Exploring neighborhoods in large metagenome assembly graphs reveals hidden sequence diversity

C. Titus Brown[a,1], **Dominik Moritz**[b], **Michael P. O'Brien**[c], **Felix Reidl**[c], **Taylor Reiter**[a], **and Blair D. Sullivan**[c,1]

[a]Department of Population Health and Reproduction, University of California Davis; [b]Paul G. Allen School of Computer Science and Engineering, University of Washington; [c]Department of Computer Science, NC State University

This manuscript was compiled on November 5, 2018

Genomes computationally inferred from large metagenomic data sets are often incomplete and may be missing functionally important content and strain variation. We introduce an information retrieval system for large metagenomic data sets that exploits the sparsity of DNA assembly graphs to efficiently extract subgraphs surrounding an inferred genome. We apply this system to recover missing content from genome bins and show that substantial genomic sequence variation is present in a real metagenome. Our software implementation is available at https://github.com/spacegraphcats/spacegraphcats under the 3-Clause BSD License.

metagenomics | sequence assembly | strain variation | bounded expansion | dominating set

Metagenomics is the analysis of microbial communities through shotgun DNA sequencing, which randomly samples many subsequences (reads) from the genomic DNA of each microbe present in the community. A common problem in metagenomics is the reconstruction of individual microbial genomes from the mixture. Typically this is done by first running an assembly algorithm that reconstructs longer linear regions based on a graph of the sampled subsequences (1), and then binning assembled contigs together using compositional features and genic content (2, 3). These "metagenome-assembled genomes" can then be analyzed for phylogenetic markers and metabolic function. In recent years, well over 10,000 metagenome-assembled genomes have been published, dramatically expanding our view of microbial life (4–8).

During the process of metagenome assembly and binning, some sequence present in the data may go unbinned due either to a failure to assemble (9, 10) or a failure to bin. The underlying technical reasons for these failures include low coverage, high sequencing error, high strain variation, and/or sequences with insufficient compositional or genic signal. Recent work has particularly focused on the problem of strain confusion, in which high strain variation results in considerable fragmentation of assembled content in mock or synthetic communities (9, 10); the extent and impact of strain confusion in real metagenomes is still unknown but potentially significant (11–13).

Associating missing metagenomic sequence to inferred bins or known genomes is technically challenging. Current approaches to extending bins use manual inspection of sequence read pairs or assembly graphs and do not scale to many genomes (14, 15). In pangenome studies with the goal of investigating strain variants of known genomes, current approaches use mapping or k-mer baiting, in which assembled sequences are used to extract reads or contigs from a metagenome (16–19). These methods fail to recover genomic content that does not overlap with the query, such as large indels or novel genomic islands.

Missing content may in some instances be connected in the assembly graph to known sequence, in which case it is possible to recover it via a graph-based neighborhood query (14, 15). A graph query approach allows the recovery of sequence distinct from but connected to extant genomic regions, as well as sequence from regions that do not assemble well but are graph-proximal to the query (15). However, many graph query algorithms are NP-hard and hence computationally intractable in the general case; compounding the computational challenge, metagenome assembly graphs are frequently large, with millions of nodes, and require 10s to 100s of gigabytes of RAM.

In this paper, we develop and implement a scalable graph query framework that crucially exploits the structural sparsity of compact De Bruijn assembly graphs in order to compute a succinct index data structure in linear time. We use this framework to perform neighborhood queries in large assembly graphs, which enables us to extract the genome of a novel bacterial species, recover missing sequence variation in amino acid sequences for genome bins, and identify missing content for metagenome-assembled genomes. Our methods are assembly-free and avoid error correction and other techniques that may discard strain information. These algorithms are available in an open-source Python software package, spacegraphcats (20).

Results

Dominating sets enable efficient neighborhood queries in large assembly graphs. We design and implement (20) a set of algorithms for efficiently finding content in a metagenome that is close to a query as measured by distance in a compact De Bruijn graph (cDBG) representation of the sequencing data (Figure 1). To accomplish this, we first organize the cDBG into *pieces* around a set of *dominators* that are collectively close to all vertices. In this context, the *neighborhood* of a query is the union of all pieces it overlaps; to enable efficient search, we build an invertible index of the pieces.

