Large Scale Metagenomic Sequence Clustering via Sketching and Maximal Quasi-clique Enumeration on Map-Reduce Clusters

Xiao Yang†, Jaroslaw Zola†, *Senior Member, IEEE*, and Srinivas Aluru, *Fellow, IEEE*

Abstract—Taxonomic clustering of species from millions of DNA fragments sequenced from their genomes is an important and frequently arising problem in metagenomics. High-throughput next generation sequencing is enabling the creation of large metagenomic samples, while at the same time making the clustering problem harder due to the short sequence length supported and sampling of hitherto unknown species. In this paper, we present a parallel algorithm for taxonomic clustering of large metagenomic samples with support for overlapping clusters. We develop sketching techniques akin to those created for web document clustering to deduce significant similarities between pairs of sequences without resorting to expensive all vs. all comparison. We formulate the metagenomic classification problem as that of maximal quasi-clique enumeration in the resulting similarity graph, at multiple levels of the hierarchy as prescribed by different similarity thresholds. We cast execution of the underlying algorithmic steps as applications of the map-reduce framework to achieve a cloud ready implementation. Apart from solving an important problem in metagenomics, this work presents a map-reduce algorithm for finding quasi-cliques in graphs, and demonstrates the applicability of map-reduce in relatively complicated algorithmic settings.

Index Terms—Map-reduce algorithms, parallel applications, computational biology.

1 Introduction

Metagenomics is the study of a population of organisms by fragmenting and sequencing their collective DNA [43]. It is typically applied to communities of microbial organisms sampled from their native environments where species-wise separation is difficult, expensive, or downright impossible. Such studies are essential for identifying and discovering novel genes, studying ecosystems, and inferring the impact of microbial composition on host species. As with other areas of genomics, high-throughput next generation DNA sequencing [1], [30] is replacing Sanger sequencing due to its enormous advantages in cost, throughput, and scale of data generation. In the case of metagenomics, this comes with one distinct handicap – most bacterial genes are small enough to be fully contained in a Sanger read (up to a 1000 bp of DNA), making the task of gene identification easier. This advantage is lost due to the short read lengths supported by next-generation sequencing.

Among the next-generation sequencing techniques, the 454 technology [48] is particularly suitable for metagenomic analysis [2], [31], [39], [40]. With average length about 400 bp, 454 reads are about a third to half the length of Sanger reads, but still long enough for reliable gene identification. An important concern in surveying metagenomic samples is the clouding out of low abundance species by highly abundant species. The significantly higher throughput of 454 technology (10-20 million reads per run) facilitates deep coverage sequencing, and helps uncover numerous new species with low abundance in an environmental sample, hence, outperforming Sanger sequencing in gene discovery. Most other next-generation sequencing techniques have even higher throughputs (for example Illumina HiSeq 2000 can produce 2 billion reads [52]), but produce much shorter reads (25-150 bp), making sequence homology inference less reliable [24].

One of the important challenges in metagenomics is understanding species diversity. It is known that there is a strong mutual interplay between microbial communities and their environments [2], [43]. At the same time only a fraction of genes discovered in metagenomic samples can be mapped to known species. Consequently, quantification of species abundance as per hierarchical taxonomic units, where each taxonomic unit is a group of organisms that belong to the same defined biological type such as genus, family or order, is of paramount importance. The abundance of a taxonomic unit in the metagenomic sample is estimated by computing the ratio of the number of DNA reads belonging to the

† *X. Yang and J. Zola should be regarded as joint first authors.*

- *Xiao Yang is with the Broad Institute of Harvard and MIT, Cambridge, MA 02142.*
- *Jaroslaw Zola is with the Rutgers Discovery Inormatics Institute, Rutgers University, Piscataway, NJ 08854.*
- *Srinivas Aluru is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, and with the Department of Computer Science and Engineering, Indian Institute of Technology Bombay, India.*
- *E-mail: xiaoyang@broadinstitute.org, jaroslaw.zola@rutgers.edu, aluru@iastate.edu*

organisms of this unit over the total number of sequenced reads. Usually the reads are derived from the 16S ribosomal DNA (rDNA) pool of the sample. These serve as a good proxy for profiling abundance since they are conserved among organisms within a species while diverging across species.

Two approaches have been pursued in addressing this problem: The first relies on a database of known 16S rDNA sequences, for instance the Ribosomal Database Project [55]. The quantification process is carried out by aligning every sequenced DNA read to the database, and assigning it to the taxonomic unit based on taxonomic classification derived from the corresponding sequences in the database [10], [11], [13], [18], [24], [27], [29], [42]. This method is obviously limited to currently known organisms, which are far from comprehensive [4], [29]. In fact, advances in sequencing technology are expected to be used to further this knowledge and uncover many hitherto unknown species. Hence, the second approach advocated is to perform direct clustering or binning of the reads based on pairwise homologies [23]. It is expected that carefully calibrated alignment thresholds will cluster sequences at different levels of the hierarchy of taxonomic classification.

Current methods for metagenomic clustering are deficient in both the scale of data they can handle and the quality of clustering. This is particularly an issue for terascale metagenomics projects [50] – 400 Mbp to 600 Mbp data could be collected via the recent 454 Roche GS Titanium system within 10 hours [48]. Among the existing methods, Cd-hit [22] can handle relatively large data sets. However, it is intended to cluster sequences that are highly similar. Other methods, such as sketching techniques [7] and parallel clustering or alignment algorithms [19], [20], [47], were proposed to scale to large data sets. But none are applicable to metagenomic data clustering. The former can only cluster highly similar sequences, and the single linkage clustering technique used in [19], [20] would cause different taxonomic units to be non-differentiable. In addition, in metagenomic data analysis, the clustering problem itself has not been defined precisely so far.

In this paper, we propose the first formal model for the metagenomic clustering problem and present parallel algorithms for computing it. We make two key contributions: the first is the development of a sketching technique for DNA sequences to compute significant pairwise homologies without resorting to $O(n^2)$ time all vs. all alignments, where n is the number of input sequences. The second contribution is a parallel algorithm for enumerating quasi-cliques in a graph. While both techniques are developed in the context of the matagenomics application, we expect them to be more broadly applicable. We achieve taxonomic clustering at multiple levels through setting acceptable threshold values on the homologies. Similarly, our quasi-clique enumeration algorithm can be run iteratively to update cliques in response to addition of edges in the underlying graphs. We cast both the presented algorithms in the map-reduce framework [9] and develop a cloud-enabled software for metagenomic clustering named CLOSET (CLoud Open SequencE clusTering). The map-reduce paradigm has been drawing the attention of the computational biology community [28], [36], [41], particularly in the last few years. However, majority of current applications are limited to simply distributing the data which is then handled by existing sequential software. In this work we demonstrate that even relatively complex algorithms can utilize map-reduce framework. Although we illustrate our clustering framework with an application to 16S rDNAs, it is a general purpose framework that is suitable for the upcoming ultra-long read technologies, for instance, single molecule real time sequencing [15].

The rest of this paper is organized as follows. In Section 2 we provide a brief introduction and formalize the clustering problem. Section 3 contains an outline of our proposed approach, followed in Section 4 by the parallel algorithms we developed for this problem. Section 5 contains details of how the algorithms are translated into a map-reduce implementation. Experimental results are presented in Section 6, and Section 7 concludes the paper.

2 Problem Formulation and Computational Modeling

Scientists categorize living organisms in the form of a taxonomic hierarchy. From a computer science perspective, this is a hierarchical tree where each level corresponds to a taxonomic rank, e.g. phylum, class, order, family or genus. At each rank (level), a node is a group of organisms that belong to the same biological type, termed a *taxonomic unit* (TU). Taxonomic units at higher levels in the tree contain one or more taxonomic units at the next lower level, giving a tree structure to the classification with the leaf nodes corresponding to individual species. To quantify a TU in an environmental sample amounts to calculating the ratio of the number of individuals belonging to this TU over the total number of organisms in the same sample. The problem is easy if each individual in a metagenomic sample can be identified properly. However, this information is only indirectly available through reads taken from individual genomes of different organisms in the sample. In our case, the input is a set of reads from 16S rDNAs, which we take as providing strong but imperfect signatures for identifying the underlying organisms.

