hierarchical tree (Figure 1B). Figure 1C–E show the three candidate partitions found by successively marking each cluster in the optimal partition as forbidden and re-cutting the tree using the dynamic programming algorithm. Notice that the candidate partitions are perturbed from the optimal partition in different ways: In Figure 1C,E the next best solution chooses the forbidden node's descendants, whereas in Figure 1D it chooses the node's ancestor (thereby merging two clusters). Amongst these candidates, we choose the one with the highest modularity. In the example, the provably second optimal clustering is the one shown in Figure 1C.

To find a provably i^{th}-best partition, we iterate this process. We successively mark each cluster in P_{i-1} as forbidden ($r, 4, s, t$ in the example when $i = 3$) in addition to marking as forbidden the nodes that were forbidden when finding P_{i-1} (b in the example), and re-cut. This gives us $|P_{i-1}|$ (not necessarily unique) new candidate partitions which are added to U. We then report the solution at the top of U (the one with the highest modularity). This may be one of the $|P_{i-1}|$ solutions generated in this step, or a solution generated in a previous step and which already lies in U.

To find the i^{th} solution our algorithm takes $O(|P_{i-1}|n + |P_{i-1}| \log(|U|))$ time, where n is the number of nodes in the network. The first term corresponds to the number of additional candidates considered, each taking $O(n)$ time to find. The second term corresponds to the time required to add the candidates to the priority queue U.

Figure 1F shows the candidate solutions stored in U for the example. The first row shows that initially no nodes were marked as forbidden, which resulted in partition $\{b, s, t\}$. Rows 2–4 show the three candidates considered in Figure 1C–E. Rows 5–8 and 9–12 shows the additional candidates considered when finding P_3 and P_4, respectively.

Algorithm 1 Modu-Cut(T, s)

```
Partitions ← [ ]  # List of near-optimal partitions
U ← [ ]  # Modularity priority queue
i ← 0
(q, cut-nodes) ← cut(T, {}) # Optimal solution
U ← push(q, cut-nodes, {}) # Add solution to the queue
while i < s do
    (q, cut-nodes, F) ← U.pop() # Next best solution
    P[i] ← cut-nodes # Save the solution

    # Iteratively perturb the solution
    for all u ∈ cut-nodes do
        (q, cut-nodes_u) ← cut(T, F ∪ {u})
        U ← push(q, cut-nodes_u, F ∪ {u})
    end for
    i ← i + 1
end while
return Partitions
```

Note: The function cut$(T, \mathcal{F})$ returns the modularity (q) and partition (cut-nodes) of the best solution in T with nodes in $\mathcal{F}$ marked as forbidden.

Note: For brevity, we omit pseudocode for handling duplicate solutions.

We call this algorithm Modu-Cut. Pseudocode is shown in Algorithm 1 and a proof of its correctness is given in the Appendix. Although we focus on modularity, our technique is applicable to finding near-optimal solutions from any hierarchical tree decomposition where the objective function is decomposable into the sum of each cluster's contribution (e.g. partition stability defined using random walks [8, 23]).

3.2 Counting the number of solutions within a bounded modularity range

The basic dynamic programing algorithm can be applied to count the number of solutions compatible with the clustering tree T that have a modularity within a given range (B_ℓ, B_u). To do this, assume all the node weights are integers. This can be achieved by multiplying the modularity values by a large constant (e.g. $4m^2$ for unweighted networks).

Define a function Count(x, b) to equal the number of node-cuts in the subtree of T rooted at x that have weight equal to b. When x is a leaf x, then Count$(x, b) = \chi(q(x) = b)$, where $\chi(q(x) = b)$ is 1 if $q(x) = b$ and 0 otherwise. When x is not a leaf, we have:

$$\text{Count}(x, b) = \chi(q(x) = b) + \sum_{i=\mu}^{\mu'} \text{Count}(x_\ell, i) \times \text{Count}(x_r, b - i), \tag{4}$$

where x_ℓ and x_r are the left and right children of node x and where μ and μ' are the minimum and maximum possible modularities achievable for any partition of any subtree (computed using the same dynamic programming algorithm from Section 3.1 with min and max, respectively). The right-hand side enumerates all possible ways of dividing up the target modularity (b) between the left and right subtrees. The function Count can be computed starting at the leaves and proceeding up to the root. The number of solutions within a range (B_ℓ, B_u) can be computed by $\sum_{i=B_\ell}^{B_u} \text{Count}(\text{Root}, i)$. The running time is $O(n(\mu' - \mu)^2)$.

