Scalable Heuristics for Clustering Biological Graphs

Inna Rytsareva*, Ananth Kalyanaraman*, Kishori Konwar†, and Steven J. Hallam†
*School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA
Email: inna.rytsareva@wsu.edu, ananth@eecs.wsu.edu
†Department of Microbiology and Immunology
University of British Columbia, Life Sciences Institute, Vancouver, BC, Canada
Email: kishori@mail.ubc.ca, shallam@interchange.ubc.ca

***Abstract*—Clustering is one of the advanced analytical functions that has an immense potential to transform the knowledgespace when applied particularly to a data-rich domain such as computational biology. The nature of clustering that is of concern in this paper is graph-theoretic — more specifically, given an input graph $G(V, E)$ where vertices represent elements and edges connect elements that are "related" by a certain biological property, the problem is one of identifying tightly-knit (potentially variable-sized) groups where each member of a group is highly related to most, if not all, of the other members of the same group. The computational hardness of the underlying theoretical problem necessitate use of heuristics in practice. In our previous work, we had evaluated the application of a randomized sampling-based heuristic called *shingling* on unweighted biological graphs. In this paper, we present a new variant of this heuristic that not only extends its application to weighted graph inputs but is better positioned to achieve qualitative gains on unweighted inputs as well. We also present parallel algorithms for this heuristic using the MapReduce paradigm. Experimental results on subsets of a medium-scale real world biological input (containing up to 10.3M vertices and 640M edges), constructed out of protein sequence data collected from metagenomic communities, demonstrate significant qualitative improvements in the reported clustering, both with and without using edge weights. Furthermore, performance studies indicate near-linear scaling on up to 4K cores of a distributed memory supercomputer.**

***Keywords*—Graph clustering; MapReduce algorithm; Shingling heuristic; metagenomics; protein family identification.**

I. Introduction

Graph representations are a popular way to model several problems in computational biology. Vertices can be used to represent experimentally acquired data and edges (weighted or unweighted) can be used to represent pairwise relationships between the data. Once modeled as a graph, several scientifically interesting questions can be posed — e.g., performing an Euler tour or Hamiltonian path for genome assembly, finding connected components to group expressed sequences (transcriptomics), and clustering, which forms the focal point for this study. Loosely defined, the clustering formulation targeted in this study is as follows: Given an input graph $G(V, E)$ with n vertices and m edges, "*clustering*" is the act of grouping vertices into tight-knit clusters, where the members of each cluster are closely related to most (if not all) other members of the same cluster, and sparsely related (if at all) to the members of other clusters.

Related work and motivation: There is a rich body of clustering related literature in the context of biological applications, of which a substantial segment is devoted to applications such as gene expression analysis and genome sequencing, where typically single-linkage clustering formulations are used. A different class of applications benefit from a clustering formulation based on detecting denser communities within biological data (which is the focus of this paper) [3], [7], [11], [15], [17], [22], [24]. Brohee *et al.* [6] evaluate clustering techniques for analyzing protein-protein interaction networks. Andreopoulos et al. [2] present a broader, and more conceptual survey of different clustering applications that benefit from tighter cluster techniques.

Despite its potential for discovery, clustering at large-scale remains a daunting computational task not only due to the sheer volumes of data available but also due to the inherent complexity of the analytics involved. The computational hardness of the underlying theoretical formulations [1], [8], [12], [14] implies that heuristics need to be deployed in practice (e.g., [4], [7], [9], [13], [14], [17]). Even with such heuristics, however, parallelism has become necessary in order to tackle modern day inputs which could contain tens of millions to even billions of vertices with even more edges. However, the built-in irregularity in graph structures along with the inherent sequentiality of certain heuristics often makes it a challenging task to design efficient parallel methods.

Recently [20], we presented the design and evaluation of a MapReduce-based parallel implementation of a well known serial graph clustering heuristic called the *Shingling heuristic*, which was originally developed by Gibson *et al* [9] to analyze unweighted internet graphs. The algorithm when implemented and tested on a Hadoop platform demonstrated linear scaling on up to 64 cores on an input size containing about 11M edges of a protein family graph. Despite the linear scaling behavior, the time to solution was rather slow — for example, even on the 11M edges input graph, which is still an order or two magnitudes smaller when compared to other real world graphs available from the same domain, the analysis took $\sim$40 minutes on 128 cores. An investigation revealed that a dominant fraction of the time was expectedly spent on the intermediate shuffle/sort stage of the Hadoop MapReduce library.