Given a set of 454 reads with average length around 400 bp, which are partial sequences derived from the full length 16S rDNAs ($\sim$1600 bp) of microbial organisms in an environmental sample, the goal is to quantify the composition of taxonomic units at different taxonomic ranks. The number of reads belonging to the same taxonomic unit defines its prevalence in the sample. With majority of these reads coming from undocumented organisms, one cannot associate reads with organisms directly. Rather, we indirectly infer through alignment

if reads are coming from the same organism or from multiple organisms within the same taxonomic unit. This task is achieved via clustering. In general, reads are more similar at a lower taxonomic rank (e.g., genus) than at a higher taxonomic rank (e.g., family).

To tackle the above problem, we assume the availability of a pairwise similarity function such that two reads of the same taxonomic unit can be differentiated from those belonging to different taxonomic units at the same taxonomic rank. The function is not expected to be (indeed it cannot be) perfect but provide trustworthy differentiation in a good majority of the cases. Examples of such functions include sequence alignment, secondary structure alignment, or their approximations. After the similarity relationships have been established, reads can be grouped together using different clustering algorithms [46].

Computationally, we need to address the following two tasks. Given a set of reads $R = \{r_1, r_2, \ldots, r_n\}$ as input:

1) Identify all pairs (r_i, r_j) such that $F(r_i, r_j) \geq t$, where F is the chosen similarity function and t is a threshold.
2) Cluster reads based on their established similarity association.

Current methods for metagenomic read clustering do not document a precisely defined goal for clustering but rather leave it to be inferred through the algorithmic approach. As observed in [16], diverse needs of clustering in relation to the application domains is the main reason behind the plethora of clustering algorithms developed in the literature. In case of metagenomics, single linkage clustering or hierarchical clustering techniques are widely used (e.g., [12], [23]). The major flaw of these strategies lies in their inability to properly deal with inaccuracies and ambiguities in the similarity measure, prevalent in metagenomic data analysis. Two major problems arise due to the way current methods deal with ambiguities: when a read is highly similar to reads from multiple taxonomic units, it is included in one of them leading to a wrong count, or even worse, the taxonomic units are merged into a single one (in case of single linkage clustering). Once a mistake is committed, it percolates upwards in the hierarchy of taxonomic ranks. In other words, current methods look for partitioning of the read set – meaningful only if read clustering can be made accurately. Note that the correct similarity score threshold that differentiates a taxonomic unit successfully from all others at the same rank is unknown.

To address these issues, we model the clustering problem as follows: we regard true clustering at a certain taxonomic rank to be a partitioning of the input reads – the ideal we seek. However, since the function F is unlikely to faithfully reflect evolution such that reads of the same taxonomic unit can be unambiguously differentiated from reads of other units, we allow a read to concurrently occur in multiple clusters when applicable. The ambiguities in read assignments are likely to be alleviated as we lower the similarity threshold. For instance, let $F(r_i, r_j) = 90\%, F(r_i, r_k) = 90\%, F(r_j, r_k) = 84\%$, and $F(r_p, r_q) = 86\%$. If we choose $t = 85\%$ as a cutoff corresponding to the genus rank and look for complete linkage clustering, three clusters can be formed: $\{r_i, r_j\}, \{r_i, r_k\}$ and $\{r_p, r_q\}$, where r_i is equally justified to be placed in two clusters. While at the family rank where t is lowered to e.g. 80%, a partition is achieved: $\{r_i, r_j, r_k\}$ and $\{r_p, r_q\}$. Ultimately, when the similarity threshold reduces to below a certain value, the input becomes a single cluster – a faithful reflection that all reads belong to the same domain. Due to false positives resulting from choosing the similarity function F, it is too stringent a requirement to expect complete linkage between every pair of reads in the same taxonomic unit. Furthermore, when the read similarity function is used, two reads from the same individual rDNA will not score highly on the similarity score if they sample different parts of the rDNA. To account for these, we propose to only enforce a certain degree of partial linkages within a cluster that grows as a function of the cluster size.

We define a cluster to be consisting of a set of reads such that there exists a sufficient number of pairwise similarities among them. Formally, a cluster is a maximal set $Q \subseteq R$ of reads such that $\left|\{(r, s) \in Q \times Q : r \neq s, F(r, s) \geq t\}\right| \geq \gamma \cdot \binom{|Q|}{2}$. Note that this definition takes into account inaccuracies in determining taxonomic unit membership based on read similarities. The parameter γ can be dialed up or down to reflect the trust in this assessment, or can be a function of the threshold t or the cluster size. The clustering problem is then defined as that of finding all maximal clusters Q at a given threshold level t. Note that the clusters need not be a partition. In addition, the clusters could be computed for a decreasing sequence of threshold values. Domain experts can then view the resulting clusters and identify the thresholds at which the clusters appear to be classifying taxonomic units at a particular rank level. For instance, one could identify thresholds at which the resulting clustering best approximates a partition. This leads to more accurate classification because the data supports the inference that clustering can be done meaningfully at these threshold levels.

3 Proposed Algorithmic Approach

We now present our algorithmic approaches for computing pairwise read similarities and subsequent clustering. The typical practice in solving the first task is to compute all pairwise scores, which has $O(\mathcal{C}n^2)$ complexity, where $\mathcal{C}$ is the time taken by the function $F(r_i, r_j)$. This is a well studied problem with sequential and parallel solutions [35], which nevertheless remain computationally expensive for large n. Different clustering algorithms try to circumvent this problem using different strategies. For example, Cd-hit [22] greedily searches for clusters

among sequences, however, its worst-case time complexity remains $O(n^2)$. Note that an overwhelming majority of F function evaluations result in unconnected pairs of reads, even at higher taxonomic ranks (the highest taxonomic rank where all reads would be in one cluster is not considered). In this paper, we avoid all pairwise similarity computations and propose a sketching based algorithm to directly infer pairs whose estimated similarity scores exceed a given threshold. Our technique relies on adapting sketching techniques particularly those used for web documents clustering [5], [26], where all vs. all comparisons are clearly impractical.

Our algorithm takes a decreasing sequence of similarity cutoffs $T = (t_1, t_2, \ldots, t_m)$ as input. For each cutoff t_k, we perform clustering based on pairs of reads (r_i, r_j) such that $F(r_i, r_j) > t_k$. We formalize the above strategy using graph theory. Let $G^k = (V, E^k)$ be a set of undirected graphs ($1 \leq k \leq m$) on the same set of nodes, where vertex $v_i \in V$ denotes read $r_i \in R$ for $1 \leq i \leq n$. Let $W : V \times V \rightarrow [0, 1]$ denote a weight function over potential edges where $w_{ij} = F(r_i, r_j)$. Edge set E^k is defined as: $e_{ij} = (v_i, v_j) \in E^k$ if and only if $w_{ij} > t_k$. Note that $E^{k-1} \subseteq E^k$ ($1 < k \leq m$). The weights of edges are immaterial in quasi-clique enumeration. However, they are needed to determine the presence or absence of edges in each graph G^k. Since t_m is the lowest threshold, it is sufficient to store W only for (i, j) pairs where $F(r_i, r_j) \geq t_m$, or in case t_m is not known *a priori*, then for (i, j) pairs where $F(r_i, r_j) \geq t$ for a sufficiently conservative lower bound $t \leq t_m$. The edge set E^k for each graph can then be computed from the stored values of the W function. Viewed another way, W is computed over edges in E^m, or a superset of edges of E^m (denoted by E from hereonwards) since t_m may not be known in advance.