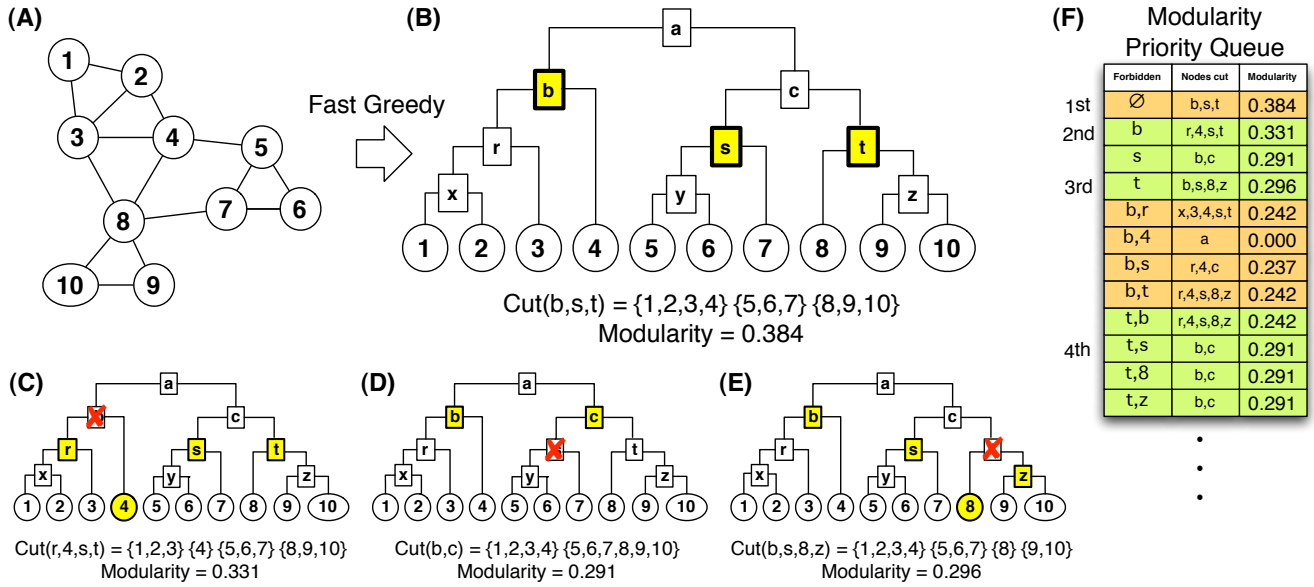

Figure 1: Overview of the MODU-CUT algorithm. (A) The input network. (B) The hierarchical clustering of the network produced by the Fast Greedy algorithm [7]. Bold tree nodes correspond to the optimal modularity node-cut. (C–E) Three candidate near-optimal partitions considered by our method when finding the second-best solution. Each candidate corresponds to marking a different cluster from the optimal clustering as forbidden. Solution (D) has the highest modularity (0.331), which is the provably second best clustering from the tree. (F) The modularity priority queue showing the solutions considered thus far along with the corresponding nodes marked as forbidden.

We call this algorithm MODU-COUNT.

The dependence of the running time on the magnitudes of the modularities is unavoidable unless P = NP without exploiting additional properties of the modularity function. This follows by reducing the SUBSET SUM problem to our problem, TREE WEIGHTED CUT COUNT: counting the number of cuts of a given value in a binary, node-weighted tree. Given an instance "$\{x_1, \ldots, x_N\}$, $B > 0$" of SUBSET SUM, create an instance of TREE WEIGHTED CUT COUNT with $2N$ leaves as shown in Figure 2. (Although this is not a binary tree, it can be converted into one by replacing the root node with a large enough binary tree.) In the constructed tree, every internal node u has weight $q(u) = 0$, and half the leaf nodes have $q(u) = 0$. This tree has a node-cut of weight B if and only if there is a subset of $\{x_1, \ldots, x_N\}$ that sums to B, solving the SUBSET SUM problem. Hence, the general TREE WEIGHTED CUT COUNT problem is NP-hard, and the pseudopolynomial MODU-COUNT algorithm is justified. For the special case of a simple, unweighted network, the multiplicative constant of $4m^2$ results in integers for all possible partitions of a network. Since m can never be greater than n^2, and the number of nodes in the tree is $2n - 1$, MODU-COUNT runs in polynomial time for simple, unweighted networks.

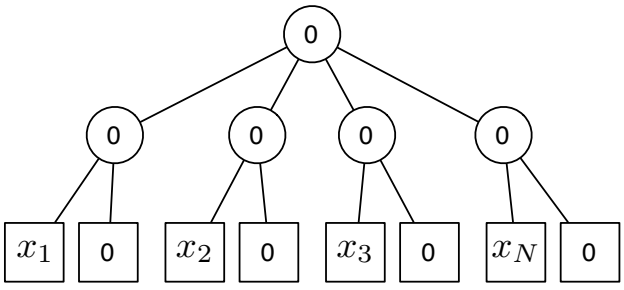

Figure 2: The instance of Tree Weighted Cut Count used in the reduction from Subset Sum.

