2011 IEEE International Parallel & Distributed Processing Symposium

Sensitive and Specific Identification of Protein Complexes in "Perturbed" Protein Interaction Networks from Noisy Pull-Down Data

William Hendrix
North Carolina State University
Raleigh, NC 27695

Tatiana Karpinets
Oak Ridge National Laboratory
Oak Ridge, TN 37831

Byung-Hoon Park
Oak Ridge National Laboratory
Oak Ridge, TN 37831

Eric Schendel
North Carolina State University
Raleigh, NC 27695

Alok Choudhary
Northwestern University
Evanston, IL 60208

Nagiza F. Samatova
North Carolina State University, Raleigh, NC 27695
Oak Ridge National Laboratory, Oak Ridge, TN 37831
Corresponding author: samatovan@ornl.gov

***Abstract*—High-throughput mass-spectrometry technology has enabled genome-scale discovery of protein-protein interactions. Yet, computational inference of protein interaction networks and their functional modules from large-scale pull-down data is challenging. Over-expressed or "sticky" bait is not specific; it generates numerous false positives. This "curse" of the technique is also its "blessing"—the sticky bait can pull-down interacting components of other complexes, thus increase sensitivity. Finding optimal trade-offs between coverage and accuracy requires tuning multiple "knobs," i.e., method parameters. Each selection leads to a putative network, where each network in the set of "perturbed" networks differs from the others by a few added or removed edges. Identification of functional modules in such networks is often based on graph-theoretical methods such as maximal clique enumeration. Due to the NP-hard nature of the latter, the number of tunings to explore is limited. This paper presents an efficient iterative framework for sensitive and specific detection of protein complexes from noisy protein interaction data.**

***Keywords*-protein interaction networks; protein complexes; proteomics; systems biology; parallel algorithms; cliques; databases**

I. Introduction

High-throughput mass-spectrometry (MS) pull-down technology is becoming popular for revealing protein complexes and for assigning functions to unknown proteins. In affinity purification experiments, also referred as pull-down experiments or affinity isolation experiments, the bait is first expressed with an affinity tag. The proteins that interact with these baits, referred to as preys, are identified due to their binding to a particular bait protein. These complexes are isolated, purified, and analyzed by mass spectrometry (MS) or other methods [10].

Large-scale affinity-based pull-down experiments have high sensitivity in protein complex identification, yet they may generate numerous false positive protein-protein interactions (sometimes more than 50%) [7], [8]. A high false positive rate results from over-expression of baits over their natural level [11], which increases the number of "contaminating" proteins [7] and, thus, increases false positive protein-protein interactions. Such spurious interactions make identification of protein complexes by proteomics methods alone challenging [13].

Aiming at reducing false positive interactions, current proteomics methods require rigorous statistical criteria for revealing specific bait-prey and prey-prey pairs. Prey-prey interactions are typically ignored because of the high rate of false positive interactions among them [7], [9]. This results in decreased coverage of the identified interactions, or increased false negative interactions. The trade-off between the coverage (or sensitivity) and the accuracy (or specificity) is inherent to almost any computational proteomics method for protein-protein interactions [7]. By tuning method parameters, through multiple trial-and-error experiments, one can change the balance between specificity and sensitivity, but it is yet difficult, if possible, to significantly improve both.

In this paper, we propose a computational framework that allows for more accurate and efficient identification of protein complexes from large-scale MS pull-down experiments. The improved performance is due to its following foci. First, we augment relatively

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.189

noisy proteomics pull-down data with genomic-context information, such as the structure of bacterial operons and the gene fusion events in eukaryotic genomes. Observing such "events" concurrently with pull-down experiments likely signifies the confidence that the interaction is native. Specifically, it is doubtful that, by chance, the two proteins in multiple pull-down experiments will also (a) belong to the same operon or (b) be found as a single fused chain across some eukaryotic organisms.

Second, we propose a graph theory–based technique for discovery of protein complexes in a network of putative protein-protein interactions derived by both proteomics and genomics methods above. It initially enumerates maximal cliques to reveal highly close relations among three or more proteins. Since cliques impose the stringent (pairwise) interactivity constraint, some experimental noise is filtered out. Some of the native interactions can still be missed due to predefined cut-offs for method parameters and limitations of computational or experimental methods. This may result in smaller, but overlapping, cliques. To identify putative complexes, we introduce an iterative clique merging procedure based on the overlap among the cliques. Some overlap will still remain to allow for proteins to be part of more than one protein complex. While revealing biologically relevant complexes, this procedure is computationally-intensive for large networks.

Third, we further improve the efficiency of the entire end-to-end pipeline by proposing a parallel index-based algorithm for the enumeration of maximal cliques in "perturbed" networks. Our assumption is that an iterative tuning procedure generates a set of "perturbed" networks; each differs from the others by a few added or removed protein interactions. Tuning may involve assigning different cut-offs to method parameters or engaging various genomic-context. Since the "backbone" structure of the network likely remains the same across the "perturbed" networks, the question is whether the cliques discovered during the first iteration could be indexed and re-used for answering queries about the changes in the cliques structure in response to perturbations. We achieve effective parallelization by developing novel theory to decompose the problem into independent operations and dividing index access among the processors.

Finally, we apply the proposed framework to discover genome-scale protein complexes from high-throughput affinity isolation experiments with *Rhodopseudomonas palustris* bacteria grown under photoheterotrophic conditions. Most identified complexes showed high functional homogeneity indicating their biological relevance; they represent a set of comprehensive biological processes in *R. palustris* under studied conditions and reveal novel functions of the organism.