We compute dominators so that the minimum distance from every vertex in the cDBG to some dominator is at most r (an *r-dominating set*) using Algorithm 1, which is based on the linear-time approximation algorithm given by Dvořák and Reidl (21). Although finding a minimum r-dominating set is NP-hard (22–24) and an approximation factor below $\log n$ is probably impossible (23) in general graphs, our approach guarantees constant-factor approximations in linear running time by exploiting the fact that (compact) De Bruijn graphs

DM, MPO, FR, and BDS designed and implemented algorithms; CTB, DM, MPO, and FR developed software; CTB and TR conducted biological data analysis; CTB and BDS supervised work. All authors interpreted results, wrote text, created figures, and approved the submitted paper.

[1]To whom correspondence should be addressed. E-mail: ctbrown@ucdavis.edu, vbsulliv@ncsu.edu

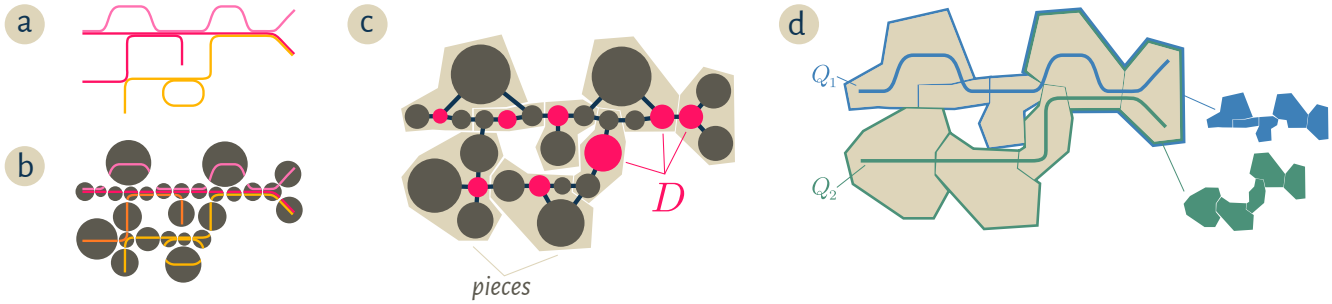

Fig. 1. Starting from a collection of genomic sequences (**a**), we form an assembly graph where nodes represent distinct linear subsequences (**b**). In this assembly graph, known as a *compact De Bruijn graph* (3), nodes may represent many k-mers. The original genomic sequences correspond to walks in the graph, and shared nodes between the walks represent shared subsequences. We then (**c**) identify a subset of nodes D called a *dominating set* so that every node in the assembly graph is at distance at most one from some member of D (marked pink). We further partition the graph into *pieces* by assigning every node to exactly one of the closest members of D (beige regions in (**c**) and (**d**)). For a genomic query Q, the *neighborhood* of Q in this graph is the union of all pieces which share at least one k-mer with the query. The colorful subsets of the pieces in (**d**) correspond to the neighborhoods of the queries Q_1, Q_2.

have *bounded expansion*, a special type of sparsity (25). Algorithm 1 first annotates the graph to determine the distances between all pairs of vertices at distance at most r (lines 1-3) and then uses these distances to ensure each vertex is close to a dominator. The core of the efficient distance computation is based on *distance-truncated transitive fraternal (dtf) augmentations* (21) which produce a directed graph in which each arc uv is labeled with $\omega(uv)$, the distance from u to v in the original cDBG.

Importantly, our implementation enhances the algorithm in (21) by computing only $r-1$ rounds of dtf-augmentations instead of $2r-1$. Since augmentation is the computationally most expensive part of the pipeline and the running time depends non-linearly on the number of rounds, this vastly improves this algorithm's scalability. To maintain approximation guarantees on the dominating set size with fewer augmentations, we introduce a threshold parameter `domThreshold`(r) which affects the constant factor in our worst-case bound. We selected a threshold (see Appendix) that produces r-dominating sets of comparable size to those computed by the algorithm in (21). Additionally, we found that processing vertices using a minimum in-degree ordering (line 6) was superior to other common orders (e.g. arbitrary, min/max total degree, max in-degree).

After computing an r-dominating set D of G with Algorithm 1, Algorithm 2 partitions the vertices of G into pieces so that each piece $P[v]$ contains a connected set of vertices for which v is the closest member of D (Figure 1). Finally, we use minimal perfect hashing (`mphfIndex`) (26) to compute an invertible index* between pieces and their sequence content in the metagenome.

One feature of this approach is that the dominating set and index only need to be computed once for a given metagenome, independent of the number and content of anticipated queries. Queries can then be performed using Algorithm 3 in time that scales linearly with the size of their *neighborhood* – all genomic content assigned to pieces that contain part of the query.