3.3 Constraining for more diverse solutions

If there are many near-optimal solutions to wade through, the techniques of Section 3.1 can only be used to track minute differences between near-optimal partitions. Although these differences can be exploited to uncover subtle structural differences between solutions [29], it would be difficult to use this technique to find highly modular *and highly different* solutions without slowly traversing the entire partition-space. Finding diverse partitions, however, is important because they offer distinct views of the network's structural composition. In this section, we explicitly constrain for diversity of partitions by introducing a repulsion term to Equation (1) that biases against solutions that are too similar to the optimal solution, P_1.

To define the distance between two solutions or partitions, we use the variation of information (VI) [28]. We use this measure, among the many others proposed to compare clusterings [28], because it is information-theoretic, a true metric in partition-space (non-negative, symmetric, obeys the triangle inequality), and importantly, can be written such that the total distance between two partitions is the sum of each

cluster's contribution. It is defined as follows:

$$\mathrm{VI}(P_1, P_K) \doteq 2H(P_1, P_K) - H(P_1) - H(P_K), \quad (5)$$

where $H(P_K)$ and $H(P_1)$ are the entropies associated with partitions P_K and P_1 (represented as random variables of cluster assignment), and $H(P_1, P_K)$ is the joint entropy of P_K and P_1. The measure ranges from 0 (identical partitions) to $\log(n)$, where n is the number of nodes in the network. Because VI is decomposable (like modularity), we can modify the function $q(x)$ in Equation (1) to include x's contribution towards the VI distance from P_1 to the cut-induced clustering P_K. Over all cuts K, $H(P_1)$ remains constant and therefore we can ignore it. A cluster x contributes

$$\text{Node-VI}(x) \doteq p(x)\log p(x) - 2\sum_{C \in P_1} p(x, C)\log p(x, C) \quad (6)$$

towards the total VI distance, where $p(x) = L(x)/n$, which is the percentage of total nodes in cluster x, and $p(x, C) = |L(x) \cap C|/n$, which is the fraction of total nodes in cluster $L(x)$ that are also in cluster C. The two terms of Equation (6) correspond to x's contribution to the overall entropy of P_K, and x's contribution to the joint entropy of P_1 and P_K, respectively. We incorporate the node's VI contribution into Equation (1) as:

$$q'(x) = q(x) + \alpha\text{Node-VI}(x)/\log(n), \quad (7)$$

where $q(x)$ is the modularity contribution of tree node x as given in Equation (1), and where α is a parameter that governs how much weight we place on the VI term. The Node-VI term acts as a repulsion factor that pushes away from solutions that are too similar to P_1. We normalize the VI value by its maximum ($\log(n)$) so that the modularity and VI measures lie on roughly the same scale.

We replace the $q(x)$ values on the tree with these $q'(x)$ values and use the dynamic programming algorithm of Equation (3) to find the cut with the highest weight. For any node-cut K, we have:

$$\sum_{x \in K} q'(x) = Q(P_K) + \frac{\alpha}{\log(n)}\mathrm{VI}(P_1, P_K). \quad (8)$$

That is, the weight of a node-cut K equals the modularity of P_K plus the the normalized VI distance of P_K to P_1. To generate an ensemble of diverse solutions, we find a single solution at each value of α and iterate for varying values of α, ignoring all previous solutions found. Each solution found is guaranteed to be one compatible with the tree that is optimal according to the chosen α trade-off. Though not guaranteed, larger α values will typically yield solutions that are more distinct from the optimal solution.

Equation (7) offers one approach to balance between the magnitude of modularity of P_K and its amount of difference from P_1. An alternative formulation would be to use hard constraints that forbids solutions that are too similar to previous solutions. Here, we use soft constraints so that the algorithm naturally uncovers a trade-off between the two optimization criteria.

We call this algorithm MODU-MIX.

4. EXPERIMENTS

We tested our algorithms on a high-confidence protein-protein interaction (PPI) network for the yeast *S. cerevisiae*, whose largest connected component contains 1,647 proteins and 2,518 interactions [41]. First, we show how our MODU-CUT algorithm (Section 3.1) discovers many similar near-optimal solutions, which can be used in conjunction to differentiate between robust and non-robust communities. Second, we empirically quantify the modularity degeneracy problem [17] by counting the number of partitions with a modularity within a bounded modularity range using our MODU-COUNT algorithm (Section 3.2). Third, we incorporate a dissimilarity term into the modularity maximization function to uncover ensembles of high-quality and diverse partitions using our MODU-MIX algorithm (Section 3.3). All experiments were run on 2.4 Ghz machine with 4 GB of RAM.