In a subsequent study [21], we also conducted a qualitative assessment of the heuristic when applied to biological graphs. While this study showed the effectiveness of the heuristic to detect dense clusters, the analysis also revealed a tendency of the heuristic to leave out a significant fraction of vertices as singletons (i.e., vertices not recruited as part of any cluster). In

978-1-4673-1321-6/12/$31.00

other words, the heuristic was more effective in capturing core elements of a cluster in an attempt to improve intra-cluster edge density albeit at the expense of peripheral elements. This is expected as the original heuristic was designed for the problem of link spam detection in internet graphs, which tend to be represented in the densest parts of the networks. However, for most biological graph inputs, it is desirable to allow some fringe elements to become part of larger, dense clusters. This is because clustering is often used as a way to functionally annotate new members of a family (e.g., proteins), or to consolidate large volumes of sequence data into fewer, non-redundant subgroups for facilitating further downstream processing and discovery. The importance of recruiting members into clusters (without significantly diluting the density of clusters) becomes even more pronounced while analyzing data from environmental microbial communities (i.e., metagenomics data), where it is typically challenging to annotate a significant fraction of the input data owing to the scattered nature in the sequencing (sampling) procedures [10].

Furthermore, for many biological graphs, there is information available on edge weights that need to be incorporated during clustering. For instance, in protein family identification, the degree of sequence similarity between two protein sequences could be used as edge weights (between the corresponding two vertices) in determining the composition of a family.

A. Contributions

In this paper, we make the following contributions:

1) We propose a *new variant* of the standard Shingling heuristic for clustering unweighted real world biological graphs. This modified heuristic uses a normalization technique during its randomized sampling process, which positions it better to overcome the qualitative challenges encountered with the standard version of the heuristic as outlined above.
2) We also show how this new variant of the heuristic extends to handling biological graphs with *edge weights* — something that the original heuristic was not capable of.
3) We present a *MapReduce algorithm* for parallelizing the new variant of the heuristic on distributed memory clusters. We implemented this algorithm using the MapReduce-MPI library [18] so that we could use any cluster with MPI installed on it.
4) We present extensive *experimental results* on a real world metagenomics graph containing 10.3M vertices and 640M edges and its subsets. These results demonstrate a) significant qualitative improvements over the standard heuristic, both with and without using edge weights; and b) significant performance improvements over the previous Hadoop MapReduce implementation, resulting in the analysis of the entire test graph (640M edges) in less than a minute on 4,096 core of a distributed memory cluster.

Organization: The rest of the paper is organized as follows. In Section II, we describe the new variant of the Shingling heuristic and its parallel MapReduce algorithm. In Section III, we present both qualitative and performance results of our experimental evaluation of the proposed method. Section IV concludes the paper with an identification of future work components.

II. Methods

A. Overview of the standard Shingling heuristic

In this section, we provide an algorithmic overview of the standard Shingling heuristic [9]. For complete details we refer the readers to the original paper.

The Shingling heuristic is a randomized approach to identify dense subgraphs in unweighted input graph $G(V, E)$. The main idea of the heuristic is as follows. For any vertex u let $\Gamma(u)$ denote the set of its neighboring vertices. A brute-force approach to detect vertices that are likely to be part of the same dense subgraph would be to compute $\frac{|\Gamma(u_i) \cap \Gamma(u_j)|}{|\Gamma(u_i) \cup \Gamma(u_j)|}$ for every pair of vertices $u_i, u_j \in V$. The Shingling heuristic takes a randomized sampling approach to reduce this search space. More specifically, it obtains random samples of size s (called "shingles") from $\Gamma(u)$ for every "source vertex" $u \in V$, and compares them against one another. If two vertices are part of the same dense subgraph, they are likely to share most of their links in common and hence with a high probability are also expected to share a shingle [5]. As this cannot be guaranteed, the algorithm performs c random trials to improve probability of detecting common shingles.