Our implementations of these algorithms in spacegraphcats can be run on metagenomic data with millions of cDBG nodes; (Table 1) indexing takes under an hour, enabling queries to complete in seconds to minutes (Table 2). See Appendix A for full benchmarking (including cDBG construction). This provides us with a tool to systematically investigate assembly graph neighborhoods.

Algorithm 1 `rdomset`(G, r)

Input: Graph G, radius r
Output: r-dominating set D of G

1: $\vec{G}_1 \leftarrow$ `orient`(G)
2: **for** $i \in 2, \ldots, r$ **do**
3: $\quad \vec{G}_i \leftarrow$ `dtfAugmentation`$(\vec{G}_{i-1})$
4: Initialize $d[v] \leftarrow \infty$ and $c[v] \leftarrow 0$ for all $v \in G$
5: $D \leftarrow \emptyset$
6: **for all** $v \in \vec{G}_r$ sorted by ascending in-degree **do**
7: $\quad$ **for all** $u \in N^-(v)$ **do**
8: $\quad\quad d[v] \leftarrow \min(d[v], d[u] + \omega(uv))$
9: $\quad$ **if** $d[v] > r$ **then**
10: $\quad\quad D \leftarrow D \cup \{v\}$ and $d[v] \leftarrow 0$
11: $\quad\quad$ **for all** $u \in N^-(v)$ **do**
12: $\quad\quad\quad d[u] \leftarrow \min(d[u], \omega(uv))$
13: $\quad\quad\quad c[u] \leftarrow c[u] + 1$
14: $\quad\quad\quad$ **if** $c[u] >$ `domThreshold`(r) **then**
15: $\quad\quad\quad\quad D \leftarrow D \cup \{u\}$ and $d[u] \leftarrow 0$
16: $\quad\quad\quad\quad$ **for all** $w \in N^-(u)$ **do**
17: $\quad\quad\quad\quad\quad d[w] \leftarrow \min(d[w], \omega(wu))$
18: **return** D

Algorithm 2 `indexPieces`$(\mathcal{M}, r)$

Input: Metagenome $\mathcal{M}$, radius r
Output: Invertible index $I\colon \mathcal{M} \to \mathcal{P}$; $\mathcal{P}$ is a set of pieces

1: $G \leftarrow$ `cDBG`$(\mathcal{M})$
2: $D \leftarrow$ `rdomset`(G, r)
3: Initialize $\delta[v] \leftarrow v$ for all $v \in D$
4: $U \leftarrow V(G) \setminus D$
5: **while** $U \neq \emptyset$ **do**
6: $\quad$ **for** $v \in V(G) \setminus U$ **do**
7: $\quad\quad$ **for** $u \in N(v) \cap U$ **do**
8: $\quad\quad\quad \delta[u] \leftarrow \delta[v]$
9: $\quad\quad\quad U \leftarrow U \setminus \{u\}$
10: $\mathcal{P}[v] \leftarrow \{u : \delta[u] = v\}$
11: $I_{\mathcal{M}} \leftarrow$ `mphfIndex`$(\mathcal{M}, \mathcal{P})$
12: **return** $I_{\mathcal{M}}$

*an invertible function that defines both an index and the corresponding inverted index