4.1 Near-optimal PPI network partitions

We ran MODU-CUT (Section 3.1) on the largest component of the yeast PPI network. It took 6.4 minutes to generate the first 300 provably near-optimal tree-compatible solutions. Figure 3A shows how slowly the modularity of near-optimal solutions decreases with respect to the optimal partition (P_1). P_{300} has a modularity that is only 0.00019 less than the optimal (0.73884 vs. 0.73865), suggesting extreme uncertainty in the optimal solution. These solutions are very structurally similar: Figure 3B plots the normalized variation of information [28] distance between P_1 and P_i, $1 \leq i \leq 300$. The VI distance between the near-optimal solutions and the optimal does not necessarily monotonically increase because, from solution to solution, the perturbed cluster can be of different size and can be either split or merged, each yielding different changes in VI. However, there is a general trend of increasing VI distance with subsequent solutions, as expected (R-value of 0.47).

Subtle differences in near-optimal solutions can be used to differentiate between robust and non-robust communities. Figure 4 shows the percentage of times each cluster in P_1 appears exactly in the 300 near-optimal partitions (Jaccard coefficient of 1). Clusters that remain exactly intact are suggestive of a resilient community that has withstood the pressure of additional constraints. The yellow bars of the plot highlight the clusters that remain unperturbed in 95% of the near-optimal partitions. The non-robust clusters (green bars) are likely to contain many nodes that are peripherally connected to the core members of the cluster. Such dynamics would not be captured by any single-solution approach.

4.2 Counting the number of solutions

To understand how partitions of various quality are globally distributed in the modularity landscape, we used our MODU-COUNT algorithm (Section 3.2) to count the number of solutions in the yeast PPI that lie within a given range of modularity. It took less than 2 minutes to count the number of solutions with multiplicative factor 1000. Figure 5 experimentally confirms the modularity degeneracy problem [17], which claims that there can be an exponential number number of equally valid, high modularity partitions. In particular, we find that there are exactly 4,068,367,271,231,892,000,117,969,958,274 (i.e. $\sim 4 \times 10^{30}$) tree-based partitions with modularity between 0.700 and 0.739 (the maximum modularity). Figure 5 also shows that the degeneracy increases with lower modularity values before decreasing again near the very non-modular solutions.

We contrasted the shape of the modularity-count curve for the PPI network with the curve for a randomly rewired

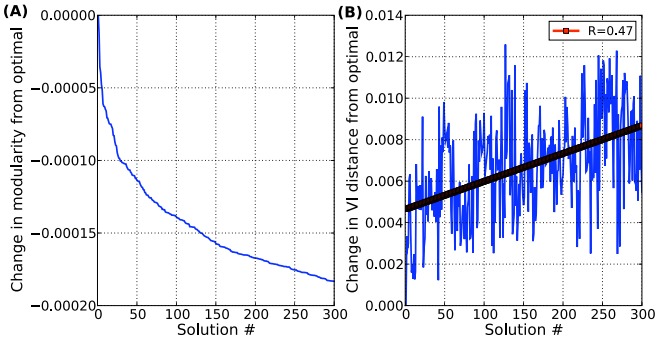

Figure 3: Comparison of how modularity and VI distances change across the modularity landscape for the MODU-CUT algorithm. (A) Modularity slowly decreases with each subsequent near-optimal solution. (B) The near-optimal solutions are also structurally very similar to the optimal solution, measured using the VI distance.

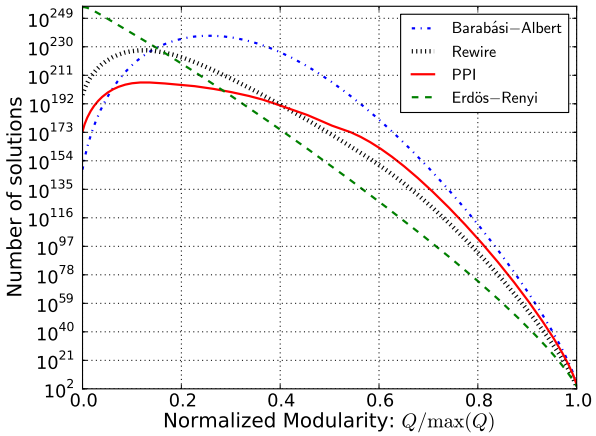

Figure 5: The MODU-COUNT algorithm applied with a multiplicative constant of 1000 for four networks of the same size and similar density: Barabási-Albert ($\gamma = 1$), Rewire, PPI, and Erdös-Renyi ($p = 0.0025$). The optimal modularity for the PPI network corresponds to $> 30,000$ different solutions all with modularity 0.738, and this number increases significantly as the modularity decreases. To contrast, there are only a few thousand solutions at the optimal value for the Erdös-Renyi random graph, and < 1000 solutions at the optimal value for the rewired graph.