II. Method

Though we have developed a complete end-to-end pipeline for the identification of protein complexes (see details in [19]), this work primarily focuses on the computational problem of *updating* the maximal clique enumeration in response to a perturbation. We briefly describe the other steps in this pipeline below.

A. Iterative end-to-end pipeline for discovery of protein complexes

The main steps of the proposed framework (Figure 1) include: (1) building protein affinity network by fusing putative specifically interacting prey-prey and bait-prey pairs from proteomics-based and genomic-context filtering, (2) discovering protein complexes in the putative protein affinity network utilizing parallel graph-theoretical algorithms, and (3) tuning the identified protein complexes by employing parallel database-assisted graph-theoretical algorithms.

B. Fusing pull-down and genomics information to build networks

1) Predicting bait-prey and prey-prey interactions from pull-down data: We estimate the probability (*p-score*) of ***bait-prey*** binding by capturing background (non-specific) binding behaviors for the bait and the prey. For the prey background, the bait-prey spectrum counts are normalized by their average among all baits. The frequency with which the prey is found at a particular spectrum is plotted against the spectrum count. This plot represents the prey background binding behavior. A similar plot is obtained for each bait. For an observed bait-prey pair, the area under the prey background distribution curve to the right of the observed spectrum estimates the probability of observing by chance a spectrum count larger than the reported spectrum for the pair. The bait background

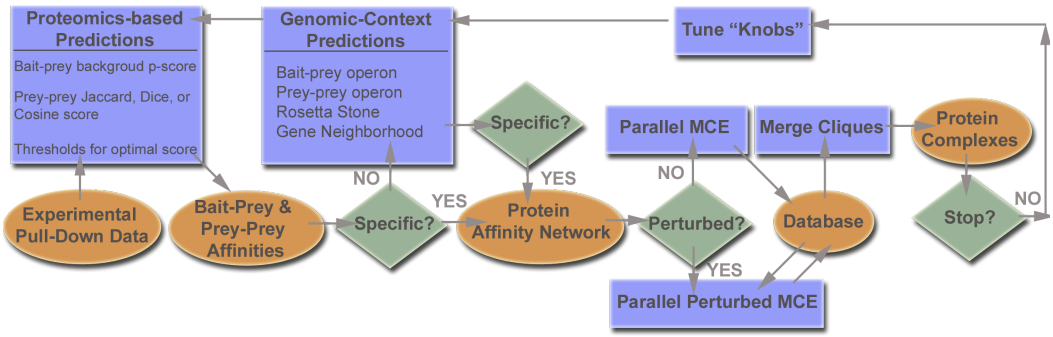

Figure 1: Iterative end-to-end framework for identification of protein complexes.

probability is similarly calculated using the bait background distribution curve. The product of the prey and bait background probabilities represents the *p-score* for a bait-prey binding event with the observed count.

For predicting a ***prey-prey*** interaction, we assume that two preys in the same protein complex are likely to be pulled down by baits together. Thus, we compare purification profiles of all prey pairs and select those that have high similarities in their profiles. A *purification profile* of a prey is a 0-1 vector given all baits in the experiments as its dimensions. The similarity of purification profiles of two preys is computed by correlating their vectors. The Jaccard, cosine and Dice scores are compared to quantify the prey-prey binding affinity in the experiment. Optimal thresholds for the *p-score* and *purification profile similarity score* are found by evaluating the prey-prey pairs against the Validation Table of known interactions. We eliminate pairs that do not meet the thresholds. We compute precision, recall, and F1-measure using the remaining pairs against the validation data. Evaluation iterates until optimal values are found.

2) Predicting protein-protein interactions from genomic-context: We consider the following genomic-context criteria to augment the bait-prey and prey-prey interactions from pull-downs: *Bait-prey operon*, *Prey-prey operon*, *Rosetta Stone*, and *Gene neighborhood*. For *Bait-prey operon*, a bait-prey pair is specifically interacting if it is transcribed from the same operon. For *Prey-prey operon*, a prey-prey pair is specifically interacting if it is transcribed from the same operon and is pulled down by the same bait. For *Gene neighborhood* and *Rosetta Stone* a bait-prey or prey-prey pair is specifically interacting if the encoding genes belong to a conserved operon or fused with certain probability. The probability metrics are taken from the Prolinks database. An important criterion for the prey-prey pair was a co-purification of the preys with two or more different baits.

C. Discovering protein complexes in the protein affinity network

Altogether, the protein pairs identified by pull-down and genomic-context methods represent a protein affinity network. We consider that a maximal clique in the network corresponds to a subset of interacting proteins in a complex. To reveal the structure of protein complexes, we apply our efficient parallel implementation of the maximal clique enumeration (**MCE**) algorithm by Bron and Kerbosch [1], presented in [15]. Many of the identified cliques overlap. To identify a set of putative complexes, we merge similar cliques based on the meet/min coefficient, defined as the ratio of the number of common proteins in both cliques to the minimum size of the two cliques. Our clique merging iterates by merging the two cliques with the highest coefficient (if the fraction of overlap is above the *merging threshold*, 0.6). We replace both cliques with the combined one. The iteration stops when no change in the clique sets between two consecutive runs is observed. The final cliques are putative protein complexes.