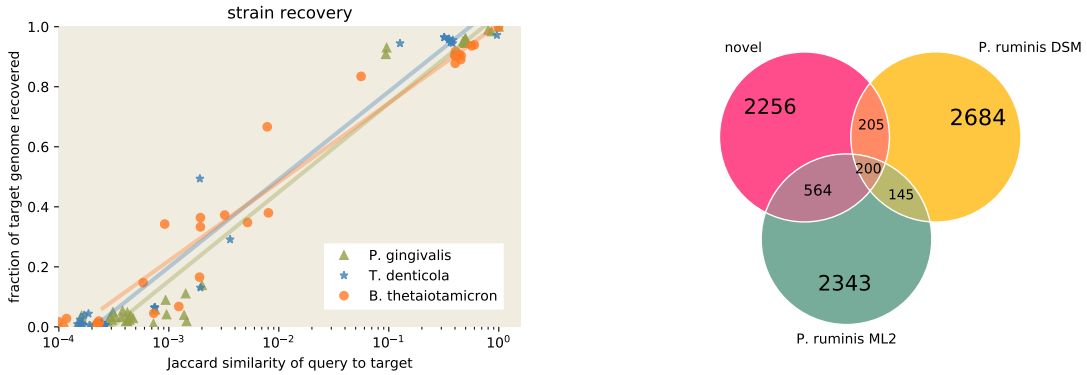

Fig. 2. Neighborhood queries enable recovery of relevant genomic content. (a) Left Panel: Recovery of each of three target genomes from `podarV` using queries at a variety of Jaccard distances from the target. Recovery is calculated as containment of target genome in query neighborhood. (b) Right Panel: Recovery of novel *Proteiniclasticum* content from `podarV`. Nucleotide content from two of the three known *P. ruminis* genomes overlapped approximately a megabase of sequence in the query neighborhood, which also contained approximately 2.3 Mbp of unknown sequence; the third known genome, *P. ruminis CGMCC*, was omitted from the figure as it is 99.7% similar to *P. ruminis DSM*. Numbers are in thousands of k-mers.

Algorithm 3 `search`$(I_{\mathcal{M}}, Q)$

Input: Index $I_{\mathcal{M}}$, Query Q
Output: The query neighborhood $\mathcal{N}_Q$
1: $\mathcal{N}_Q \leftarrow \bigcup_{k \in Q} I_{\mathcal{M}}^{-1}(I_{\mathcal{M}}(k))$
2: **return** $\mathcal{N}_Q$

Dataset	$\lvert V\rvert$	$\lvert E\rvert/\lvert V\rvert$	$\lvert D\rvert$	$\overline{\lvert Q\rvert}$	$\overline{\lvert \mathcal{P} \cap \mathcal{N}_Q\rvert}$
`podarV`	916 041	2.2	542 350	1 475 892	4 106
`HuSB1`	13 852 950	2.6	6 724 505	1 112 516	106 091

Table 1. Number of cDBG nodes $|V|$, edge density of cDBG $|E|/|V|$, size of 1-dominating set $|D|$, average query size (k-mers) $\overline{|Q|}$, and average number of pieces in query neighborhood $\overline{|\mathcal{P} \cap \mathcal{N}_Q|}$. Queries are the 51 genomes and 23 genome bins fully present in `podarV` and `HuSB1`, respectively.

Dataset	Algorithm	Time (s)	Memory (MB)
	`rdomset`	78.1	4 304
`podarV`	`indexPieces`	359.9	14 108
	`search`	14.9	3 463
	`rdomset`	1 181.1	60 238
`HuSB1`	`indexPieces`	859.3	40 713
	`search`	67.9	15 228

Table 2. Time and memory usage of spacegraphcats for Algorithms 1-3 on representative metagenome data. The times for Algorithm 3 are averaged over all queries (see Table 1). Statistics reported for Algorithm 2 exclude lines 1-2 of pseudocode. Times are rounded to the nearest tenth of a second; memory is rounded to the nearest Megabyte.

Neighborhood queries enable recovery of relevant unknown genomic content. We first measure how well an inexact query can recover a target genome from a metagenome. For a benchmark data set, we use the `podarV` data set (27). This is a "mock" metagenome containing genomes from 65 strains and species of bacteria and archaea, each grown independently and rendered into DNA, then combined and sequenced as a metagenome. This metagenome is a commonly used benchmark for assembly (10, 28–30).

To evaluate the effectiveness of neighborhood query at recovering strain variants, we choose three target genomes from `podarV`– *Porphyromonas gingivalis ATCC 33277*, *Treponema denticola ATCC 35405*, and *Bacteroides thetaiotaomicron VPI-5482* – that have many taxonomically close relatives in Genbank. We then use these relatives to query the `podarV` mixture and measure the recovery of the target genome. The results, in Figure 2(a), show that graph neighborhood query can recover 35% or more of some target genomes starting from a relative with Jaccard similarity as low as 1%: even a small number of shared k-mers suffice to define a much larger neighborhood that contains related genomes.

We next apply neighborhood query to retrieve an unknown genome from `podarV`. Several papers have noted the presence of unexpected sequence in the assemblies of this data, and Awad et al. identify at least two species that differ from the intended mock metagenome contents (10, 28). One species variant has partial matches to several different *Fusobacterium nucleatum* genomes, while the other incompletely matches to three strains of *Proteiniclasticum ruminis*.