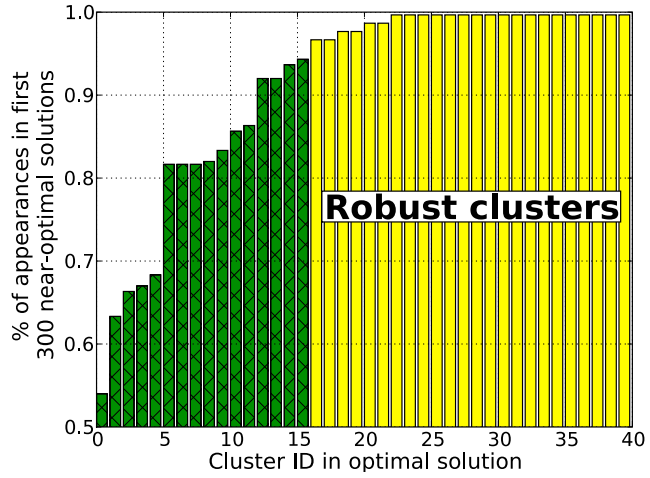

Figure 4: Near-optimal solutions can be used to differentiate between robust and non-robust communities. The x-axis shows the cluster ID of each cluster in the optimal solution. The y-axis shows the percentage of the first 300 near-optimal solutions that contain the cluster exactly. The yellow bars highlight the clusters which exactly appear in $\geq 95\%$ of the near-optimal partitions.

network ("Rewire" in Figure 5), a Barabási-Albert (BA) [3] random network, and an Erdös-Renyi (ER) random network with the same number of nodes. The parameters ($p = 0.0025$ for ER and $\gamma = 1$ for BA) were chosen to produce random networks with density similar to the PPI. The rewired network's curve is similar to the PPI network's curve, yet at the optimal modularity only 768 solutions exist compared to 32,256 solutions for the PPI network. The ER network has a much lower maximum modularity due to its inherently random nature of adding links and has 4,032 solutions at maximum modularity. Interestingly, the BA network's curve is similar to the PPI network's, which is additional evidence of the coherence of the BA model with PPI networks [10]; it also suggests that modularity degeneracy is not intrinsic to the PPI network alone.

These results suggest that, for even one heuristically constrained search structure (the FG tree), the modularity landscape offers a plethora of reasonable solutions and that full confidence should not be placed in a single partition.

4.3 Constraining for modularity and diversity

Finally, we ran our MODU-MIX algorithm on the PPI network to try and discover a mélange of non-redundant partitions. Following the procedure outlined in Section 3.3, we added a repulsion term — the variation of information — to the quality score of each tree node as a means to penalize solutions that are similar to the optimal. The level of diversity is governed by a single paramter, α, which we varied from 0.0 to 3.0 in steps of 0.2. Each value yielded a different partition. It took 0.1 minutes to generate the 15 total partitions. Figure 6 presents the change in modularity and VI of each solution with respect to the optimal solution.

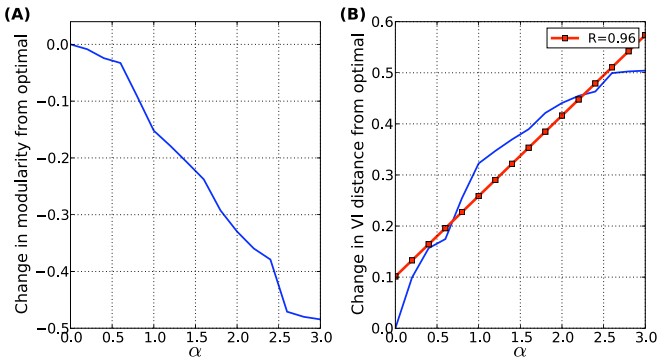

Figure 6: For the MODU-MIX algorithm: (A) Each α value corresponds to a different solution. Modularity decreases quickly with increasing diversity constraints (α). (B) Increasing α also yields structurally diverse solutions. Taken together, with only a 10% drop-off in modularity, we can find a new solution that is 25% different structurally ($\alpha = 0.8$).

The curve decays much more rapidly than the corresponding plot for the MODU-CUT algorithm (Figure 3A), which shows that we can find diverse and reasonable solutions much more quickly than before. In particular, with only a 10% drop-off in modularity, we can get a solution that is 25% different structurally ($\alpha = 0.8$).

A common method to gauge the value of a biological network partition is to test each of its clusters for functional enrichment. Clusters consist of nodes (proteins) that engage in one or more biological functions in the cell, such as protein synthesis, regulation of metabolism, or transcription [19]. Sets of nodes engaged in a common function often appear close together in PPI networks [6, 37, 20].

To evaluate the biological usefulness of our multiple partitions, we tested how well the clusters found in each partition match known biological functions. This is a useful task because the function of many proteins, even in well-studied species such as yeast, remains unknown. As a result, computational approaches have flourished as a means to transfer annotations (association with a biological function) to an unannotated protein based on the annotations of its co-clustered neighbors [5, 31] in the PPI network. To gauge each partition's usefulness for this task, we test each of its clusters for enrichment of MIPS functions [19] using the hypergeometric test (strict p-value $< 10^{-7}$), ignoring small clusters with < 3 proteins and correcting for multiple hypotheses. With this test, a cluster is said to be enriched for a biological function f if the probability of obtaining the observed intersection size (or greater) between the set of genes in the cluster and the set of genes known to be associated with f is less than 10^{-7}.