We compute clustering on graphs $G^1, G^2, \ldots, G^m$, in that order. We add edges incrementally in going from one graph to the next. In what follows, the superscript is dropped for convenience and each graph is simply referred to as G with the threshold becoming clear from the context. A cluster is a maximal quasi-clique in G: let $U \subseteq V$; the U-induced subgraph $G' = (U, E_U)$ is a γ-quasi-clique if $|E_U| \geq \gamma \cdot \binom{|U|}{2}$, for $0 < \gamma \leq 1$.

4 Algorithms for Metagenomic Clustering

In this section, we present our algorithms for 1) edge construction and validation, and 2) clustering via incremental maximal quasi-clique enumeration. The map-reduce realization of these algorithms will be described in detail in the next section. In what follows, we use the notation $\langle key, value \rangle$ to denote a key and value pair.

4.1 Edge Construction and Validation

Our algorithm for edge construction is adapted from the sketching technique originally proposed for web based document clustering [5]. We briefly recall here the relevant key ideas from document clustering. Initially, each document D_i is converted to its corresponding tuple set S_i, where each tuple in S_i is a sequence of k consecutive words in D_i. Then, a universal hash function is utilized to map every tuple in S_i uniformly to the space of integers (usually 64-bit). After hashing, D_i has been converted to a set of integers, denoted by H_i. The Jaccard similarity coefficient between D_i and D_j, defined as $\frac{|H_i \cap H_j|}{|H_i \cup H_j|}$, has been shown [6] to be equivalent to the probability that the minimum values of H_i and H_j are the same. Theoretically, to derive the Jaccard similarities among a set of highly similar documents, it is sufficient to use the above strategy to compare the extracted minimum values from each of them. These extracts are termed *sketches*, and using the sketches to pair documents that share the same sketch avoids all-pair comparisons. Nonetheless, in practice, the clustering results are greatly influenced by the chosen hash function. The accuracy of using this technique degrades when used to cluster documents that are less similar [17]. At the same time, more sketches can be chosen (e.g., via a modular function) to better represent each document under comparison.

For metagenomic read clustering, we are facing the challenge to cluster not only reads with high similarities (e.g., 95%) but also reads with much lower similarities (e.g., 75%) to be able to classify microbial organisms at different taxonomic ranks. In addition, reads are in the DNA alphabet of size just 4, compared to the much larger alphabet set used in documents, and a large number of reads may share a common substring. In addition, DNA sequences are continuous strings with no word decomposition as in written text. Hence, the concept of word is replaced by substring of fixed length k, referred to as a k-mer. Our solution for adapting sketching techniques to metagenomic reads is given in Fig. 1. Even though presented as a serial algorithm for convenience, it is designed such that the individual steps can be easily executed in parallel. Details of parallel execution using the map-reduce framework are deferred to the next section.

Each read in $R = \{r_1, r_2, \ldots, r_n\}$ is initially converted to a set of integers comprising of the hash value of every constituent k-mer (substring of length k) (line 1). Instead of choosing the minimum value to represent a read r_i, we select a subset S_i of hash values that are l modulo M, where M is a preset constant (line 3). The similarity between reads r_i and r_j is computed as $|S_i \cap S_j|$ (derived in lines 4–14) divided by $\min(|S_i|, |S_j|)$. Our design of this similarity function is motivated by the need to capture containment relationships, and account for differences in read lengths. Note that if read r_i is a substring of read r_j, then $S_i \subseteq S_j$, resulting in a perfect similarity score of 100% as desired.

To avoid the $O(n^2)$ complexity in computing the similarity function for each pair, we let common hash values

1: For each read $r_i \in R$, hash every constituent k-mer to a 64-bit integer, to obtain a hash set $H_i = \{h_1, h_2, \ldots\}$.
2: **for** $l = 0$ to $M - 1$ **do**
3: Generate the l^{th} sketch for each r_i: $S_i = \{h_j \mid h_j \in H_i \wedge (h_j \mod M) = l\}$.
4: For each S_i, and each $h_j \in S_i$ generate $\langle h_j, rID_i \rangle$, where rID_i is the unique identifier assigned to r_i. Let *SR* denote the list of all pairs so generated.
5: Let n_j denote the number of elements in *SR* with key h_j. Let C_{max} be a user specified threshold.
6: **for** every unique key h_j of *SR* **do**
7: **if** $n_j \leq C_{max}$ **then**
8: For all pairs of $\langle h_j, rID_a \rangle, \langle h_j, rID_b \rangle \in SR$, such that $rID_a \neq rID_b$, generate $\langle (rID_a, rID_b), 1 \rangle$.
9: **else**
10: Retain all *rID*s sharing the same key h_j as an entry in the list SR_{rem}.
11: **end if**
12: **end for**
13: Merge all entries generated in line 8 with the same key by summing up the corresponding values.
14: Update every pair $\langle (rID_a, rID_b), count \rangle \in SR$ by incrementing count by one each time rID_a, rID_b concurrently belong to an entry in SR_{rem}.
15: For each $\langle (rID_i, rID_j), count \rangle \in SR$, derive similarity score $J_{ij} = \frac{count}{\min(|S_i|, |S_j|)}$. If $J_{ij} \geq C_{min}$, where C_{min} is the user specified similarity value, add the pair $\langle rID_i, rID_j \rangle$ to the candidate list L_l.
16: **end for**
17: Let $\mathbf{L} = \bigcup_{l=0}^{M-1} L_l$.
18: For every $\langle rID_i, rID_j \rangle \in \mathbf{L}$, let $w_{ij} = F(r_i, r_j)$. If $w_{ij} \geq t$ add $\langle (rID_i, rID_j), w_{ij} \rangle$ to the final list describing set E.

Fig. 1: The edge construction and validation algorithm.

dictate which pairs to evaluate. Each common hash value shared between a pair of reads causes the generation of that pair with a frequency count of one. The frequency counts are later aggregated to reflect the number of common hash values. However, since DNA alphabet is small, some common k-mers may appear even between reads that do not share significant similarity. In particular, short repeats or elements that frequently appear in multiple DNAs can cause a significant throwback to the $O(n^2)$ complexity by creating many pairs with low frequency counts that will later be eliminated. To avoid this, we postpone using high frequency k-mers to avoid generating exceedingly large number of read pairs (line 10). This practice is also supported by biological justification: substrings common to DNAs from many organisms are not useful in differentiating among them. However, once we decide to explore similarity of two reads based on sharing of low frequency substrings, the high frequency ones have to be added back into the mixture to determine the corresponding similarity score (line 14).

The quality of results can be severely affected by sketching bias – the hash function does not guarantee to map k-mers uniformly into the integer space as assumed by the sketching technique [6]. To mitigate this effect, we apply M rounds of sketching and read pairs (i.e. graph edges) generation. In round l ($0 \leq l < M$), the sketch is composed of hash values that are l modulo M. An edge survives as long as it is generated by at least one of these sketches. In a probabilistic sense, this exponentially decreases the chance that a read pair is completely missed. On the flip side, many edges are expected to be identified multiple times. The storage issues concerning this can be mitigated by combining the set of pairs for each successive sketch on the fly with the pairs seen so far, rather than combine them all at once as mathematically indicated in line 17.

Finally, the entire exercise of generating read pairs based on sketching can be seen as a filter to produce pairs worthy of further evaluation. Any user defined similarity function F can then be applied to assess the generated read pairs directly. However, the sketch-based function we designed is accurate enough to be directly used in practice. If so, line 18 of the algorithm is unnecessary, and t can be used in place of C_{min} in line 15. Thus, we are able to provide a standalone solution for metagenomics clustering, while at the same time providing flexibility to the user to specify any arbitrary similarity function of their choice and immediately benefit from a highly efficient parallel implementation.