The main alternative for finding strongly related groups within a network are polynomial-time clustering heuristics, such as UVCLUSTER [25], Molecular Complex Detection (MCODE) [23], and Markov Clustering (MCL) [22]. However, clique-based techniques hold several advantages over such techniques. By making use of the clique enumeration of the network, nodes can be assigned to multiple, overlapping clusters. This

property is especially desirable in the context of protein affinity networks, as a single protein may exhibit multiple functions [27], [19]. In addition, clique-based techniques are less sensitive to noise in the networks [24] and identify more biologically-relevant protein complexes (for example, cliques show more than 10% higher functional homogeneity than heuristic clusters in [19]). While the worse-case complexity for clique-based techniques is quite high (a graph of n vertices may have up to $3^{n/3}$ maximal cliques [26]), actual performance on biological networks is fast, due to the sparsity of connections in the networks.

D. Efficient parallel MCE for "perturbed" networks

We describe a parallel algorithm for enumerating maximal cliques in perturbed graphs. A serial version of the algorithm has already been published in [16], [17], but we summarize our previous results here.

The algorithm handles two different types of perturbations, edge removal and edge addition. These perturbations correspond to raising or lowering an edge-weight threshold applied to a protein affinity network.

III. Parallel edge removal algorithm

A. Serial algorithm

In this section, we will briefly recapitulate the relevant theoretical results from our earlier work [16], [17] for updating the set of maximal cliques of a graph when some number of edges are being removed from the graph. We refer to the graph before the perturbation as G, and we use G_{new} to denote the graph after the perturbation. Similarly, we use $\mathcal{C}$ and $\mathcal{C}_{new}$ to denote the set of maximal cliques of G and G_{new}, respectively. The objective of the perturbation algorithm is to enumerate the "difference sets" $\mathcal{C}_+ = \mathcal{C}_{new} \setminus \mathcal{C}$ and $\mathcal{C}_- = \mathcal{C} \setminus \mathcal{C}_{new}$ so that $\mathcal{C}_{new}$ may be determined from $\mathcal{C}$.

If no edges are being added to the graph, then any maximal clique of G that is also a clique in G_{new} must be maximal in G_{new}. From this observation, we can see that the cliques of $\mathcal{C}_-$ will be exactly the cliques of G that do not contain an edge being removed. Conversely, the cliques of $\mathcal{C}_+$ will be the subgraphs of the cliques in $\mathcal{C}_-$ that form maximal cliques in G_{new}.

Theorem 1: If a set of edges E_- is being removed from G, then
$\mathcal{C}_- = \{S \in \mathcal{C} \mid S \text{ contains an edge being removed}\}$ and
$\mathcal{C}_+ = \{T \mid T \text{ is a complete subgraph of some } S \in \mathcal{C}_- \text{ that is maximal in } G_{new}\}$.

To enumerate the cliques of $\mathcal{C}_-$, we simply need to retrieve the set of maximal cliques of G that contain an edge being removed. In order to retrieve this set efficiently, we pre-calculate and index the cliques of $\mathcal{C}$ that contain each edge of G, associating each clique of $\mathcal{C}$ with a clique ID and associating each edge of G with the IDs of cliques that contain the edge. We retrieve the set of clique IDs associated with each edge being removed and combine these sets, eliminating the "duplicate" clique IDs that contain more than one edge being removed.

To enumerate the cliques of $\mathcal{C}_+$, we employ a recursive procedure to divide each formerly maximal clique of G, as these subgraphs are likely to be missing relatively few edges. At each step of the procedure, we choose a single vertex, v, that is incident to at least one edge being removed and form two subgraphs that may be further divided. The first subgraph is formed by removing v from the current subgraph, and the other is formed by removing the vertices not adjacent to v in G_{new}. Effectively, these two subgraphs serve to eliminate all of the "non-edges" that are incident on v at each iteration, eventually resulting in fully connected subgraphs.

To ensure that the subgraphs we generate for $\mathcal{C}_+$ are maximal, we maintain arrays containing the vertices outside of the current subgraph adjacent to some vertex of the subgraph, as well as the number of subgraph vertices to which each of these *counter* vertices are not adjacent in G_{new}. As vertices are removed from the clique, the connectivity of the *counter* vertices is updated, and the algorithm backtracks if the non-adjacency value for a *counter* vertex becomes zero. A clique is maximal if and only if there is no vertex of the graph adjacent to every vertex of the clique—if a *counter* vertex becomes adjacent to all of the vertices of our subgraph, we may stop the recursive division, as no further subgraph will be a maximal clique. We continue the division procedure until we have found all complete subgraphs of cliques in $\mathcal{C}_-$ that are maximal in G_{new}.

B. Parallel work division

We parallelize execution along the retrieval and processing of the cliques of $\mathcal{C}_-$, using clique IDs as the units of work. The main advantage of using clique IDs

is that, clique IDs are lightweight and easily passed between processors. This level of parallelism is somewhat coarse, though. While clique IDs likely correspond to relatively small workloads, a single clique ID could potentially generate a huge search space, resulting in a large chunk of work that could not be split across processors.

For our implementation, we chose to use a producer-consumer model to distribute the work among the processors. In the producer-consumer model, a single processor is responsible for accessing the edge index to retrieve the set of clique IDs for cliques that contain an edge being removed from the graph. After the producer has retrieved these clique IDs and placed them on a queue, it begins distributing these clique IDs to the consumers, or processing clique IDs if all of the consumers already have work.

All processors other than the producer are referred to as consumers. Each consumer iteratively requests a block of work from the producer and then processed the clique IDs it receives. For our implementation, we chose to distribute work in blocks of 32 clique IDs. A consumer continues to request work until the producer responds that it has no work left, at which point the consumer stops.