Table 1 shows how the multiple partitions found by our MODU-MIX algorithm provide different views of the PPI network's functional organization. Each value of α corresponds to a single solution derived using Equation (7). Enriched(α) corresponds to the number of annotations enriched in the solution. Cumulative(α) corresponds to the total number of unique enriched annotations in all solutions seen up to that point. Interestingly, the optimal modularity partition ($\alpha = 0$) is enriched for only 3 protein functions. The solution at $\alpha = 0.2$ is enriched for four annotations, two of which (endocytosis and general transcription activities) were new relative to the enriched annotations in the optimal solution (lysosomal and vacuolar protein degradation, autoproteolytic processing, and cyclic nucleotide binding). The first several solutions each offer only a few enriched annotations; however, at $\alpha = 0.8$ we see a spike in the number of enriched functions. This suggests that the algorithm has found a new, perhaps more relevant, region in the modularity landscape that has high modularity and that corresponds better to the biological processes underlying the network. This space is marked with a greater number of smaller-sized modules, chosen by selecting more clusters lower in the tree. We also observe that Cumulative(α) levels off as α increases further and that this happens when the modularity of the network is quite low. However, by itself, the single solution at $\alpha = 3.0$ has more enriched functions than the optimal modularity partition. Taken together, this ensemble of solutions provides meaningful and diverse views of how proteins form clusters to carry out fundamental cellular tasks.

α	Q	VI	Enriched(α)	Cumulative(α)
0.0	0.7388	0.000	3	3
0.2	0.7304	0.099	4	5
0.4	0.7146	0.157	2	6
0.6	0.7061	0.174	3	7
0.8	0.6474	0.255	13	18
1.0	0.5868	0.323	24	27
1.2	0.5599	0.347	29	32
1.4	0.5309	0.369	32	35
1.6	0.5011	0.390	35	37
1.8	0.4465	0.421	37	39
2.0	0.4091	0.441	36	39
2.2	0.3794	0.455	38	42
2.4	0.3598	0.463	40	43
2.6	0.2679	0.499	10	43
2.8	0.2590	0.503	9	43
3.0	0.2544	0.504	11	44

Table 1: Ensembles of partitions can reveal multiple views of biological networks. The first column is the α value used in Equation 7. The second and third columns contain the modularity of the solution and its VI distance from the optimal, respectively. The fourth column, Enriched(α), is the number of functions enriched in the solution. The fifth column, Cumulative(α), is a cumulative sum of the number of *unique* functions enriched in all previous solutions. For example, the solution at $\alpha = 0.8$ was enriched for 13 annotations, 11 of which were not enriched in any of the previous solutions ($\alpha < 0.8$).

5. CONCLUSIONS

We described two new methods to find ensembles of highly modular tree-derived network partitions using a modified modularity maximization function. The proposed MODU-CUT algorithm can be used to enumerate provably near-optimal partitions that are in agreement with a heuristically constructed tree. These partitions were used collectively to distinguish between robust and non-robust communities. We showed how to count the number of solutions within a bounded modularity range (MODU-COUNT) and found an astronomical number of highly modular solu-

tions in the yeast PPI, reconfirming and further quantifying the modularity degeneracy problem. To quickly uncover highly modular and highly different partitions, we incorporated diversity constraints (MODU-MIX) and showed how the resulting partitions were enriched for different biological annotations, which suggests that alternative clusterings may be useful for improved function prediction. Our results suggest that these algorithms can produce ensembles of partitions that offer alternative and complementary views of the network's structure.

As future work, it would be interesting to experiment with hard constraints (instead of soft constraints) as an alternative way to forcefully uncover different partitions. We would also like to experiment with other structured search spaces, including alternative hierarchical clustering trees that correspond to different regions of the clustering space.

Further, additional approaches for assessing the biological quality of various near-optimal partitions need to be developed. While the hypergeometric test used here is a standard procedure used when considering function enrichment [12], we are unaware of a statistical test of cluster enrichment *given* a partition and the development of such a test would be interesting future work.

6. AVAILABILITY

The implementation of our algorithms is open source and available online at `http://www.cbcb.umd.edu/ModuTree`.

7. ACKNOWLEDGMENTS

C.K. thanks the National Science Foundation for grants 0849899 and 0812111.

8. REFERENCES

[1] G. Agarwal and D. Kempe. Modularity-maximizing graph communities via mathematical programming. *Eur. Phys. J. B*, 66(3):409–418, 2008.

[2] E. Bae and J. Bailey. COALA: a novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. *IEEE International Conference on Data Mining*, 0:53–62, 2006.