The overall clustering algorithm proposed by Gibson *et al.* iteratively computes shingles, and in the process transforms the input graph into newer graphs with shingles replacing the original vertex set. The output clusters are computed by detecting connected components from the transformed graph in the final iteration. Algorithm 1 details the steps that occur within one iteration of the algorithm. It has been shown that two iterations are typically sufficient in practice to capture dense subgraphs within real world graphs.

Algorithm 1 Shingling (Input: $G(V, E), s, c$)

```
Generate c random number pairs { < A_j, B_j > }
P ← a big prime number
for u ∈ V do
    Let Γ(u) ← {v | v is a neighbor of u}
    for j = 1 → c do
        Generate Γ^j(u) ← {v^j | v^j = (A_j × v + B_j) % P, v ∈ Γ(u)}
        Let shingle s^j(u) ← Minimum s elements of Γ^j(u)
        Store tuple of the form < s^j(u), u >
    end for
end for
for j = 1 → c do
    Sort all tuples < s^j(u), u > by their shingle ids
    Output transformed graph G_I(V', E'), where V' is the
    set of all shingles and E' = {(s, u) | u generated s}
end for
```

B. The design of a new variant for the Shingling heuristic

Although the standard heuristic has been shown to be effective at capturing a majority of the dense subgraphs [9], its application to biological graphs such as protein sequence homology graphs reveals that a significant fraction (up to a third) of the vertices do *not* get recruited into any of the

clusters for some of the real world data sets we tested (please see experimental results section). Such vertices, referred to as "singletons", typically tend to be low degree vertices. The potential cause for these singletons is that either these vertices have degree less than parameter s and as a result no shingles get generated, or these vertices with low degree have larger degree neighbors which in turn fail to generate the same shingle as their low degree neighbors during the random sampling method.

Figure 1 illustrates a couple of contrasting instances on how low degree vertices should be handled during clustering. In Figure 1a, vertices $u_1 \ldots u_4$ are low degree vertices who should be assigned the same clusters as their larger degree neighbors $u'_1 \ldots u'_4$. The standard heuristic, however, will mark these peripheral elements of the cluster as singletons (for $s > 2$). On the other hand, not all low degree vertices should follow their larger degree neighbors in their cluster assignment. For example, in Figure 1b, the vertex v_1 is a weak bridge between the two dense clusters containing its neighbors v_2 and v_3 respectively, and therefore should be left out as a singleton, keeping the two clusters separate.

Algorithm 2 Normalized Shingling (Input: $G(V, E), s, c$)

```
Generate c random number pairs { < A_j, B_j > };
P ← a big prime number
C ← normalizing constant (default to 100)
for u ∈ V do
  Let Γ(u) ← {v|v is a neighbor of u}
  for v ∈ Γ(u) do
    /* Normalize edge weight for every edge*/
    w_{u,v} ← w_{u,v} × C / Σ_{∀v'∈Γ(u)} w_{u,v'};
  end for
  for j = 1 → c do
    Generate multi-set Γ^j(u)←{w_{u,v} copies of v^j|v^j=(A_j×
    v + B_j)%P, v ∈ Γ(u)}
    Let shingle s^j(u) ←Minimum s elements of Γ^j(u)
    Store tuple of the form < s^j(u), u >
  end for
end for
for j = 1 → c do
  Sort all tuples < s^j(u), u > by their shingle ids
  Output transformed graph G_I(V', E'), where V' is the
  set of all shingles and E' = {(s, u)|u generated s}
end for
```

In Algorithm 2, we present a new algorithmic variant of the standard shingling heursitic. This new algorithmic variant is not only better suited to overcome the challenges outlined above, but also automatically extends the scope of clustering to graphs with *edge weights*. If the graph is unweighted, then we trivially assign a weight of 1 to every edge. In what follows, we explain the algorithm assuming weighted inputs. The main idea of the new algorithm is as follows. Let u be a source vertex. Given that a shingle is a random sample of size s from u's neighbors, the probability that a neighbor v becomes part of a shingle of u should be dictated by the weight of the edge connecting u to v ($w_{u,v}$). This led us to develop a simple normalization method, in which the edge weights of all edges connected to u are recomputed to represent their relative importance to that vertex and the normalized weights are subsequently used for shingle generation. As the