4.2 Incremental Quasi-clique Enumeration

Exact maximal (quasi-) clique enumeration has been extensively studied in the literature and there have been parallel algorithms designed for this problem using both message passing and shared memory paradigms [14], [37], [45], as well as the map-reduce framework [44]. Due to the irregular graph structures arising in practice and the existence of an exponential number of maximal (quasi-) cliques in such graphs, available methods are typically applied to random graphs or graphs of relatively smaller sizes than the ones emerging in the current application. Therefore, we designed an approximate maximal quasi-clique enumeration strategy suitable for large metagenomic graphs.

To cluster reads at different taxonomic ranks, we will use a series of similarity cutoffs $T = (t_1, t_2, \ldots, t_m)$, sorted in the decreasing order, and the corresponding graphs $G^1, G^2, \ldots, G^m$. The results can be obtained by performing maximal quasi-clique enumeration to identify clusters on each graph G^k independently. However, this is wasteful because each succeeding graph G^k constitutes addition of edges to the previous graph G^{k-1}. Hence, a maximal quasi-clique in G^{k-1} will continue to be a quasi-clique in G^k, but the addition of edges may render it non-maximal. Thus, our algorithms starts with the maximal quasi-cliques generated for G^{k-1}, treats each of the newly added edges as a clique of size 2, and performs clique merging iteratively to form maximal

cliques for G^k. Using a series of similarity cutoffs gives domain experts the flexibility to view different resulting clusters and identify the thresholds at which the clusters would be more meaningful in the context of a particular application.

Our method is presented in Fig. 2. Each cluster c (a maximal γ-quasi-clique) is denoted by a pair: $\langle key, value \rangle$, where the key field, denoted by $c.key$, represents the set of vertices in c, and the value field, denoted by $c.value$, represents the set of edges in this cluster. Initially, every edge with similarity score higher than given threshold is considered a clique (lines 6–9). Then, two clusters that share common vertices are joined together if a larger γ-quasi-clique can be formed (lines 11–17). Note that our algorithm offers no theoretical guarantee that it enumerates all maximal γ-quasi-cliques. In fact, the number of such cliques can be exponential in the worst case. Instead, our algorithm is a heuristic to generate maximal quase-cliques appropriate to the problem at hand. We do not expect arbitrary input but given that the reads come from organisms which should fall into groups at various taxonomic ranks, the clique generation process is expected to discover these groupings. Note that the above notation for a cluster is for ease of presentation. In practice, the vertices need not to be stored explicitly but can be inferred from the edges, and identification of clusters with a common set of nodes can be implemented without testing all c_i, c_j pairs.

1: Let $T = (t_1, t_2, \ldots, t_m)$ for $t_1 > t_2 > \ldots > t_m$, and $t_0 = 100\%$.
2: $k \leftarrow 1$
3: Let C denote the clustering result, $C \leftarrow \emptyset$.
4: **while** $k \leq m$ **do**
5: **for** every $e_{ij} \in E$ **do**
6: **if** $t_{k-1} \geq w_{ij} > t_k$ **then**
7: $c.key \leftarrow \{rID_i, rID_j\}$; $c.value = \{(rID_i, rID_j)\}$
8: $C = C \cup \{c\}$
9: **end if**
10: **end for**
11: **repeat**
12: Identify $c_i, c_j \in C$ such that $c_i.key \cap c_j.key \neq \emptyset$.
13: $n_1 \leftarrow c_i.key \cup c_j.key$; $n_2 \leftarrow c_i.value \cup c_j.value$
14: **if** $|n_2| \geq \gamma \cdot \binom{|n_1|}{2}$ **then**
15: $C = C \setminus \{c_i, c_j\}$; $C = C \cup \{\langle n_1, n_2 \rangle\}$
16: **end if**
17: **until** no change in C is observed.
18: $k \leftarrow k + 1$
19: **end while**

Fig. 2: Algorithm for maximal quasi-clique enumeration.

5 Map-Reduce Implementation

We implemented our clustering algorithm using the map-reduce framework. This choice was motivated by several practical considerations. Typical metagenomic data sets consist of gigabytes of data and impose serious I/O and storage requirements. At the same time available map-reduce implementations, for instance Apache Hadoop [51], deliver highly efficient distributed file systems (e.g., HDFS) that remove complexity of managing I/O explicitly, and hide its low-level details from programmers. Because map-reduce applications are typically executed in large cluster environments that have high probability of a single node failure, map-reduce implementations provide strong fault-tolerance mechanisms. Finally, map-reduce applications can be easily deployed in popular cloud environments, for instance Amazon EC2 [49], making them more accessible to users who are not high performance computing experts. The later argument seems to be especially noteworthy, taking into account increasing interest of biology researchers in cloud-enabled solutions.

The aforementioned advantages of map-reduce come at a price of constrained flexibility, since algorithms must be expressed as a series of map and reduce stages, through which the input data is streamed. While this pattern is sufficient for many embarrassingly parallel applications, it becomes challenging, and sometimes infeasible, for more algorithmically difficult problems.

To express our clustering procedure using the map-reduce paradigm, we designed a series of data transformations, where each transformation is a single map-reduce task, and output of one task is streamed as an input to the next one. Below we provide a detailed description of each task specifying its input as well as map and reduce functions. For convenience we use the same notation as in Figs. 1 and 2. Additionally, we use keywords **Map**, **Reduce** and **emit** following their standard meaning in the map-reduce framework [9]. Finally, we assume that the input data, i.e. the set of reads R, is initially stored as pairs $\langle rID_i, r_i \rangle$, which is required by map-reduce.

5.1 Edge Construction

Our implementation of the edge construction stage uses five tasks outlined in Fig. 3. Note that some parts of our algorithm can be directly cast as map-reduce computation, while others require more involved data conversion to enable efficient execution in the map-reduce setup.

We start by generating a list of candidate edges with similarity at least C_{min}. This part of the algorithm naturally translates to map-reduce and in our implementation is achieved by repeatedly executing (for M iterations) *Task 1* and *Task 2*. These tasks correspond to the main loop in Fig. 1.

The goal of *Task 1* is to create a list of reads that share common k-mers. For a given read r_i mapper performs hashing of its k-mers, and identifies sketches taking into account iteration counter l and parameter M. Although in the original algorithm k-mer hashing is performed outside the main loop, in our implementation we merged it into the sketch generation phase for sake of simplicity. Each generated sketch is used as a key associated with

Task 1: Sketch Selection

Input: Read $\langle rID_i, r_i \rangle$, parameters M, l, C_{max}.
Map: Generate sketch set S_i of read r_i for given M and l. For each sketch $s_j \in S_i$ **emit** $\langle s_j, rID_i \rangle$.
Reduce: For each key s get list $L = [rID_i, rID_j, \ldots]$ of reads sharing sketch s. If $|L| \leq C_{max}$ **emit** $\langle |L|, L \rangle$. Otherwise, store L in a temporary file.

Task 2: Edge Generation

Input: Pair $\langle z, L = [rID_{i1}, rID_{i2}, \ldots, rID_{iz}] \rangle$, parameter C_{min}.
Map: For each read pair (rID_i, rID_j), $rID_i, rID_j \in L$, $i \neq j$, **emit** $\langle (\min(rID_i, rID_j), \max(rID_i, rID_j)), 1 \rangle$.
Reduce: For each key (rID_i, rID_j) get list $L = [q_1, q_2, \ldots]$ of corresponding values.
Let $count = \sum_{q_x \in L} q_x$.
If rID_i and rID_j are both present in the temporary file generated in *Task 1*, increase $count$ by one.
Compute J_{ij} as described in Fig. 1, line 15.
If $J_{ij} \geq C_{min}$ **emit** $\langle rID_i, rID_j \rangle$.