The primary advantage of the producer-consumer model is its simplicity and similarity to the serial code. Additionally, it requires no explicit load balancing, as the consumers simply request work from the producer until all of the work has been completed. One disadvantage of this approach is that the producer is the only processor that looks up the set of clique IDs containing an edge being removed, serializing this phase of the algorithm. However, in our experiments, the time spent in this phase was quite low (less than 0.01 seconds), so this strategy was quite effective.

C. Pruning duplicate subgraphs

After retrieving the set of cliques containing an edge being removed, the edge removal algorithm recursively subdivides the retrieved cliques in order to find the novel maximal cliques introduced by the edge removal. To improve the efficiency of this division procedure, we wish to avoid applying the recursive procedure multiple times to subgraphs that are contained in more than one clique of $\mathcal{C}$. In order to eliminate these duplicate subgraphs without requiring any communication between processors, we introduce novel theory based on lexicographic ordering. Essentially, we allow a subgraph to be produced only by the clique of $\mathcal{C}_-$ that is lexicographically first among those cliques of $\mathcal{C}_-$ that are supergraphs.

Definition 1: We define some lexicographic ordering among the vertices, i.e., $V(G) = \{v_1, v_2, \ldots, v_n\}$. A subgraph S *lexicographically precedes* subgraph T iff there exists some $v_i \in S \setminus T$ such that $i < j$ for all $v_j \in T \setminus S$. We denote this relationship as $S \overset{\mathcal{L}}{<} T$.

Note that this definition differs from the usual lexicographic ordering in that a supergraph of a graph would precede the graph it contains; however, as the perturbation algorithm does not compare subgraphs that contain one another, this definition is sufficient for our purposes. We now present a technical result that will allow us to recognize whether a subgraph is being produced from its lexicographically first supergraph in $\mathcal{C}_-$.

Theorem 2: Let C be a clique of $\mathcal{C}_-$, and let $S \subset C$ be a maximal subgraph of C produced by the recursive procedure. Let $R = C \setminus S$, and let v_i be the lexicographically first *counter* vertex that is adjacent to every vertex of C in G (but not in G_{new}). C is the lexicographically first clique of $\mathcal{C}_-$ that contains subgraph S if and only if some vertex of $R_i = \{v_k \in R \mid k < v_i\}$ is nonadjacent to v_i in G.

Proof: We first prove the negation of the claim. Suppose that every vertex of R_i is adjacent to v_i. Consider the subgraph $X = S \cup R_i \cup \{v_i\}$. $S \cup R_i$ is a subset of C, so all vertices in R_i and S must be adjacent in G, and v_i is adjacent to all of S and R_i in G by supposition. Thus, X must form a clique in G. Further, X precedes C lexicographically, as every vertex of $C \setminus X$ is lexicographically after v_i by the construction of R_i. Thus, some supergraph of X forms a maximal clique in $\mathcal{C}_-$ that lexicographically precedes C.

Suppose that there is some clique $C' \in \mathcal{C}_-$ such that $C' \supset S$ and $C' \overset{\mathcal{L}}{<} C$. Let C' be the lexicographically first such clique. By Definition 1, this means that there is some $v_k \in C' \setminus C$ such that $k < j$ for all $v_j \in C \setminus C'$. Note that, as C' is a clique in G, v_k is adjacent to every vertex of S in G. Also, k must be the lexicographically first such vertex—if there were some v_j adjacent to all of S in G such that $j < k$, $S \cup \{v_j\}$ would be a clique of G that contains S and lexicographically precedes C' (as would the clique of $\mathcal{C}_-$ that contains $S \cup \{v_j\}$).

Since $C' \overset{\mathcal{L}}{<} C$, no vertex of $C \setminus C'$ may precede v_k. As such, every vertex of R_i must be contained in C' and thus be adjacent to v_i in G. ■

To make use of this result in the algorithm, we maintain R, the set of vertices removed from the original clique of $\mathcal{C}_-$, in a lexicographically sorted array. In addition, for each *counter* vertex, we maintain arrays not only for the number of subgraph vertices that are nonadjacent in G_{new} (as in the serial algorithm), but also for the number of vertices nonadjacent in G. When a *counter* vertex v_i becomes adjacent to all vertices of the subgraph in G, we check whether v_i lexicographically precedes the lexicographically first vertex of R. If so, we iterate through R until we find a vertex lexicographically later than v_i (indicating that the original clique is the lexicographically first) or a vertex nonadjacent to v_i, at which point we would backtrack.

D. Parallelizing index accesses

One of the key ways in which we achieve efficient calculation of the cliques in the perturbed graphs is by using various indices to speed up retrieving the cliques of the original (unperturbed) graph. However, in a typical large-scale parallel system, disk accesses are relatively expensive and unlikely to scale, as all of the users on the system may share a single distributed file system. As such, disk accesses need to be minimized and replaced with memory accesses where possible, since memory is local to each processor and is able to scale. Thus, we adopt a strategy of reading in the entire index when possible, or a large segment of the index when the index is too large to fit into memory.

IV. Parallel edge addition algorithm

A. Serial algorithm

We approach the problem of adding edges to a graph as the inverse of the edge removal problem. Specifically, if edges are being added to G to form G_{new}, we consider the perturbation caused by removing those same edges from G_{new} to form G. From Section III-A, the set of maximal cliques being removed from G_{new} by the reverse perturbation (which is exactly $\mathcal{C}_+$, the set of cliques being added to G by the original perturbation) will be the cliques of $\mathcal{C}_{new}$ that contain an edge being "removed from G_{new}." Further, the set of maximal cliques being added to G_{new} by the reverse perturbation (which is exactly $\mathcal{C}_-$) will be the complete subgraphs of the cliques in the previous set that form maximal cliques in G.