The complete genomes of these two variants are not in public databases and, for the *Proteiniclasticum* variant, cannot be entirely recovered with existing approaches: when we assemble the reads that share k-mers with the available genomes, a marker-based analysis with CheckM estimates that 98.8% of the *Fusobacterium* variant is recovered, while only 72.96% of the *Proteiniclasticum* variant is recovered. We therefore try using neighborhood queries to expand our knowledge of the *Proteiniclasticum* variant.

We perform a neighborhood query into `podarV` with all three known *Proteiniclasticum* genomes from GenBank. We then extract the reads overlapping this neighborhood and assemble them with MEGAHIT. The retrieved genome neighborhood for *Proteiniclasticum* contains 2256K novel k-mers (Figure 2(b)). The reads from the query neighborhood assemble into a 3.1 Mbp genome bin. The assembly is estimated by CheckM to be 100% complete, with 10.3% contamination. The mean amino acid identity between *P. ruminis ML2* and the neighborhood assembly is above 95%, suggesting that this is indeed the genome of the *Proteiniclasticum* variant, and that neighborhood query retrieves a full draft genome sequence (see

Appendix A).

Query neighborhoods in a real metagenome do not always assemble well. Real metagenomes may differ substantially from mock metagenomes in size, complexity, and content. In particular, real metagenomes may contain a complex mixture of species and strain variants (31) and the performance of assembly and binning algorithms on these data sets is challenging to evaluate in the absence of ground truth. One recent comparison of single-cell genomes and metagenome-assembled genomes in an ocean environment found that up to 40% of single-cell genome content may be missing in metagenome-assembled genomes (13).

We first ask whether genome query neighborhood sizes in a real metagenome differ from mock metagenomes. We examine genomes inferred from the SB1 sample from the Hu et al. (2016) study, in which 6 metagenomic samples taken from various types of oil reservoirs were sequenced, assembled, binned, and computationally analyzed for biochemical function (32). Examining the 23 binned genomes in GenBank originating from the SB1 sample, we compare the `HuSB1` neighborhood size distribution with the `podarV` data set (Figure 3(a)). We see that more genome bins in `HuSB1` have 1.5x or larger query neighborhoods than do the genomes in `podarV`. This suggests the presence of considerably more local neighborhood content in the real metagenome than in the mock metagenome.

We next investigate the metagenomic content in the query neighborhoods. As with the unknown variants in `podarV`, we use CheckM to estimate genome bin completeness. The estimated bin completeness for many of the query genomes is low (Appendix A). To see if the query neighborhoods contain missing marker genes, we assemble reads from the query neighborhoods using MEGAHIT. However, we find little improvement in the completion metrics (Figure 3(b)).

Investigating further, we find that the query neighborhood assemblies contain only between 4% and 56% of the neighborhood k-mer content (Appendix A), suggesting that MEGAHIT is not including many of the reads in the assembly of the query neighborhoods. This could result from high error rates and/or high strain variation in the underlying reads (9, 10).

To attempt to recover more gene content from the assemblies, we turn to the Plass amino acid assembler (33). Plass implements an overlap-based amino acid assembly approach that is considerably more sensitive than nucleotide assemblers and could be more robust to errors and strain variation (34).

When we apply Plass to the reads from the query neighborhoods, we see a further increase in neighborhood completeness (Figure 3(b)). This suggests that the genome bin query neighborhoods contain real genes that are accessible to the Plass amino acid assembler. We note that these are unlikely to be false positives, since CheckM uses a highly specific Hidden Markov Model (HMM)-based approach to detecting marker genes (35).

Some query neighborhoods contain substantial strain variation. If strain variation is contributing to poor nucleotide assembly of marker genes in the query neigborhoods, then Plass should assemble these variants into similar amino acid sequences. Strain variation for unknown genes can be difficult to study due to lack of ground truth, but highly conserved proteins should be readily identifiable.

The *gyrA* gene encodes an essential DNA topoisomerase that participates in DNA supercoiling and was used by (32) as a phylogenetic marker. In the GenBank bins, we find that 15 of the 23 bins contain at least one gyrA sequence (with 18 *gyrA* genes total). We therefore use *gyrA* for an initial analysis of the Plass-assembled neighborhood content for all 23 bins. To avoid confounding effects of random sequencing error in the analysis and increase specificity at the cost of sensitivity, we focus only on high-abundance data: we truncate all reads in the query neighborhoods at any k-mer that appears fewer than five times, and run Plass on these abundance-trimmed reads from each neighborhood. We then search the gene assemblies with a gyrA-derived HMM, align all high-scoring matches, and calculate a pairwise similarity matrix from the resulting alignment.