Task 3: Redundant Edges Removal

Input: Predicted edge $\langle rID_i, rID_j \rangle$.
Map: emit input entry as is.
Reduce: For each key rID_i get list of its adjacent nodes $L = [rID_{j1}, rID_{j2}, \ldots]$. For each unique $rID_j \in L$ **emit** $\langle rID_i, rID_j \rangle$ and **emit** $\langle rID_j, rID_i \rangle$.

Task 4: Data Aggregation

Input: Input read $\langle rID_i, r_i \rangle$ or candidate edge $\langle rID_a, rID_b \rangle$.
Map: emit input entry as is.
Reduce: For each key rID_i assign its corresponding read to r, get list of its adjacent nodes $L = [rID_{j1}, rID_{j2}, \ldots]$, then **emit** $\langle rID_i, (r, L) \rangle$.

Task 5: Edge Validation

Input: Edge data $\langle rID_i, (r_i, L = [rID_{j1}, rID_{j2}, \ldots]) \rangle$, parameter t.
Map: For each read $rID_j \in L$
emit $\langle (\min(rID_i, rID_j), \max(rID_i, rID_j)), r_i \rangle$.
Reduce: For each key (rID_i, rID_j) get r_i and r_j.
If $F(r_i, r_j) \geq t$ **emit** $\langle (rID_i, rID_j), F(r_i, r_j) \rangle$.

Fig. 3: Map-reduce tasks implementing the edge construction and validation algorithm.

the read from which it has been derived, so that reducer can directly aggregate all reads containing given sketch. The list so obtained is then emitted as output, to be processed by the next task, or if its size is greater than C_{max} it is stored in a temporary file. Collectively, the output and temporary files generated by reducers in *Task 1* represent lists SR and SR_{rem}, respectively.

Given a list of reads that share common sketches the purpose of *Task 2* is to directly estimate Jaccard index between all implied read pairs. To compute the estimate for a single pair the number of its shared sketches is required. This problem is similar to the classic task of counting words using map-reduce [9] and can be solved in exactly the same way. Therefore, each mapper in *Task 2* enumerates possible "words", which in our case are pairs of reads with common sketches, and reducers perform counting. The resulting counts are further updated with the data from temporary files generated in *Task 1*. Given this information, reducers can compute Jaccard index for each pair, and emit all pairs with the similarity score above the threshold C_{min}. Note that the final output of *Task 2* represents list L_l of candidate edges.

As explained earlier when describing the edge construction and validation algorithm, many edges will be enumerated in more than one of M iterations of *Task 1* and *Task 2*. Therefore in *Task 3* we remove redundant edges. However, to enable efficient implementation of the validation stage we replace each undirected edge with two corresponding directed edges. Notice that conceptually this task corresponds to line 17 in Fig. 1, with the exception that each undirected edge in list $\mathbf{L}$ is now represented twice.

The idea behind such data transformation is to facilitate successive edge validation steps. Recall that map-reduce does not provide explicit support for random access to secondary storage. Instead the data has to be streamed through mappers and reducers. In order to perform edge validation however we need to access both: information about edges, that is output of *Task 3*, and actual read sequences required to compute function F. At the same time accessing each read sequence directly from the hard drive, using for instance the `libhdfs` interface [53], would generate significant overhead and hence adversely affect its performance in practice. Therefore to overcome this limitation we divided the edge validation stage into two tasks, wherein *Task 4* we bring together edge and sequence data and in *Task 5* we compute function F.

In *Task 4* for every read that has been selected to form a candidate edge we group its adjacent nodes and its sequence. We concurrently stream reads from the original input data and candidate edges generated by *Task 3*. Note that both data sets use read identifier as a key. Consequently mappers can simply forward each input entry directly to reducers, which then perform data aggregation. Once all values for a given key are combined, reducer can directly identify the one value that

represents read sequence. What is important, because during merging in the previous step every undirected edge has been replaced by two complementary directed edges, the output of *Task 4* will include read sequence for every unique read present in the set of candidate edges. Moreover, because input sequences are streamed rather than randomly accessed, we maintain efficiency at the cost of increased secondary storage use.

Having both edges and sequences in place, the final task is to compute function F performing validation. This step is implemented in *Task 5*. Observe that the output of *Task 4* is in fact an augmented adjacency list, in which every read is described by its sequence and a list of adjacent nodes. Such data representation makes the following conversion easy to perform by a mapper. For a given read all edges to which it is incident are generated. Each resulting edge is represented by an ordered pair of reads, and the same ordering is implemented by all mappers. The ordered pairs are used as keys associated with sequences of reads for which they have been generated. By imposing ordering we remove directionality information as now each edge is represented twice by the same key. However, both copies are associated with different read sequence, one for each incident read. Hence, together they provide all data required to evaluate function F for a given edge. The evaluation is performed by a reducer, which then checks if computed score is above the desired threshold t, and emits edge and its score accordingly.

Task 6: Edge Filtering

Input: Input edge $\langle (rID_i, rID_j), w_{ij} \rangle$, parameter t_k.
Map: If $t_k \leq w_{ij}$ **emit** $\langle rID_i, rID_j \rangle$.
Reduce: For each key rID_i get list of its adjacent nodes $L = [rID_{j1}, rID_{j2}, \ldots]$ and **emit** $\langle rID_i, L \rangle$.

Task 7: Quasi-clique Generation

Input: Adjacency list $\langle rID_i, L = [rID_{j1}, rID_{j2}, \ldots,] \rangle$ or cluster $\langle h, c = [(rID_{a1}, rID_{b1}), (rID_{a2}, rID_{b2}), \ldots] \rangle$, and parameter γ.
Map: For each read $rID_j \in L$ **emit** $\langle rID_i, (rID_i, rID_j) \rangle$ and **emit** $\langle rID_j, (rID_i, rID_j) \rangle$. For each read rID_i adjacent to edge from c **emit** $\langle rID_i, c \rangle$.
Reduce: For each key rID_i get list $\mathcal{L} = [c_{j1}, c_{j2}, \ldots]$ of clusters containing rID_i. If $c_j, c_k \in \mathcal{L}$, $j \neq k$, can be merged to form a γ-quasi-clique **emit** $\langle \mathsf{h}(c_j \cup c_k), c_j \cup c_k \rangle$.

Task 8: Merging Clusters

Input: Cluster $\langle h_i, c = [(rID_{a1}, rID_{b1}), (rID_{a2}, rID_{b2}), \ldots] \rangle$.
Map: emit input entry as is.
Reduce: For each key h_i get list $\mathcal{L} = [c_{j1}, c_{j2}, \ldots]$ of clusters with the same hash h_i. Let $\mathbf{C} = \bigcup_{c_j \in \mathcal{L}} c_j$. **emit** $\langle \mathsf{h}(\mathbf{C}), \mathbf{C} \rangle$.

Fig. 4: Map-reduce tasks implementing the maximum quasi-clique enumeration algorithm.

5.2 Quasi-cliques Enumeration

The output of *Task 5* represents a weighted undirected graph in which we want to perform maximal quasi-clique enumeration. Fig. 4 shows tasks we have designed to express our enumeration algorithm using map-reduce. Our implementation consists of three tasks that are executed for each threshold $t_k \in T$. *Task 6* performs edge filtering, while *Task 7* and *Task 8* are responsible for iterative quasi-clique generation – equivalent of the internal loop in Fig. 2, line 11–17.

We start the entire process with *Task 6* in which we convert a list of edges having similarity score higher than t_k to an adjacency list. This procedure is executed only once for each similarity threshold, and its output is streamed to *Task 7* during the first iteration of the internal loop. Mappers in *Task 7* convert newly introduced edges into clusters and merge them with clusters identified in earlier iterations. Note that each cluster is described by a unique label, which is generated by a hashing function h based on read identifiers which are contained by the cluster. In order to detect clusters that share common nodes, and thus may form a γ-quasi-clique, mappers in *Task 7* create a mapping between reads and clusters containing them. Next, reducers aggregate all clusters shared by a given read, and merge pairs that form γ-quasi-cliques. Each new cluster created in this way is assigned an identifier and written to the output. Using the above procedure we avoid testing all possible cluster pairs as we compare only some fraction that shares common reads.