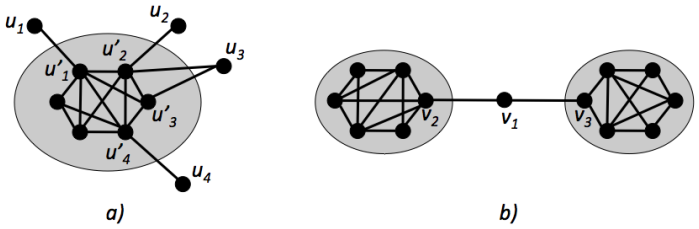

Fig. 1. Cases illustrating the decisions to be made on the clustering of low degree vertices.

original algorithm is oblivious to edge weights during shingle generation, we modified the shingle generation process so that it now conceptually treats an edge with an integer weight x as implicitly containing x copies of unit weight edges (i.e., multi-edge), prior to applying the standard shingle generation method from the original algorithm. To guarantee integer edge weights, we multiply the normalized edge weight by a constant C and ignore the decimal places. In other words, for $C = 100$, this implies that the edge weights are normalized to the scale of 100. Ideally, one would like to set the value of C to the value of the maximum degree (or weighted degree) of a vertex, as otherwise there is a possibility that the normalized weight value evaluates to less than 1 (and therefore interpreted as 0). For instance, in an unweighted graph, if a vertex has a degree of more than C, then each edge's normalized weight will evaluate to zero, which implies that these edges will never participate in the shingling process. Forcibly resetting the normalized weight to 1 in such cases intuitively provides those edges a chance to participate in the shingling process. (In all our experimental studies (please see Section III), we used this trick combined with a value of $C = 100$ and this combination yielded good results.)

The above idea of normalization also helps in the recruitment of singletons into clusters. For instance, a low degree vertex which has less than s neighbors, would now be able to generate shingles due to normalized weights and subsequently participate in clustering. A caveat is that the new "shingles" that we generate using this new variant could potentially have a size smaller than s (as duplicate occurrences of a vertex within a shingle should be removed). For instance, in Figure 1a all shingles generated from vertex u_1 will be the set $\{u'_1\}$. Yet, from our experiments, we observe that this is not a problem and that the large number of random trials ($c \approx 100$) is still sufficient to cluster these low degree vertices into their appropriate neighboring clusters.

We also note that the normalization approach introduces a possibility, although with very low probability, that two disparate cluster structures get combined into a larger cluster. For the example shown in Figure 1b, let us assume unit weight edges connected originally to v_1. After normalization to the scale of 100, these edges will be assigned weights of 50 each. This leads to the possibility of v_1 enumerating any of these three shingles: $\{v_2, v_3\}$, $\{v_2\}$ or $\{v_3\}$ during different random trials. Given the same possibility persists for the other neighbors of v_2 and v_3 within each of their individual clusters, it becomes theoretically possible that the clusters are combined based on common shingles. However, the probability of such merging events are expected to be extremely low

owing to the larger degrees of the vertices in the individual clusters, and the fact that these shingles need to be produced during the same random trial in order to lead to merges. Although we acknowledge this to be a theoretical possibility for false merging by the normalization-based method, we did not observe such events in our empirical evaluation.

C. Parallel MapReduce algorithm

The new variant of the Shingling heuristic using normalization also maps directly to the MapReduce model. Each Map process takes as input the adjacency list for a given source vertex u, and performs the following tasks:

Step1) Normalize all the edge weights for u.
Step2) Generate c shingles for u, and emit the shingles in the form of tuples $< s, u >$, where s is a shingle generated by u.

The shingle structure contains information on the shingle id and the set of at most s vertices constituting that shingle. The shuffle stage sorts the tuples by their shingle ids.

Each Reduce process takes the list of all source vertices that generated a given shingle id and emits tuples of the form $< s, u >$ where s is the shingle and u is every distinct source vertex in the list. The union of all the tuples emitted by all the reducers constitute the transformed graph $G_I(V', E')$, which is passed as input to the next Shingling phase.