When we examine all of the high-scoring gyrA protein matches in the hard-trimmed data, we see considerable sequence variation in some query neighborhoods (Figure 4(a)). Much of this variation is present in fragmented Plass assemblies; when the underlying nucleotide sequences are retrieved and used to construct a compact De Bruijn graph, the variation is visible as spurs off of a few longer paths (insets in Figure 4(a)). When we count the number of well-supported amino acid variants in isolated positions (i.e. ignoring linkage between variants) we see that ten of the 23 neighborhoods increase in the number of gyrA genes, with four neighborhoods gaining a gyrA where none exists in the bin (Appendix A; see lowest inset in Figure 4(a) for one example). Only one neighborhood, *M. bacterium*, loses its gyrA genes, due to the stringent k-mer abundance trimming. Collectively, the use of the Plass assembler on genome neighborhoods substantially increases the number of gyrA sequences associated with bins.

We see this same pattern for many genes, including alaS, gyrB, rpb2 domain 6, recA, rplB, and rpsC (Appendix A). This shows that multiple variants of those proteins are present within at least some of the neighborhoods and implies the presence of underlying nucleotide strain variation. This strain variation may be one reason that nucleotide assembly performs poorly: on average, only 19.6% of Plass-assembled proteins are found within the nucleotide assemblies.

Query neighborhoods assembled with Plass contain additional functional content. In addition to capturing variants close to sequences in the bins, we identify many novel genes in the query neighborhoods. We use KEGG to annotate the Plass-assembled amino acid sequences, and subtract any homolog already annotated in the genome bin. We also ignore homolog abundance so that each homolog is counted only once per neighborhood.

Novel functional content is distributed throughout pathways present in the genome bins, and increases functionality associated with binned genomes by approximately 13% (Figure 4(b)). This includes orthologs in biologically relevant pathways such as methane metabolism, which are important for biogeochemical cycling in oil reservoirs (32).

Genes in these neighborhoods contain important metabolic functionality expanding the pathways already identified in (32). We find 40 unique orthologs involved in nitrogen fixation across eight neighborhoods, four of which had no ortholog in the bin. Importantly, we find the ratio of observed orthologs approximately matches that noted in (32), where two thirds of nitrogen fixation functionality is attributable to archaea (29 of 40 orthologs). This is in contrast to most ecological systems