Unlike the removal case, though, we have not indexed the cliques of G_{new} by the edges they contain (as we are trying to *calculate* the cliques of G_-). Thus, to calculate the set of cliques in G_{new} that contain one of the added edges, we employ a variation of the Bron-Kerbosch (BK) clique enumeration [1]. To calculate the cliques containing edge (u, v), we initialize the `compsub` array used by BK to contain u and v as the clique set, the common neighbors of u and v that succeed u and v lexicographically as the *candidate* set, and the common neighbors of u and v that precede u and v lexicographically as the *not* set.

Once we have calculated the set $\mathcal{C}_+$, we apply the recursive removal procedure described earlier to the cliques of $\mathcal{C}_+$ to calculate $\mathcal{C}_-$. However, unlike the edge removal case, we can check the maximality of the resulting subgraphs by looking up the cliques in an index that maps clique hash values to the IDs of maximal cliques of G that correspond to those hash values. As such, we only need to keep track of *counter* vertices adjacent to the clique in G_{new} but not in G, so that we can still eliminate duplicate subgraphs as described in Section III-C.

B. Parallel edge addition algorithm

In order to parallelize the edge addition algorithm, we adapt the parallel BK implementation described in [15] to the problem of finding cliques in G_{new} that contain an edge being added. As before, we can modify the initial workload of the BK algorithm in order to enumerate only those maximal cliques containing the added edges. We distribute the set of added edges (and thus the corresponding initial *candidate list* structures) among the processors in a Round-Robin fashion. The load balancing strategy is performed on two levels—local and remote work sharing. Local work sharing occurs when one thread on a processor runs out of work, at which time it checks the other threads' work stacks (in some randomized order) to see if another thread on the same processor has additional work. Remote work sharing is handled similarly, in that when all threads on a shared-memory processor have exhausted their work stacks, that processor polls the other processors to see if another processor still has work in its work stack. Again, polling is performed in

a random order so as to avoid having a single processor inundated with work requests.

Once a processor or thread has located some work that it can steal, the target thread or process transfers a single *candidate list* structure from the bottom of its work stack to each thread requesting work. The rationale behind this load balancing strategy is that a single *candidate list* structure may represent a wildly variable amount of work, and those *candidate list* structures that were generated earliest (and therefore reside on the bottom of the work stack) are the most likely to represent a large amount of work.

As with the edge removal case, we treat the recursive removal operation on the resulting cliques of $\mathcal{C}_+$ as an indivisible unit of work, as the perturbation being applied to the graph—and hence the typical individual subgraph division operation—is assumed to be small. Duplicate subgraph pruning is performed as described in Section IV-A, and the hash value index is read into memory. For all of the graphs that we tested, this index was able to be read into memory in its entirety; however, for larger graphs, it may be necessary to split the index and read in only a section of the index at a time into memory. In this event, it may be more effective to distribute the index among the processors and pass the potential cliques of $\mathcal{C}_-$ to the processor that possesses the appropriate section of the hash value index.

V. Results

A. Scalability of parallel perturbed MCE

To test the scalability of the edge removal algorithm, we first use a protein-protein interaction network derived by Zhang *et al.* [19]. This network was constructed by applying a threshold of 1.5 to the Purification Enrichment scores [21] calculated using experimental data published by Gavin *et al.* [2]. The network contains 2,436 vertices, which represent various proteins in the yeast *Saccharomyces cerevisiae*, and 15,795 edges, which represent likely interactions between the proteins. The graph contains 19,243 maximal cliques of size three or larger, which represent potential protein complexes. For this experiment, we generated a 20% removal perturbation in which 3,159 edges of the graph were randomly selected to be removed, with an equal probability for each edge to be selected.

Our experiments were performed on the Jaguar system at ORNL; the results appear in Figure 2. Speedup

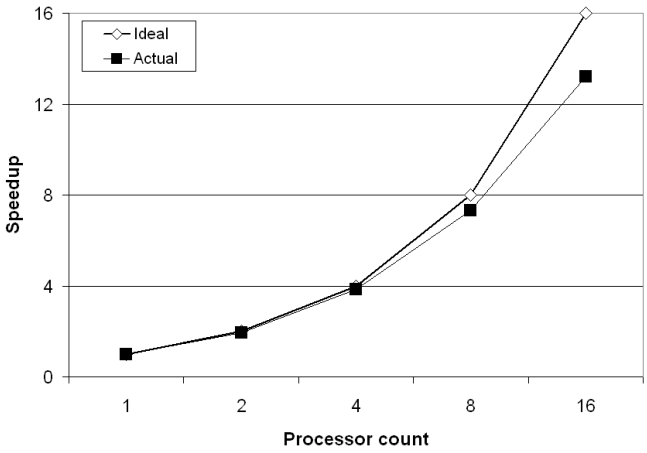

Figure 2: Enumeration time speedup for parallel edge addition.

was calculated for the time spent in the main work phase of the algorithm, in which the algorithm retrieves the vertices of a given clique, performs the recursive removal procedure, and balances the processor workloads, and this speedup is plotted against ideal (linear) speedup. As you can see in the figure, scaling for the edge removal algorithm was quite good, with a speedup of 13.2 at 16 processors.

Using the index access and load balancing strategies outlined in Sections III-D and IV-B, we performed several experiments to test the scalability of the edge addition algorithm. For this experiment, we use a much larger graph derived from the Medline database [20] described in [18]. We chose to use this dataset because of its large size (2.6 million vertices), sparsity (1.9 million total edges), and weighted nature. By applying edge weight thresholds of 0.85 and 0.80 to the Medline data, we are able to generate graphs of 713,000 and 987,000 edges, respectively, resulting in an edge addition perturbation of about 38.5% on the smaller graph. This perturbation adds 73,623 maximal cliques to the 70,926 cliques of 0.85-weight graph and removes 34,745, for a total of 109,804 maximal cliques in the 0.80-weight graph.