Implementation: All our implementations for the standard and normalized algorithms were in C/C++ using MapReduce-MPI library [18] that is written with standard MPI calls and compiled into a regular MPI program.

III. Experimental Results

Experimental setup: The experiments were performed on the *Hopper* supercomputer at the National Energy Research Scientific Computing Center (NERSC). Hopper is a 1.28 petaflop/sec Cray XE6 consisting of 6,392 compute nodes made up of 2 twelve-core AMD 'MagnyCours' 2.1GHz prosessors and 32 GB RAM per node. The MPI library is a custom version of mpich2 for Cray XE systems, version 5.5.5. For MapReduce on this MPI platform, we used the MapReduce-MPI library [18], [19].

As our test input, we used a metagenomics sequence homology graph (see Table I). This graph contains open reading frame/amino acid sequences obtained from a variety of environmental microbial community sources (referred as the "UBC data set"). Homology detection was performed on this set using *pGraph* [23] and the resulting graph contains a total of 10.3M vertices (sequences) and 640M edges (homologous pairs). For edge weight calculations, we used the degree of sequence similarity computed through Smith-Waterman alignments. The clustering of such a protein sequence homology graph can lead to metagenomic protein family identification [24], [22].

TABLE I. Input statistics for metagenomics protein sequence data.

Input label	*Number of edges (m)*	*Number of vertices (n)*
UBC-25M	25×10^6	$3,965 \times 10^3$
UBC-50M	50×10^6	$4,525 \times 10^3$
UBC-100M	100×10^6	$6,795 \times 10^3$
UBC-200M	200×10^6	$7,336 \times 10^3$
UBC-400M	400×10^6	$8,958 \times 10^3$
UBC-640M	640×10^6	$10,346 \times 10^3$

A. Qualitative evaluation

There is no single metric that is perfectly suited to measure all aspects of clustering quality. Therefore, we used a combination metrics — such as cluster modularity, the average density of clusters (which is agnostic to edge weights), and other clustering statistics such as the number of clusters, number of singletons and number of vertices in the largest clusters — as indicators of quality. For modularity calculation, we used Newman's formula [16] as shown in Equation 1, taking edge weights into account.

$$Q = 1/2m \sum (A_{ij} - k_i k_j / 2m) \delta(c_i, c_j), \tag{1}$$

where A_{ij} represents the weight of the corresponding edge (or 0 if there is no edge), k_i is the total weight of edges adjacent to vertex i, k_j is the total weight of edges adjacent to vertex j and m is the total weight of all edges. The density of a cluster is defined as the ratio between the number of its intra-cluster edges and the theoretical maximum on the number of such edges. Note that the upperbound for the density and modularity is 1.

For qualitative evaluation, we compared the quality of the clusters (by the above metrics) generated by three methods: A) the *normalized* implementation of the Shingling heuristic proposed in this paper, B) our previous implementation of the *standard* Shingling heuristic, and C) the *Louvain* method [4], which is one of the most widely used (sequential) methods for detecting communities based on the modularity-maximization procedure. We also compared the quality difference between the unweighted and weighted inputs of the same graph. All analysis was performed on two subgraphs from the UBC data set — containing 25M and 100M edges, and using shingling parameters $s = 2$ and $c = 100$.

Tables II shows the results for analyzing the 25M and 100M data sets. A comparison of the results from our standard vs. normalized implementations of the Shingling heuristic shows the effectiveness of our normalized heuristic in detecting significantly denser clusters than the standard heuristic. It can also be observed that the normalized heuristic significant improves modularity over the standard heuristic for the 25M data set (0.84 to 0.99). Given that the Shingling heuristic is a density-driven approach, it is impressive to note that all the modularity figures reported are directly comparable to that of the Louvain method, which is solely a modularity-driven approach.

The results also show that normalization is effective in reducing the number of singletons drastically relative to the standard heuristic (35% to $\sim$2% for 25M, $\sim$28% to $<$1% for 100M) without compromising on the clustering quality. Reducing singletons is important from the point of view of keeping the loss of sequence information as minimal as possible. However, the main challenge is to recruit more sequences into clusters without adversely affecting the overall quality of clustering, especially cluster density. To this effect, it

TABLE II. QUALITATIVE RESULTS OF THE CLUSTERING OBTAINED FROM OUR DIFFERENT IMPLEMENTATIONS AND THE LOUVAIN METHOD FOR TWO SUBSETS OF THE UBC DATA SET.