The final step is to merge quasi-cliques defined over the same set of reads. Observe that because individual mappers/reducers are always executed independently, it is possible that several different γ-quasi-cliques covering the same set of reads may be enumerated. In *Task 8* we perform merging of such quasi-cliques exploiting the fact that each quasi-clique has key assigned, which is a hash h of its nodes. During the merging step reducers take care of potential collisions, which for brevity is not indicated in Fig. 4. The output of *Task 8* is a list of clusters C.

As we already mentioned, for a given threshold both *Task 7* and *Task 8* are repeated until no further changes in the set of output clusters are possible, which concludes the enumeration process.

6 Experimental Results

6.1 Test Data

To assess efficiency of our clustering approach and its map-reduce implementation we performed a set of experiments with synthetic and real-life data.

From the Ribosomal Database Project (RDP) repository [55] Release 10 (Update 27), we extracted all high

quality annotated sequences from individual isolates. The resulting set comprised of 7,289 bacterial 16S rDNA fragments. To each sequence we assigned a taxonomic lineage from the domain level to the genus level, based on the original RDP taxonomy. Next, from each sequence we extracted a 850 bp long substring starting 200 bp downstream from the position matching the primer sequence 337F. The goal of this procedure is to extract variable regions V1 to V3 of 16S rDNA. These particular regions are generally considered to be the most informative parts of 16S rDNA, and are suitable for distinguishing bacterial species to the genus level [8], [21]. We will refer to this data set as RDPX-I.

We used RDPX-I to further simulate 454 sequencing and obtain a synthetic metagenomic sample. Classification of known species found in environmental samples indicates that their abundance may follow the power law distribution with a few high and many low abundance species [32]. In practice, abundance is manifested by the number of reads derived from each microorganism. Consequently, to obtain the data set RDPX-II we applied the following procedure. Using sampling with replacement of RDPX-I we created a pool of $n = 5{,}000{,}000$ sequences, with the property that the probability of a sequence coming from the k-th abundant species is given by $P(k) \sim k^{\beta}$, where $\beta = -0.33$ based on [32]. This results with the expected number of sequences derived from the least and the most abundant species being 460 and 8,688, respectively (see Fig. 5). Next, we employed the `454sim` tool [25] to simulate sequencing of the selected 5,000,000 16S rDNA fragments on the 454 Roche GS Titanium platform. Finally, we post-processed obtained reads by cleaving low-quality ends and removing reads shorter than 250 bp. The resulting data set comprises 4,485,498 reads with average length 420 bp, and maximum length 622 bp. Note that the above *in silico* procedure closely reproduces a typical metagenomic sequencing pipeline, while it delivers annotated data that can be readily used to assess quality of metagenomic tools.

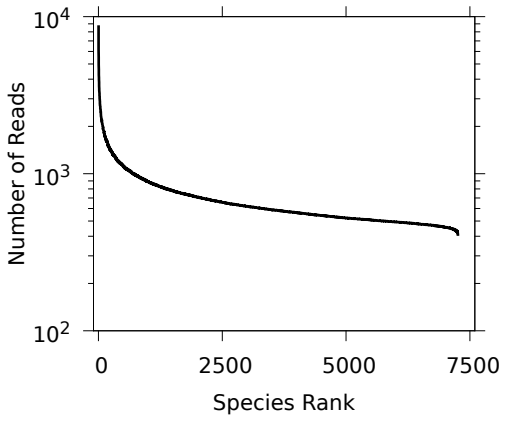

Fig. 5: Reads abundance with respect to species rank in the data set RDPX-II.

6.2 Experimental Environment

To implement map-reduce tasks described in the previous section, and to deploy a map-reduce cluster, we used the latest Apache Hadoop framework. Hadoop provides three different interfaces (Java, Pipes and Streaming), which vary in trade-off between efficiency and flexibility. Because our algorithm requires high data throughput (difficult to achieve in Hadoop Streaming), and depends on computationally intensive subroutines (hard to implement efficiently in Java), we decided to rely on C++ and Hadoop Pipes combined with the `libhdfs` library to provide direct access to Hadoop distributed file system (required to manage temporary files). Although Hadoop Pipes do not support native data types we used Ascii85 encoding to overcome this limitation. To implement hashing function used in the sketching stage we investigated several popular choices, including djb2 [3], Rabin fingerprint [33] and 64bit Murmur2 hash [54], and found the last to provide the best performance. To orchestrate map-reduce tasks we implemented additional shell scripts that keep track of iteration progress and manage intermediate data whenever required. Finally, we created a set of small tools to convert input and output data between the standard biological data formats and our internal representation. The complete software package constitutes a cloud-enabled framework, which we named CLOSET (CLoud Open SequencE clusTering).

We deployed CLOSET on a 31 node Hadoop cluster with a total of 248 GB main memory, and 5.4 TB secondary storage with average of 60 MB/s buffered read (as reported by `hdparm -t`) under the control of HDFS. The cluster has 31 nodes with dual AMD 2.2 GHz 4-core CPUs for a total of 248 cores, and uses Gigabit Ethernet for interconnect. We used a typical Hadoop configuration with one node serving as a master tracking jobs and maintaining the HDFS metadata, and remaining nodes acting as workers executing computations and storing data blocks. We set HDFS replication factor to 2, and used 64 MB data block size.

6.3 Sketching Performance

In the first set of experiments we focused on the analysis of our sketching approach, addressing two main questions: first, how sensitive is this technique, and second, how well does it scale? Here we should note that because the method never produces false edges, i.e. edges with similarity score lower than given threshold t, sensitivity is a sufficient criterion to characterize sketching accuracy. At the same time, to be scalable sketching must be able to deliver high sensitivity while performing only a small fraction of all vs. all sequence comparisons.

In our implementation of the edge construction algorithm, we use Jaccard index of k-mer spectra to obtain similarity scores between pairs of sequences. Therefore, to measure sensitivity we first computed similarity scores between all pairs of sequences in the RDPX-I data set, which is feasible taking into account its size,

and then used the resulting graph as a reference to compare graphs generated by the sketching approach, executed with different parameters. In all experiments we set $k = 20$ to generate a reliable k-mer spectrum, i.e. the one consisting of k-mers that have low probability of occurring randomly in a given read. To obtain such spectrum it is sufficient to set k such that $d \ll 4^k$, where $d = 1{,}600$ is the approximate length of 16S rDNA sequences from which our data has been derived. Finally, we set $C_{min} = t$ in all cases, and $C_{max} = 730$, which is approximately 10% of the RDPX-I size (for larger data sets we expect C_{max} to be a much smaller fraction).

TABLE 1: Sensitivity of the sketching approach for different configurations of M and threshold t. For each choice of M only 3, 5 and 9 iterations have been executed.

	$t = 0.50$			$t = 0.90$		
M	3	5	9	3	5	9
100	0.994	1	1	0.985	0.998	1
62	0.972	0.997	1	0.94	0.992	1
50	0.989	0.997	1	0.979	0.99	1
40	0.988	0.999	1	0.991	1	1
33	0.991	0.999	1	0.976	0.997	1
25	0.988	0.999	1	0.965	0.993	1
20	0.998	1	1	0.993	1	1
14	0.979	0.996	1	0.966	0.996	1
10	0.992	1	1	0.977	1	1
9	0.997	1	1	0.997	1	1

As we can see in Table 1 sketching very accurately detects similar sequences irrespective of selected threshold t, and it requires only 9 iterations to achieve 100% sensitivity. This clearly demonstrates that sketching can be a reliable replacement for all vs. all comparisons. Decreasing M, which is equivalent to increasing the number of sketches sampled from each sequence in a single iteration, seems to have only minor effect on sensitivity. However, one can expect that the effect will be more pronounced for lower threshold values, where similarity between sequences is harder to detect.