The timing results for this perturbation appear in Table I. All times are given in seconds and represent the longest duration that a single processor spent on the given task. The timing results are broken down as follows. *Init* time represents the amount of time taken by the algorithm in allocating memory for the

algorithm data structures and reading the graph and indices into memory. *Root* time represents the amount of time taken by the algorithm in generating the initial set of *candidate list* structures for the modified BK algorithm. *Main* time represents the combined amount of time required to run the BK algorithm to detect the set of cliques in $\mathcal{C}_+$ as well as performing the recursive removal procedure, index lookups required to calculate $\mathcal{C}_-$, and load balancing. *Idle* time represents the time spent by a processor that has finished its workload and has no other work to steal.

Table I: Scalability for edge weight–induced perturbation on the Medline graph.

Procs	Init	Root	Main	Idle
1	0.876	0.000	1.459	0.000
2	0.951	0.000	0.773	0.005
4	1.197	0.000	0.489	0.002
8	1.381	0.000	0.249	0.007

In Table I, the *Init* phase plays a significant role in the overall runtime of the algorithm and does not scale with the addition of more processors. However, the *Main* phase of the algorithm scales reasonably well, for a speedup of 5.86 at 8 processors, and the time spent by idling processors is minimal. Also, it is clear that the time spent in the *Main* phase of the algorithm is too short to permit a full scalability study. As such, we evaluate the scalability of the edge addition algorithm to larger processor counts by using the Medline data to generate larger workloads. In order to increase the problem size evenly, we formed successively larger graphs made up of independent components identical to original graph, linearly increasing the number of vertices, edges, perturbation size, maximal cliques, and resultant index data. We term these independent components "copies," and we increased the number of copies in our graph from 1 to 6 as we increased the number of processors from 1 to 64. We only present results as to the scalability of the *Main* time—the *Root* and *Idle* phases last less than 0.07 seconds in all cases, and the *Init* time, which consists of reading in the graph and index data from the disk, does not scale and eventually dominates the algorithm runtime. The results for our experiments appear in Figure 3.

Interestingly, despite the relatively large perturbation size, the edge addition algorithm is far faster than enumerating the maximal cliques of the perturbed graph using BK, particularly, as the size of the graph

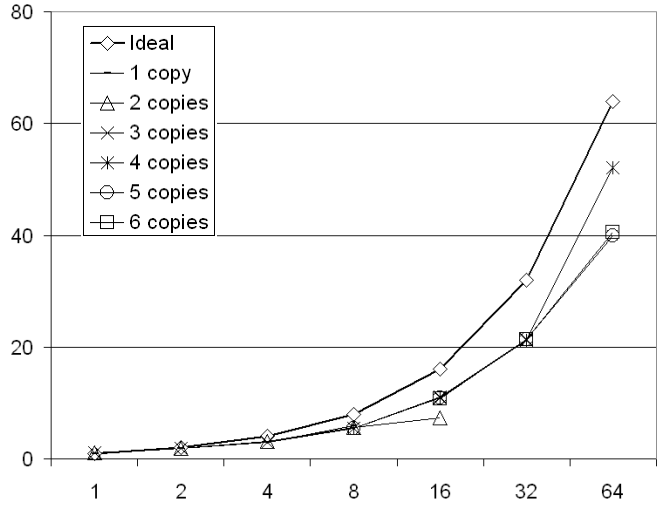

Figure 3: Normalized speedup versus number of processors.

(number of copies) is increased. We do not present full results here, but as a point of reference, enumerating the maximal cliques of the four-copy Medline graph took over 20 minutes using 128 processors on Jaguar, with more than 99% of that time being spent in the initial workload generation (*Root*) phase (compared to around 8 seconds on 4 processors for the edge addition algorithm).

From Figure 3, we see that the *Main* time for the perturbed edge algorithm scaled almost perfectly as the size of the graph (number of copies) increased. Speedups were calculated as $(t_1 * n_c)/t_{c,p}$, where t_1 is the *Main* time for calculating the perturbation on one copy of the graph using a single processor, n_c is the number of copies of the graph, and $t_{c,p}$ is the *Main* time for calculating the perturbation on c copies using p processors. *Main* time represents the combined amount of time required to run the BK algorithm to detect the set of cliques in $\mathcal{C}_+$ as well as performing the recursive removal procedure, index lookups required to calculate $\mathcal{C}_-$, and load balancing. The scalability of the *Main* time is within two-thirds of ideal, though this may be due to the short times involved—the *Main* time for 6 copies of the graph was only 0.216 seconds. As the memory requirements for the algorithm increase with the number of copies, we were unable to test our algorithm for graphs larger than 6 copies, or 15.6 million vertices and 11.4 million edges.