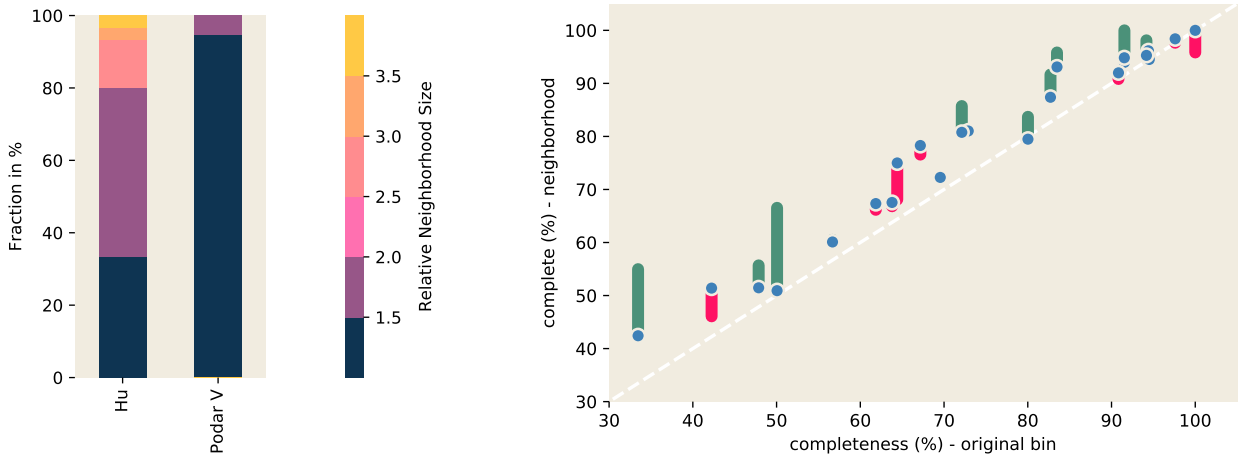

Fig. 3. Query neighborhoods in `HuSB1` metagenome are large and contain additional marker genes. (a) Left Panel: Neighborhood sizes are larger in `HuSB1` than in `podarV`. Here we query `podarV` and `HuSB1` using each of 64 and 23 genomes fully present in the respective datasets and measure the relative size of its neighborhood—a size of 1 indicates that no additional sequence is present in the neighborhood, while a size of 2 indicates that the retrieved neighborhood is twice the size of the query genome. (b) Right Panel: Query neighborhoods are estimated to be more complete than the original genome bins. We query `HuSB1` using each of 23 genomes binned from SB1, and assemble the resulting neighborhoods using MEGAHIT and Plass. The yellow points represent completeness estimates of MEGAHIT-assembled neighborhoods, while green and pink bars represent the additional or missing content in the Plass assemblies respectively.

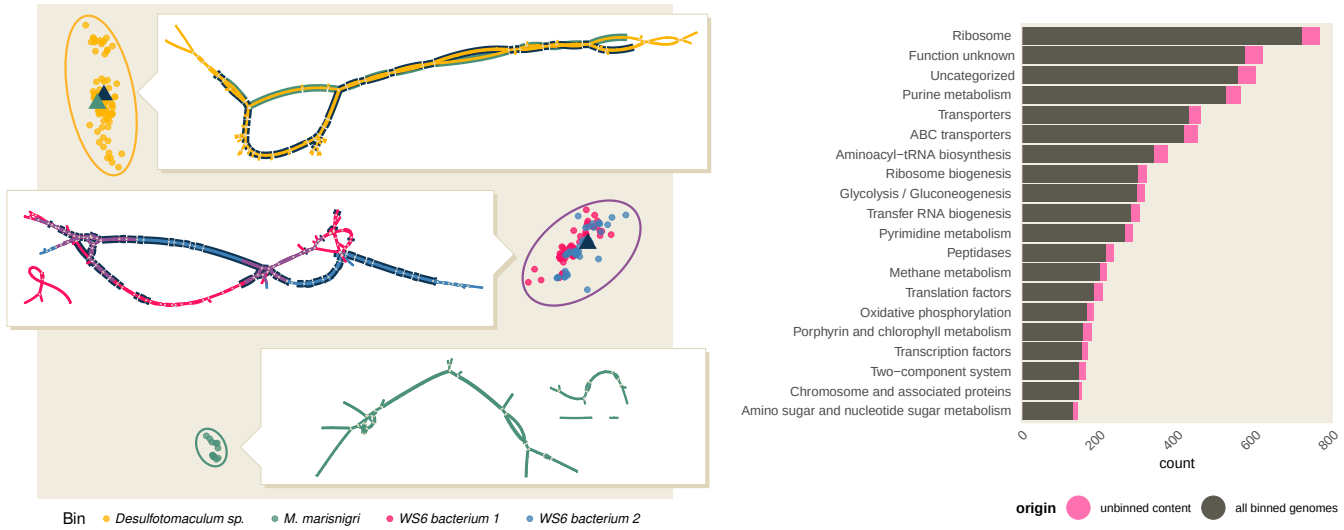

Fig. 4. Query neighborhoods in `HuSB1` contain sequence variants and new genes. (a) Left Panel: gyrA has substantial minor sequence variation in several query neighborhoods. In this multidimensional scaling plot, each point represents a distinct gyrA sequence from the Plass assemblies of four representative query neighborhoods, colored by query binned genome. The triangles represent gyrA sequences originating from the query binned genome, if any are present. The inlays are visualizations of assembly graphs of reads that contain *gyrA* sequence in each neighborhood. Unitigs are colored by their cluster of origin; matches to *gyrA* sequences from the bin are highlighted using color from relevant triangle. (b) Right Panel: Genome neighborhoods re-associate annotated functionality to binned genomes. For each of 23 genome bins originating from `HuSB1`, we find the unbinned content by removing all orthologs found in the binned genomes in (36) and by counting distinct ortholog annotations once. Functional content is distributed throughout pathways present in the binned genomes, and increases functionality associated with binned genomes by approximately 13%.

where bacteria are the dominant nitrogen fixers (32).

Discussion

Efficient graph algorithms provide novel tools for investigating graph neighborhoods. Recent work has shown that incorporating the structure of the assembly graph into the analysis of metagenome data can provide a more complete picture of gene content (14, 15). While this has provided evidence that it is useful to analyze sequence that has small graph distance from a query (is in a "neighborhood"), this approach has not been widely adopted due to the lack of available tools and methods. Naïvely, local expansion around many queries in the assembly graph does not scale to these types of analyses due to the overhead associated with searching in a massive graph. The neighborhood index structure described in this work overcomes this computational obstacle and enables rapid exploration of sequence data that is local to a query.