To assess sketching scalability we measured fraction of all vs. all comparisons required for a given threshold to validate predicted edges, and then fraction of predicted edges that are rejected during this step. We executed 7 iterations for different choices of M and t, each time obtaining sensitivity above 99%. Table 2 summarizes obtained results.

TABLE 2: Fraction of all vs. all comparisons during validation step and fraction of rejected edges for different choices of M and t.

t	$M = 100$	$M = 33$	$M = 9$
0.30	0.11769/0.59192	0.12385/0.54955	0.08971/0.35745
0.45	0.07908/0.82039	0.05519/0.74056	0.03236/0.55742
0.60	0.05007/0.85564	0.02607/0.72271	0.01123/0.35627
0.75	0.03257/0.93084	0.01086/0.79251	0.00591/0.61864
0.90	0.01294/0.97425	0.00474/0.92978	0.00204/0.83646

As expected the sketching approach provides significant reduction in the number of comparisons required to obtain a set of validated edges: even for very low similarity threshold only 11% of all possible comparisons are required. The gain becomes even more pronounced for higher threshold values, independent of M. Note that for larger data sets, e.g. consisting of millions of reads, the savings will be significantly higher. The fraction of rejected edges increases with the threshold value, which is not surprising taking into account that similar sequences cannot be easily differentiated without exact comparison. Finally, the performance of sketching improves with decreasing values of M. However, we should keep in mind that by decreasing M we significantly increase the overall number of sketches generated, which may cause the cost of generating candidate edges to offset savings in the validation stage. This of course will depend on the input data and the choice of function F (e.g. secondary structure alignment is significantly more computationally demanding than the standard pairwise alignment).

6.4 Clustering Performance

To assess applicability of our quasi-clique based clustering procedure we performed a set of tests using the RDPX-II data set. First, we performed sketching of RDPX-II with $k = 20$, $M = 100$ and $t = 0.75$, running for 7 iterations. Next, we performed clustering of the resulting graph with $\gamma = 0.65$ and $T = (0.96, 0.93, 0.90)$.

As we described earlier RDPX-II consist of 454 reads derived from known DNA sequences, for which complete taxonomic lineage is given. This information can be used to verify how well given clustering correlates with the "true" clustering induced by the taxonomic data. However, given predicted and reference clustering the challenge is to quantify their similarity. Although several methods have been proposed to compare partitions, e.g. Rand Index [34], no satisfactory approach exists for comparing soft clusters in which given element may belong to multiple bins at the same time. Therefore to assess the quality of obtained clustering we decided to use the following approach. Define *divergence* of cluster c as $d(c) = \frac{\log(q)}{\log(|c|)}$, where q is the number of different taxonomic units contained in c with respect to the known taxonomic lineage. For a given cluster the divergence is 0 only if all elements of that cluster belong to the same TU, and is 1 if no two elements share the same TU. Divergence captures cluster composition, however it cannot be used alone to assess clustering performance as it neglects clustering granularity. For instance clustering consisting of many small clusters will naturally tend to have a small average divergence. Therefore, we juxtapose divergence obtained for different similarity thresholds with clustering statistics. Obtained results are presented in Table 3.

There are several interesting observations in place. For all similarity threshold values t_k the maximum and average divergence decrease when moving from lower taxonomic level (genus) to the higher one (family). This is in line with what we expect as it is easier to perform clustering at higher taxonomic levels. With decreasing similarity threshold the cluster size increases which shows that by introducing new edges larger quasi-cliques are formed. Finally, smaller difference between clustering results obtained for threshold values t_1 and

TABLE 3: Cluster divergence and cluster size for clustering at different similarity and taxonomic levels.

	$t_1 = 0.96$	$t_2 = 0.93$	$t_3 = 0.90$
Divergence: *genus level*			
Maximum	0.903	1	1
Average	0.018	0.020	0.022
Deviation	0.091	0.089	0.091
Divergence: *family level*			
Maximum	0.792	0.827	0.829
Average	0.007	0.007	0.007
Deviation	0.052	0.053	0.052
Cluster size			
Minimum	2	2	2
Maximum	2,119	2,210	16,152
Average	15.408	13.270	14.578
Deviation	31.870	33.128	48.221
Median	6	6	7

t_2 than t_2 and t_3 suggests that cutoff t_3 is closer to the value representative for family level.

6.5 Computational Scalability

In the final set of experiments we tested how our map-reduce implementation scales with respect to the size of input data. From the RDPX-II data set we extracted two random subsets, RDPX-II-1M and RDPX-II-2M, consisting of 1 mln and 2 mln reads respectively (see Table 4). Next, we performed clustering of all three data sets for $k = 20$, $M = 100$, $t = 0.75$, $\gamma = 0.65$ and $T = (0.96, 0.93, 0.90)$, executing 7 iterations of sketching stage. Obtained results are summarized in Table 5.

TABLE 4: Characteristics of the data used to test CLOSET scalability.

	RDPX-II-1M	RDPX-II-2M	RDPX-II
No. reads	1,000,000	2,000,000	4,485,498
Size [MB]	556	1,137	2,551
Read length (avg/max)	420.75/619	420.78/619	420.81/622

TABLE 5: Run time in seconds for different stages of CLOSET. Total time includes intermediate data management.

	RDPX-II-1M	RDPX-II-2M	RDPX-II
Sketching	1,706	2,373	4,105
Validation	1,196	2,250	4,091
Clustering ($t_1 = 0.96$)	651	861	840
Clustering ($t_2 = 0.93$)	1,495	2,807	3,985
Clustering ($t_3 = 0.90$)	7,752	16,381	21,566
Total	12,879	24,762	34,692

Analysis of efficiency of map-reduce tasks is in general a challenging problem. Performance of any map-reduce job is strongly affected by how well different (often very obscure) parameters of the underlying map-reduce library are tuned, taking into account input data and the underlying hardware. In fact the process of map-reduce job configuration is referred to as a "black art" [38]. To perform experiments we set all typically configured Hadoop parameters (e.g., the number of reducers spawned, memory allocation to Java daemons, etc.) so as to avoid out-of-core execution and at the same time maintain the ratio of mappers to reducers that guarantees high CPU utilization. CLOSET implementation provides an easy to use mechanism to configure each map-reduce task separately. Finally, we configured all parameters with respect to the RDPX-II data set, and to make results comparable between executions, we used the same configuration to run analysis of the remaining two data.

Table 5 shows that both sketching and validation stage scale nearly sub-linear with the size of data, which indicates excellent performance of our sketching approach. The time required to perform validation is comparable to the time taken by the sketching phase. This result however may depend on the input data, the choice of similarity function F, and configuration of CLOSET parameters. For a given data set the cost of clustering increases exponentially for decreasing similarity threshold values. This is not surprising taking into account exponential increase in the number of quasi-cliques that have to be processed when more edges are introduced into clustering. Finally, the total execution time increases linearly with the size of input data. This result is data specific – the number of edges with similarity above the threshold t_3 increases only slightly from one data set to another. For significantly lower thresholds the total execution time will be dominated by the clustering stage and will be increasing exponentially with the size of input data. Note however, that in our application such low threshold values have no practical justification from the biological perspective.