[3] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[4] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. *IEEE T. Knowl. Data En.*, 20(2):172–188, 2008.

[5] S. Brohee and J. van Helden. Evaluation of clustering algorithms for protein-protein interac tion networks. *BMC Bioinformatics*, 7:488+, November 2006.

[6] D. Bu et al. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucleic Acids Research*, 31(9):2443–2450, May 2003.

[7] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6 Pt 2):066111, 2004.

[8] J. C. Delvenne, S. N. Yaliraki, and M. Barahona. Stability of graph communities across time scales. *ArXiv*, physics.soc-ph(0812.1811), Mar 2009.

[9] I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD '05: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 629–634, 2005.

[10] E. Eisenberg and E. Y. Levanon. Preferential attachment in the protein network evolution. *Phys Rev Lett*, 91(13):138701, 2003.

[11] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010.

[12] W. Fury, F. Batliwalla, P. Gregersen, and W. Li. Overlapping probabilities of top ranking gene lists, hypergeometric distribution, and stringency of gene selection criterion. *Conf Proc IEEE Eng Med Biol Soc*, pages 5531–5534, 2006.

[13] A. Gavin, P. Aloy, P. Grandi, R. Krause, M. Boesche, M. Marzioch, C. Rau, L. Jensen, S. Bastuck, B. Dümpelfeld, et al. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084):631–636, 2006.

[14] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(12):7821–7826, June 2002.

[15] J. Golbeck. The dynamics of web-based social networks: Membership, relationships, and change. *First Monday*, 12(11), 2007.

[16] D. Gondek and T. Hofmann. Non-redundant data clustering. *IEEE International Conference on Data Mining*, 0:75–82, 2004.

[17] B. H. Good, Y. A. de Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106+, Apr 2010.

[18] R. Guimerà and L. A. Nunes Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

[19] U. Guldener et al. CYGD: the comprehensive yeast genome database. *Nucleic Acids Res.*, 33(Suppl. 1):D364+, 2005.

[20] J.-D. J. Han et al. Evidence for dynamically organized modularity in the yeast prot ein-protein interaction network. *Nature*, 430(6995):88–93, July 2004.

[21] J. Hopcroft, O. Khan, B. Kulis, and B. Selman. Tracking evolving communities in large linked networks. *Proc. Natl. Acad. Sci. USA*, 101 Suppl 1:5249–5253, 2004.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[23] R. Lambiotte, J. C. Delvenne, and M. Barahona. Laplacian dynamics and multiscale modular structure in networks. *ArXiv*, physics.soc-ph(0812.1770), Oct 2009.

[24] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 631–640, New York, NY, USA, 2010. ACM.

[25] J. Leskovec, M. McGlohon, C. Faloutsos, N. G. lance, and M. Hurst. Cascading behavior in large blog graphs: Patterns and a model. In *Proceedings of SDM*, 2007.

[26] H. C. Leung, Q. Xiang, S. M. Yiu, and F. Y. Chin. Predicting protein complexes from PPI data: a core-attachment approach. *J. Comp. Biol.*,

16(2):133–144, 2009.

[27] C. P. Massen and J. P. Doye. Identifying communities within energy landscapes. *Phys. Rev. E*, 71(4 Pt 2):046101, 2005.

[28] M. Meilă. Comparing clusterings—an information based distance. *J. Multivariate Anal.*, 98(5):873–895, 2007.

[29] S. Navlakha and C. Kingsford. Exploring Biological Network Dynamics with Ensembles of Graph Partitions. In *Proceedings of the PSB Pacific Symposium on Biocomputing*, volume 15, pages 166–177, 2010.

[30] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD International Conference on Management of data*, pages 419–432, New York, NY, USA, 2008. ACM.

[31] S. Navlakha, M. C. Schatz, and C. Kingsford. Revealing biological modules via graph summarization. *J. Comp. Biol.*, 16(2):253–264, 2009.

[32] S. Navlakha, J. White, N. Nagarajan, M. Pop, and C. Kingsford. Finding biologically accurate clusterings in hierarchical tree decompositions using the variation of information. In *Proceedings of RECOMB*, volume 5541, pages 400–417, 2009.

[33] M. E. J. Newman. Modularity and community structure in networks. *Proc. Natl. Acad. Sci. USA*, 103(23):8577–8582, June 2006.

[34] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.

[35] M. A. Porter, J.-P. Onnela, and P. J. Mucha. Communities in networks. *Social Science Research Network Working Paper Series*, March 2009.

[36] Z. Qi and I. Davidson. A principled and flexible framework for finding alternative clustering s. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conferen ce on Knowledge Discovery and Data Mining*, pages 717–726, New York, NY, USA, 2009. ACM.

[37] A. W. Rives and T. Galitski. Modular organization of cellular networks. *Proc. Natl. Acad. Sci. USA*, 100(3):1128–1133, February 2003.