Input	*Data set, Algorithm*	Quality metrics		Clustering statistics		
		Modularity	*Density*	*# non-singleton clusters*	*% of singletons*	*Largest cluster size*
UBC-25M	Unweighted, Standard	0.8454	0.3126±0.2506	27,479	35.15%	21,973
	Unweighted, Normalized	0.9928	**0.5603±0.3866**	95,505	2.39%	25,827
	Weighted, Normalized	**0.9937**	**0.5603±0.3866**	95,505	2.12%	25,864
	Weighted, Louvain	0.9695	0.5309±0.3864	116,558	0%	13,297
UBC-100M	Unweighted, Standard	0.9299	0.6715±0.2526	145,627	27.96%	36,586
	Unweighted, Normalized	0.8839	**0.8658±0.2466**	601,707	0.65%	38,938
	Weighted, Normalized	0.8988	**0.8658±0.2466**	601,716	0.54%	38,951
	Weighted, Louvain	**0.9628**	0.8634±0.2505	701,527	0%	13,531

TABLE III. RUN-TIME (IN SECONDS) FOR OUR IMPLEMENTATION AS A FUNCTION OF THE INPUT AND SYSTEM SIZES ON THE HOPPER SUPERCOMPUTER FOR UNWEIGHTED GRAPHS.

Input graph	*Time using p cores*						
	p =64	*p =128*	*p =256*	*p =512*	*p =1024*	*p =2048*	*p =4096*
UBC-25M	104.4	59.81	37.21	26.66	17.5	14.8	
UBC-50M	159.87	100.43	50.89	32.66	25.96	17.84	20.69
UBC-100M		158.22	90.74	53.51	36.48	27.88	24.86
UBC-200M			110.57	60.23	43.66	30.53	31.14
UBC-400M				121.81	73.91	36.71	25.53
UBC-640M					102.49	70.78	36.08

is noteworthy that our normalized heuristic achieves this goal by significantly increasing the cluster density.

Even though the Louvain method achieves a comparable or slightly higher modularity than our implementations, it is interesting to note that this improvement comes with an increase in the total number of non-singleton clusters. This shows that the modularity driven codes could detect large number of small clusters in the interest of increasing the modularity. However, from a clustering point of view, it is desirable to keep the number of output clusters smaller in order to reduce the burden on the downstream processing of clusters. It is also noteworthy that our normalized implementation is able to increase the size of the largest cluster by factors between 2x to 3x compared to the Louvain method, and doing so by still maintaining a high cluster density.

Finally, Table II shows the effect of incorporating edge weight information into the normalized heuristic method is minimal (please compare Unweighted, Normalized vs. Weighted, Normalized). More specifically, adding edge weight information results only in a marginal improvement in modularity and number of singletons recruited, while maintaining density. This effect, however, is input dependent. For this particular sequence homology graph, the graph construction procedure retained only those edges corresponding to a strong similarity based on a predefined cutoff. We expect the edge weight information to play a more significant role in clustering decisions for inputs where there is a wider divergence in the quality of edges.

B. Performance results

Table III shows the runtime of our implementation as a function of the input and system sizes. Note that this new implementation uses MapReduce-MPI as opposed to Hadoop which was used in our previous implementation [20]. To enable comparison between these two platforms, we present the runtime results for our standard heuristic implementation in Table III.

As can be observed, the runtime decreases as more cores are added to the system. Also, the scaling improves as the input size increases. For the largest input size (640M) we observe near-linear scaling up to 4K cores. More notably, even for the smallest data set (25M) the implementation delivers considerable performance improvement until 512 cores, beyond which the work becomes too small to benefit from parallelism. Also note that values along the diagonal of the table are roughly constant, which is a good indicator that when the input size is doubled alongside doubling the system size, the runtime is roughly maintained. These results demonstrate the overall scaling potential of our implementation to even larger number of cores as larger inputs are analyzed. It is noteworthy that our new implementation built over the MapReduce-MPI library is significantly faster when compared to our previous implementation on Hadoop platforms. For instance, the analysis of a smaller input with only 11M edges took about 1,612 seconds on 128 cores of another Hadoop cluster [20]; whereas the same analysis on the same number of cores using our new implementation took only 41 seconds. We attribute the superior performance of the MapReduce-MPI implementation to its ability to perform in-core operations — i.e., if the intermediate temporal data owned by processor fits within an allocated pages of memory, the library operates on the data in-core with no disk files are written or read [18].