B. Effects of duplicate subgraph pruning

To illustrate the effectiveness of our theoretical results on duplicate subgraph pruning described in Section III-C, we evaluate the edge removal code with and without the code that detects duplicate subgraphs enabled. This experiment is performed on the same 20% perturbation of the Gavin *et al.* protein-protein interaction data [2] and is performed on the ORNL Jaguar system. For this experiment, we used a single processor, as well as in-memory index accessing strategy. The results of this experiment, in Table II, show that duplicate cliques can potentially represent a majority of the algorithm output if not properly pruned. Here, we report the number of (potentially duplicate) cliques of $\mathcal{C}_+$ identified by the algorithm as well as timing results using the in-memory index access and producer-consumer strategies on a single processor. As the duplicate pruning results are only relevant during the recursive removal phase of the algorithm, we include here only the time spent in the *Main* phase of the algorithm, which encompasses clique retrieval, recursive removal, and load balancing. The amount of time spent during the other phases of the algorithm was consistent between the two runs. Unsurprisingly, algorithm runtime was improved dramatically by the application of this theory. Moreover, the algorithm results without duplicate pruning would require further processing to produce an enumeration of maximal cliques without duplicates.

Table II: Effects of duplicate pruning on algorithm performance.

Duplicate pruning?	$\lvert\mathcal{C}_+\rvert$	Main enumeration time
Without	228373	25.681
With	33941	6.830

C. Genome-scale reconstruction of R. palustris protein complexes

We applied our framework to identify protein complexes from pull-down experiments with 186 unique proteins as baits and 1,184 unique proteins as preys in *Rhodopseudomonas palustris* bacterium. The Validation Table, including 205 genes clustered into 64 known complexes, was manually created based on the annotation of genes in the *R. palustris* genome from GenBank (2006). After tuning multiple "knobs," for the pull-down step, we ended up using the *p-score* and Jaccard's score with the threshold of 0.3 and 0.67, respectively. For the genomic-context step, we used the predicted transcription units from BioCyc, the predicted gene fusion events and conserved operons from Prolinks. The thresholds for probability were 3.5E-14 and 0.2 for *Gene neighborhood* and *Rosetta Stone*, respectively. Both steps identified 1020 specific protein-protein interactions, with only 6% from the pull-down step. This set produced 59 isolated modules, 33 complexes, and 3 networks. A module is defined as an isolated set of interacting proteins. A complex is a subset of at least three interacting proteins in the module; all proteins in the subset are supposed to physically interact with each other. A module is a network if it includes more than one complex.

Many identified complexes represent important cellular functions and provide insight into new metabolic capabilities of this versatile bacterium. Most complexes belong to the three large protein networks. The first network is comprised of one large complex of interacting subunits of ABC transporters and 5 smaller complexes including multi-subunit enzymes tryptophan synthase and acyl-CoA dehydrogenase. Acyl-CoA dehydrogenase complex in the network is represented by an electron transfer flavoprotein alpha-subunit (etfA) and by two proteins (RPA1612, RPA4798) annotated as putative acyl-CoA dehydrogenase. Acyl-CoA dehydrogenase in *R. palustris* has been shown to be transcribed from a gene in the *pimFABCDE* operon, along with other enzymes involved in the beta-oxidation of dicarboxylic acids [3]. An interesting observation can be made about the interaction of the acyl-CoA dehydrogenase complex with the complex that we annotated as representing the electron transport to nitrogenase. The latter is comprised of an electron transfer flavoprotein etfA, 3 electron transfer flavoprotein *fixA*, *fixB*, *fixC*, and possible electron transfer flavoprotein dehydrogenases transcribed from RPA1037. In some diazotrophic bacteria a set of proteins transcribed form the *fixABCX* operon forms a membrane protein complex that plays a central role in electron transfer to nitrogenase [4]. The identified complex provides a similar metabolic route to reduction of nitrogenase in *R. palustris*. Thus, acyl-CoA dehydrogenase complex and the complex of electron transfer to nitrogenase may link beta oxidation of dicarboxylic acids to nitrogenase activity in *R. palustris* metabolism.

The second network has 10 complexes. Two relate to fatty acid biosynthesis. Complex 1 relates to the cobalamin synthesis complex (CobB, CobD, CobO, CobQ2) through the lipoic acid synthetase (*lipA2*). The other annotated complexes in the second network include the Calvin cycle related complex (CbbA CbbF, CbbP, CbbM, CbbT1), succinyl-CoA synthetase complex (SucA, SucB, SucC, SucD, SdhA, DldH), and a chaperone complex (DnaK, DnaJ, recN and possible grpE). A centerpiece of the third network is a ribosomal complex. Four other complexes include RNA polymerase complex, two complexes of ATP synthase, ATP sulfurylase complex and cell division related complex. The individual complexes, which are not part of a network, mainly include enzymes comprising several subunits, such as NADH-ubiquinone dehydrogenase, nitrogenase, carbon-monoxide dehydrogenase, bacteriochlorophyllide reductase, and chaperonin.

VI. Conclusion

In spite of low specificity of bait-prey and prey-prey interactions in pull-down data, the proposed framework captured the composition of known complexes and predicted the composition for several unknown ones. Most predicted complexes are biologically meaningful and represent a comprehensible set of cellular functions in *R. palustris*. The complexes provide insight into function of uncharacterized proteins, metabolic capabilities, and even specific mechanisms underlying these capabilities. The framework requires the enumeration of maximal cliques in protein affinity networks. The proposed parallel, scalable algorithm enables the efficient enumeration of maximal cliques in response to changes in the genome-scale network. These computational advancements allow for reconstruction of genome-scale protein interaction networks and the efficient tuning of parameters while finding the optimal networks.

Acknowledgment

The authors would like to thank the authors of the parallel maximal clique enumeration code and the developers of the biological databases that have been utilized in this study. This work was supported in part by grants DOE DE-SC0005340, DOE DE-FG02-08ER25848, NSF IIS-0905205, NSF OCI-0724599, and NSF CCF-1029166. This research has also been supported by the "Ultrascale Computational Modeling of Phenotype-Specific Metabolic Processes in Microbial Communities" project from the U.S. Department of Energy, Office of Science. Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725.