[38] J. Song and M. Singh. How and when should interactome-derived clusters be used to predict functional modules and protein function? *Bioinformatics*, 25(23):3143–3150, 2009.

[39] C. Tantipathananandh and T. Berger-Wolf. Constant-factor approximation algorithms for identifying dynamic communities. In *KDD '09: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 827–836, 2009.

[40] S. van Dongen. Graph clustering via a discrete uncoupling process. *SIAM J. on Matrix Analysis and Applications*, 30(1):121–141, 2008.

[41] H. Yu et al. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110, October 2008.

APPENDIX

The following is a sketch of the proof of correctness of Algorithm 1. For simplicity, we ignore the complication of handling ties in this sketch. Let $C(F)$ denote the set of solutions compatible with the tree that do not use any nodes in the set F. Let $\mathcal{F}(P)$ be the set of forbidden nodes in effect when P was added to the priority queue U. Let $\text{Expand}(P) = \{\mathcal{F}(P) \cup \{x\} : x \in P\}$ for any node-cut P.

LEMMA 1. $\bigcup_{F \in \text{Expand}(P_{i+1})} C(F) = C(\mathcal{F}(P_{i+1})) \backslash \{P_{i+1}\}$.

PROOF. The left-hand side is a subset of the right-hand side because $C(\mathcal{F}(P_{i+1}) \cup \{x\}) \subseteq C(\mathcal{F}(P_{i+1}))$ with the caveat that P_{i+1} can not exist in $C(\mathcal{F}(P_{i+1}))$ by definition. Let Y be any solution in $C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\}$. Because $Y \neq P_{i+1}$, there must exist some cluster $x' \in P_{i+1}$ such that $x' \notin Y$. That means $Y \in C(\mathcal{F}(P_{i+1}) \cup \{x'\})$, which corresponds exactly to one of the $C(F)$ terms on the left-hand side. □

Let $\mathcal{S}_{<Q(P_i)}$ be the set of all solutions with modularity strictly less than $Q(P_i)$. Let $\mathcal{L}(U^i)$ be the set of forbidden node sets in U after the i^{th} step of the algorithm. The next theorem shows that after each iteration, the set of solutions considered by the algorithm includes the next-best solution.

THEOREM 1. $\bigcup_{F \in \mathcal{L}(U^i)} C(F) = \mathcal{S}_{<Q(P_i)}$. *In other words, the set of solutions that are compatible with the set of forbidden nodes in the queue U^i is equivalent to the set of all solutions with modularity $< Q(P_i)$.*

PROOF BY INDUCTION.

Base case: $i = 1$. Let $\mathcal{X} = \bigcup_{x \in P_1} C(\{x\})$, which is the left-hand side of the theorem statement when $i = 1$. Let partition $Z \notin \mathcal{X}$. Z must not be in any $C(\{x\})$ in the union defining $\mathcal{X}$. Hence, for all clusters $x \in P_1$, we have $x \in Z$, and therefore $Z = P_1$.

Induction step: We need to show that $\bigcup_{F \in \mathcal{L}(U^{i+1})} C(F) = \mathcal{S}_{<Q(P_{i+1})}$. After processing P_i, the MODU-CUT algorithm proceeds by removing $\mathcal{F}(P_i)$ from the priority queue, and add the solutions associated with $\text{Expand}(P_i)$ to U.

$\bigcup_{F \in \mathcal{L}(U^{i+1})} C(F)$ can be rewritten as:

$$\left(\bigcup_{F \in \mathcal{L}(U^i) \backslash \{\mathcal{F}(P_{i+1})\}} C(F) \right) \cup \left(\bigcup_{F \in \text{Expand}(P_{i+1})} C(F) \right)$$

Applying Lemma 1 and substituting, we can replace the second term with $C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\}$:

$$\left(\bigcup_{F \in \mathcal{L}(U^i) \backslash \{\mathcal{F}(P_{i+1})\}} C(F) \right) \cup C(\mathcal{F}(P_{i+1})) \setminus \{P_{i+1}\} \quad (9)$$

The above expression equals $\left(\bigcup_{F \in \mathcal{L}(U^i)} C(F)\right) \backslash \{P_{i+1}\}$. Applying the induction hypothesis, this equals $\mathcal{S}_{<Q(P_i)} \backslash \{P_{i+1}\}$. By definition, this in turn equals $\mathcal{S}_{<Q(P_{i+1})}$. □

The correctness of the algorithm follows immediately from Theorem 1. We know that the $(i+1)^{\text{st}}$ solution P_{i+1} is in $\mathcal{S}_{<Q(P_i)}$. By Theorem 1, $P_{i+1} \in C(F')$ for some $F' \in \mathcal{L}(U^i)$. Since the heap contains the best solution compatible with every $F \in \mathcal{L}(U^i)$, the heap will contain the solution P_{i+1}.