We also evaluated the performance of our normalized parallel implementation separately on both weighted and unweighted inputs. Table IV shows the run-times for both inputs for varying number of cores. Note that the implementation scales near-linearly up to the system size of 1,024 cores tested on both weighted and unweighted inputs. Furthermore, as to be expected from the effect of normalization, the runtimes for the weighted and unweighted inputs were nearly identical. The marginal increase in the runtimes for the weighted inputs can be attributed to the increased number of floating point operations performed during the summation of edge weights prior to normalization.

All the tests that were run in our experiments for this paper

TABLE IV. RUN-TIME (IN SECONDS) FOR OUR NORMALIZED HEURISTIC FOR VARYING SYSTEM SIZES ON THE WEIGHTED AND UNWEIGHTED INPUTS FOR UBC-25M.

# cores	*Time*	
	Unweighted, Normalized	*Weighted, Normalized*
128	450.55	458.73
256	237.96	249.42
512	126.87	132.22
1024	73.64	78.81

are using our parallel implementation that assumes that the input graph is made available as an adjacency list. However, we also re-implemented our previously developed edge-list based algorithm for Hadoop systems [20] using MapReduce-MPI[1], and subsequently compared their performance. Our results (not shown here due to space constraints) show that the adjacency list implementation was consistently twice as fast as the edge list implementation, and is also more memory efficient. More importantly, our results led us to conclude that the adjacency list based implementation is better suited for the MapReduce-MPI model where it is desirable to keep the volume of the intermediate <key,value> pairs as small as possible so as to enable in-core access.

IV. CONCLUSION

In this paper, we presented a new variant of the shingling heuristic for clustering weighted and unweighted biological graphs. This new algorithmic heuristic is designed to overcome the qualitative problems encountered in the standard version, and also extends its application to weighted graphs. When applied to real world metagenomics graphs, our implementation demonstrated significant improvements in quality than the previous version, and in performance under the MapReduce parallel model. From a practical standpoint, it is desirable for a clustering method to detect as fewer number of non-singleton clusters and singletons as possible, while maintaining a highly density within clusters. The newly proposed clustering heuristic using normalization in this paper has been demonstrated to be highly effective in all these respects. Future line of research includes comparison against other clustering methods such as MCL, incorporating cluster composition information as part of qualitative study, extension to enumerate possibly overlapping clusters, and a comprehensive application on larger real world graphs with billions of edges.

V. ACKNOWLEDGMENTS

This research was supported by DOE award DE-SC-0006516. This research used resources of the National Energy Research Scientific Computing Center, supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] Andersen, R. and Chellapilla, K. (2009) 'Finding dense subgraphs with size bounds', *Lecture Notes in Computer Science*, Vol. 5427, pp.25–36.

[2] Andreopoulos, B., Wang, X. and Schroeder, M. (2009) 'A roadmap of clustering algorithms: finding a match for a biomedical application', *Briefings in Bioinformatics*, Vol. 10, No. 3, pp.297–314.

[3] Bader, G. and Hogue, C. (2003) 'An automated method for finding molecular complexes in large protein interaction networks', *BMC Bioinformatics*, Vol. 4, No. 2, pp.1471–2105.

[4] Blondel, V.D., Guillaume, J., Lambiotte., R., and Lefebvre., E. (2008) 'Fast unfolding of communities in large networks', *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 2008, pp.P10008.

[5] Broder, A.Z., Charikar, M., Frieze, A., and Mitzenmacher, M. (2000) 'Min-wise independent permutations', *Journal of Computer and System Sciences*, Vol. 60, pp.630–659.

[6] Brohee, S. and Helden, J.V. (2006) 'Evaluation of clustering algorithms for protein-protein interaction networks', *BMC Bioinformatics*, Vol. 7, pp.488.