References

[1] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, pp. 575-577, 1973.

[2] Gavin, A.C., *et. al.*, "Proteome survey reveals modularity of the yeast cell machinery," *Nature*, vol. 440, pp. 631-636, 2006.

[3] F. H. Harrison and C. S. Harwood, "The pimFABCDE operon from Rhodopseudomonas palustris mediates dicarboxylic acid degradation and participates in anaerobic benzoate degradation," *Microbiology*, vol. 151, pp. 727-736, 2005.

[4] T. Edgren and S. Nordlund, "The fixABCX genes in Rhodospirillum rubrum encode a putative membrane complex participating in electron transfer to nitrogenase," *J Bact.*, vol. 186, pp. 2052-2060, 2004.

[5] T. W. Morris, K. E. Reed, and J. E. Cronan, "Lipoic acid metabolism in Escherichia coli: the lplA and lipB genes define redundant pathways for ligation of lipoyl groups to apoprotein," *J Bact.*, vol. 177, pp. 1-10, 1995.

[6] D. W. Bollivar, T. Elliott, and S. I. Beale, "Anaerobic protoporphyrin biosynthesis does not require incorporation of methyl groups from methionine," *J Bact.*, vol. 177, pp. 5778-83, 1995.

[7] C. von Mering, R. Krause, B. Snel, M. Cornell, *et al.*, "Comparative assessment of largescale data sets of protein-protein interactions," *Nature*, vol. 417, pp. 399-403, 2002.

[8] B. A. Shoemaker and A.R. Panchenko, "Deciphering protein-protein interactions," *PLoS Comput Biol*, vol. 3, p. e43, 2007.

[9] K. Venkateswaran, *et al.*, "Polyphasic taxonomy of the genus Shewanella and description of Shewanella oneidensis sp.," *International Journal of Systematic and Evolutionary Microbiology*, vol. 49, pp. 705-724, 1999.

[10] K. M. Downard, "Ions of the interactome: the role of MS in the study of protein interactions in proteomics and structural biology," *Proteomics*, vol. 6, pp. 5374-5384, 2006.

[11] A. Dziembowski and B. Séraphin, "Recent developments in the analysis of protein complexes," *FEBS Lett*, vol. 556, pp. 1-6, 2004.

[12] O. Puig, *et al.*, "The tandem affinity purification (TAP) method: a general procedure of protein complex purification," *Methods*, vol. 24, pp. 218-229, 2001.

[13] J. Goll and P. Uetz, "The elusive yeast interactome," *Genome Biol.*, vol. 7, p. 223, 2006.

[14] D. S. Goldberg and F. P. Roth, "Assessing experimentally derived interactions in a small world," *Proc Natl Acad Sci U S A*, vol. 100, pp. 4372-4376, 2003.

[15] M. C. Schmidt, *et al.*, "A scalable, parallel algorithm for maximal clique enumeration," *J. Parallel Distrib. Comput.*, vol. 69, no. 4, pp. 417-428, 2009.

[16] W. Hendrix, M. C. Schmidt, P. Breimyer, and N. F. Samatova, "On perturbation theory and an algorithm for maximal clique enumeration in uncertain and noisy graphs," In *U '09: Proceedings of the 1st ACM SIGKDD Workshop on Knowledge Discovery from Uncertain Data*, pp. 48-56, 2009.

[17] W. Hendrix, M. C. Schmidt, P. Breimyer, and N. F. Samatova, "Theoretical underpinnings for maximal clique enumeration on perturbed graphs," *Theor. Comput. Sci.* 411(26-28), pp. 2520-2536, 2010.

[18] A. Awekar, N. F. Samatova, and P. Breimyer, "Incremental All Pairs Similarity Search for Varying Similarity Thresholds," In *SNA-KDD '09: Proceedings of the 3rd Workshop on Social Network Mining and Analysis*, pp. 1-10, 2009.

[19] B. Zhang, *et al.*, "From pull-down data to protein interaction networks and complexes with biological relevance," *Bioinformatics*, vol. 27, no. 7, pp. 979-986, 2008.

[20] Medline Fact Sheet, June 2010. http://www.nlm.nih.gov/pubs/factsheets/medline.html.

[21] Purification Enrichment Scores. http://interactome-cmp.ucsf.edu/.

[22] A. J. Enright, S. Van Dongen, and C.A. Ouzounis, "An efficient algorithm for large-scale detection of protein families," *Nucleic Acids Research*, vol. 30, no. 7, pp. 1575-1584, 2002.

[23] G. Bader and C. Hogue, "An automated method for finding molecular complexes in large protein interaction networks," *BMC Bioinformatics*, vol. 4, no. 1, p. 2, 2003.

[24] D. L. Tabb, M. R. Thompson, G. Khalsa-Moyers, N. C. VerBerkmoes, and W. H. McDonald, "Ms2grouper: group assessment and synthetic replacement of duplicate proteomic tandem mass spectra," *Journal of the American Society for Mass Spectrometry*, vol. 16, no. 8, pp. 1250-61, 2005.

[25] V. Arnau, S. Mars, and I. Marín, "Iterative cluster analysis of protein interaction data," *Bioinformatics*, vol. 21, no. 3, pp. 364-378, 2005.

[26] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23-28, 1965.

[27] T. Aittokallio and B. Schwikowski, "Graph-based methods for analysing networks in cell biology," *Briefings in Bioinformatics*, vol. 7, no. 3, pp. 243-255, 2006.