7 Conclusions

Clustering of large-scale environmental samples is considered a difficult and challenging problem in metagenomics. Our work is motivated by the inability of existing methods and software to scale to large data sets and lack of formal computational modeling and elegant algorithmic solutions. This paper represents our effort to satisfactorily address all the above problems and issues. We presented a rigorous framework for metagenomic clustering based on sketching and iterative quasi-clique enumeration. Our framework has the following advantages: it can accommodate arbitrary user-defined similarity functions, and by performing clustering at many threshold levels, it can guide researchers in tuning clustering to better fit different taxonomic ranks. Our software is implemented on widely available Hadoop platform with the map-reduce framework. It can easily handle some of the largest data collections that are currently being generated for metagenomic studies. While the presented algorithms are motivated by this specific problem in metagenomics, we proposed two novel techniques that have much broader applicability. One is the development of sketching techniques for the realm of genomics. The other is a parallel algorithm for quasi clique enumeration.

Acknowledgments

This research is supported in part by the NSF under grant no. DMS-1120597.

REFERENCES

[1] W. Ansorge. Next-generation DNA sequencing techniques. *Nat. Biotechnol.*, 25(4):195–203, 2009.

[2] A.K. Benson et al. Individuality in gut microbiota composition is a complex polygenic trait shaped by multiple environmental and host genetic factors. *Proc. Natl. Acad. Sci. USA*, 107(44):18933–18938, 2010.

[3] D. Bernstein. Hash??? Not quite clear on what this is... comp.lang.c, 1990.

[4] N. Blow. Metagenomics: Exploring unseen communities. *Nature*, 453(7195):687–690, 2008.

[5] A. Broder et al. Syntactic clustering of the web. *Comput. Networks ISDN Systems*, 29(8–13):1157–1166, 1997.

[6] A. Broder et al. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630 – 659, 2000.

[7] M. Cameron, Y. Bernstein, and H.E. Williams. Clustered sequence representation for fast homology search. *J. Comput. Biol.*, 14(5):594–614, 2007.

[8] S. Chakravorty et al. A detailed analysis of 16S ribosomal RNA gene segments for the diagnosis of pathogenic bacteria. *J. Microbiol. Methods*, 69(2):330–339, 2007.

[9] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proc. of Symp. on Opearting Systems Design & Implementation*, volume 6, pages 1–10, 2003.

[10] T.Z. DeSantis et al. Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB. *Appl. Environ. Microbiol.*, 72(7):5069–5072, 2006.

[11] T.Z. DeSantis et al. NAST: A multiple sequence alignment server for comparative analysis of 16S rRNA genes. *Nucleic Acids Res.*, 34(Web Server Issue):W394–W399, 2006.

[12] T.Z. DeSantis et al. NAST: A multiple sequence alignment server for comparative analysis of 16S rRNA genes. *Nucleic Acids Res.*, 34(Web Server Issue):W394–W399, 2006.

[13] N. Diaz et al. TACOA: Taxonomic classification of environmental genomic fragments using a kernelized nearest neighbor approach. *BMC Bioinf.*, 10:56, 2009.

[14] N. Du et al. A parallel algorithm for enumerating all maximal cliques in complex network. In *Proc. of Int. Conf. on Data Mining Workshops*, pages 320–324, 2006.

[15] J. Eid et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[16] V. Estivill-Castro. Why so many clustering algorithms: a position paper. *ACM SIGKDD Explorarion Newsletter*, 4(1):65–75, 2002.

[17] M. Henzinger. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 284–291, 2006.

[18] D. Huson et al. MEGAN analysis of metagenomic data. *Genome Res.*, 17(3):377–386, 2007.

[19] A. Kalyanaraman et al. Efficient clustering of large EST data sets on parallel computers. *Nucleic Acids Res.*, 31(11):2963–2974, 2003.

[20] A. Kalyanaraman et al. Assembling genomes on large-scale parallel computers. *J. Parallel Distrib. Comput.*, 67(12):1240–1255, 2007.

[21] M. Kim, M. Morrison, and Z. Yu. Evaluation of different partial 16S rRNA gene sequence regions for phylogenetic analysis of microbiomes. *J. Microbiol. Methods*, 84(1):81–87, 2011.

[22] W. Li and A. Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.

[23] W. Li, J. C Wooley, and A. Godzik. Probing metagenomics by rapid cluster analysis of very large datasets. *PLoS One*, 3(10):e3375, 2008.

[24] Z. Liu et al. Accurate taxonomy assignments from 16S rRNA sequences produced by highly parallel pyrosequencers. *Nucleic Acids Res.*, 36(18):e120, 2008.

[25] F. Lysholm, B. Andersson, and B. Persson. An efficient simulator of 454 data using configurable statistical models. *BMC Res. Notes*, 4(1):449, 2011.

[26] G. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proc. of WWW Conf.*, pages 141–150, 2007.

[27] A. McHardy et al. Accurate phylogenetic classification of variable-length DNA fragments. *Nat. Methods*, 4(1):63–72, 2007.

[28] A. McKenna et al. The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, 20(9):1297–1303, 2010.

[29] F. Meyer et al. The metagenomics RAST server – a public resource for the automatic phylogenetic and functional analysis of metagenomes. *BMC Bioinf.*, 9(1):386, 2008.

[30] J.M. Perkel. Sanger who? Sequencing the next generation. *Science*, 10:275–279, 2009.

[31] H. Poinar et al. Metagenomics to paleogenomics: Large-scale sequencing of mammoth DNA. *Science*, 311(5759):392–394, 2006.

[32] J. Qin et al. A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464(7285):59–65, 2010.

[33] M.O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Harvard University, 1981.

[34] W.M. Rand. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.*, 66(336):846–850, 1971.

[35] A. Sarje, J. Zola, and S. Aluru. Accelerating pairwise computations on Cell processors. *IEEE Trans. Parallel Distrib. Syst.*, 22(1):69–77, 2011.

[36] M. Schatz, B. Langmead, and S. Salzberg. Cloud computing and the DNA data race. *Nat. Biotechnol.*, 28(7):691–693, 2010.

[37] M.C. Schmidt et al. A scalable, parallel algorithm for maximal clique enumeration. *J. Parallel Distrib. Comput.*, 69(4):417–428, 2009.

[38] S. Sharma. Advanced Hadoop tuning and optimization, 2009.

[39] C. Simon et al. Phylogenetic diversity and metabolic potential revealed in a glacier ice metagenome. *Appl. Environ. Microbiol.*, 75(23):7519–7526, 2009.

[40] P. Turnbaugh et al. A core gut microbiome in obese and lean twins. *Nature*, 457(7228):480–484, 2009.

[41] D. Wall et al. Cloud computing for comparative genomics. *BMC Bioinf.*, 11:259, 2010.

[42] Q. Wang et al. Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.*, 73(16):5261–5267, 2007.

[43] J.C. Wooley, A. Godzik, and I. Friedberg. A primer on metagenomics. *PLoS Comput. Biol.*, 6(2):e1000667, 2010.

[44] B. Wu et al. A distributed algorithm to enumerate all maximal cliques in mapreduce. In *Proc. of Int. Conf. on Frontier of Computer Science and Technology*, pages 45–51, 2009.

[45] B. Wu and X. Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Emerging technologies in knowledge discovery and data mining*, volume 4819 of *Lecture Notes in Computer Science*, pages 476–483, 2009.

[46] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Trans. Neural Networks*, 16(3):645–678, 2005.

[47] J. Zola et al. Parallel-TCoffee: A parallel multiple sequence aligner. In *Proc. of ISCA Int. Conf. on Parallel and Distributed Computing Systems*, pages 248–253, 2007.

[48] 454 Life Sciences. http://www.454.com/.

[49] Amazon EC2. http://aws.amazon.com/ec2/.

[50] Earthmicrobiome Project. http://www.earthmicrobiome.org/.

[51] Hadoop. http://hadoop.apache.org/.

[52] Illumina Systems/HiSeq 2000. http://www.illumina.com/systems/hiseq_2000.ilmn.

[53] Hadoop LibHDFS. http://wiki.apache.org/hadoop/LibHDFS.

[54] MurmurHash. http://sites.google.com/site/murmurhash/.

[55] Ribosomal Database Project. http://rdp.cme.msu.edu/.