[7] Enright, A.J., Van Dongen, S. and Ouzounis, S.A. (2002) 'An efficient algorithm for large-scale detection of protein families', *Nucleic Acids Research*, Vol. 30, No. 7, pp.1575–1584.

[8] Feige, U., Kortsarz, G. and Peleg, D. (2001) 'The dense k-subgraph problem', *Algorithmica*, Vol. 29, pp.410–421.

[9] Gibson, D., Kumar, R. and Tomkins, A. (2005) 'Discovering large dense subgraphs in massive graphs', *Proceedings of the International Conference on Very Large Data Bases*, pp.721–732.

[10] Handelsman, J. (2004) 'Metagenomics: application of genomics to uncultured microorganisms', *Microbiology and Molecular Biology Reviews*, Vol. 68, pp.669–685.

[11] Jeong, H., Mason, S.P., Barabasi, A. and Oltvai, Z.N. (2001) 'Lethality and centrality in protein networks', *Nature*, Vol. 411, pp.41–42.

[12] Khuller, S., Saha, B., Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S. and Thomas, W. (2009) 'On Finding Dense Subgraphs', *Automata, Languages and Programming*, Springer Berlin / Heidelberg, pp.597–608.

[13] Lancichinetti, A. and Fortunato, S. (2009) 'Community detection algorithms: A comparative analysis', *Phys. Rev. E*, Vol. 80, No. 056117.

[14] Lee, V.E., Ruan, N., Jin, R. and Aggarwal, C. (2010) 'A Survey of Algorithms for Dense Subgraph Discovery', *Managing and Mining Graph Data*, Springer US, pp.303–336.

[15] Ma, H. and Zeng, A. (2003) 'The connectivity structure, giant strong component and centrality of metabolic networks', *Bioinformatics*, Vol. 19, pp.1423–1430.

[16] Newman. M.E.J., and Girvan, M. (2004) 'Finding and evaluating community structure in networks', *Phys. Rev. E*, Vol. 69, No. 2, pp. 026113.

[17] Olman, V., Mao, F., Wu, H. and Xu, Y. (2007) 'A parallel clustering algorithm for very large data sets', *IEEE/ACM Transaction on Computational Biology and Bioinformatics*, Vol. 5, No. 2, pp.344–352.

[18] Plimpton, S. J. and Devine, K. D (2011) 'MapReduce in MPI for Large-Scale Graph Algorithms', *Parallel Computing*, Vol. 37, pp.610–632.

[19] Plimpton, S. J. and Devine, K. D (2011) http://mapreduce.sandia.gov/index.html, Last date accessed: April 2013.

[20] Rytsareva, I. and Kalyanaraman, A. (2011) 'An efficient MapReduce algorithm for parallelizing large-scale graph clustering', *Proc. ParGraph'11 - Workshop on Parallel Algorithms and Software for Analysis of Massive Graphs*, Held in conjunction with HiPC'11, Bengaluru, India.

[21] Rytsareva, I. Chapman, T. and Kalyanaraman, A. (2013) 'Parallel algorithms for clustering biological graphs on distributed and shared memory architectures', *International Journal of High Performance Computing and Networking: Special issue on Architectures and Algorithms for Irregular Applications (IJHPCN)*, In Press, 2013.

[22] Wu, C. and Kalyanaraman, A. (2008) 'An efficient parallel approach for identifying protein families in large-scale metagenomic data sets', *Proceedings ACM/IEEE conference on Supercomputing*, pp.1–10.

[23] Wu, C., Kalyanaraman, A., and Cannon, W.R. (2012) '*pGraph*: Efficient parallel construction of large-scale protein sequence homology graphs', *IEEE Transactions on Parallel and Distributed Systems*, Preprint, DOI *10.1109/TPDS.2012.19*.

[24] Yooseph, S., Sutton, G., Rusch, D.B. *et al.* (2007) 'The Sorcerer II Global Ocean Sampling expedition: expanding the universe of protein families', *PLoS Biology*, Vol. 5, No. 3, pp.432–466.

[1] Specific details about the differences between these two implementations are omitted due to space constraints