Because a partition into pieces provides an implicit data reduction (the cDBG edge relationships are subsumed by

piece membership), the query-independent nature of the index allows many queries to be processed quickly without loading the entire graph into memory. Our approach consequently provides a data exploration framework not available in any other automated software packages.

Exploiting the structural sparsity of cDBGs is a crucial component of our algorithms. First, it is necessary to use graph structure to obtain a guarantee that Algorithm 2 finds a small number of pieces since the size of a minimum r-dominating set cannot be approximated better than a factor of $\log n$ in general graphs[†] unless NP $\subseteq$ DTIME($n^{O(\log \log n)}$) (23). Without such a guarantee, we cannot be sure that we are achieving significant data reduction by grouping cDBG vertices into pieces. Being able to do this in linear time also ensures that the indexing and querying can scale to very large data sets. Furthermore, because we utilize a broad structural characterization (bounded expansion) rather than a highly specialized aspect of cDBGs, our methods enable neighborhood-based information retrieval in any domain whose graphs exhibit bounded expansion structure; examples include some infrastructure, social, and communication networks (21, 37, 38).

Neighborhood queries extend genome bins. In both the `podarV` and `HuSB1` metagenomes, neighborhood queries were able to identify additional content likely belonging to query genomes. In the `podarV` mock metagenome, we retrieved a potentially complete genome for an unknown strain based on partial matches to known genomes. In the `HuSB1` metagenome, we increased the estimated completeness of all but one genome bin – in some cases substantially, e.g. in the case of *Microgenomates bacterium 39_7* we added an estimated 16.5% to the genome bin. In both cases we rely solely on the structure of the assembly graph to expand the genome bins. We do not make use of sequence composition, contig abundance, or phylogenetic marker genes in our search. Neighborhood search provides an orthogonal set of information for genome-resolved metagenomics that could be used to improve current binning techniques.

Query neighborhoods from real metagenomes contain substantial strain variation that may block assembly. Previous work suggests that metagenome assembly and binning approaches are fragile to strain variation, which would prevent the characterization of some genomes from metagenomes (9, 10). The extent of this problem is unknown, although the majority of approaches to genome-resolved metagenomics rely on assembly and thus could be affected.

In this work, we find that some of the sequence missing from genome bins can be retrieved using neighborhood queries. For `HuSB1`, some genome bins are missing as much as 66.6% of marker genes from the original bins, with more than half of the 22 bins missing 20% or more; this accords well with evidence from a recent comparison of single-cell genomes and metagenome-assembled genomes (13), in which it was found that metagenome-assembled genomes were often missing 20% to 40% of single-cell genomic sequence. Neighborhood query followed by amino acid assembly recovers additional content for all but two of the genome bins; this is likely an underestimate, since Plass may also be failing to assemble some content.

We note that when we bioinformatically analyze the function of the expanded genome content from neighborhood queries, our results are consistent with the previous metabolic analyses by (32), and extend the set of available genes by 13%. This suggests that current approaches to genome binning are specific, and that the main question is sensitivity, which agrees with the conclusions in the recent analysis of Baltic Sea microbial genomes (13).

Neighborhood queries enable a genome-targeted workflow to recover strain variation. The analysis workflow described above starts with genome bins. The genome bins are used as a query into the metagenome assembly graph, following which we extract reads from the query neighborhood. We assemble these reads with the Plass amino acid assembler, and then analyze the assembly for gene content. We show that the Plass assembly contains strain-level heterogeneity at the amino acid level, and that this heterogeneity is supported by underlying nucleotide diversity. Even with stringent error trimming on the underlying reads, we identify at least thirteen novel gyrA sequences in ten genome neighborhoods.

Of note, this workflow explicitly associates the Plass assembled proteins with specific genome bins, as opposed to a whole-metagenome Plass assembly which recovers protein sequence from the entire metagenome but does not link those proteins to specific genomes. The binning-based workflow connects the increased sensitivity of Plass assembly to the full suite of tools available for genome-resolved metagenome analysis, including both phylogenomic and metabolic analysis. The spacegraphcats workflow does not separate regions of the graph shared in multiple query neighborhoods; existing strain recovery approaches such as DESMAN or MSPminer could be used for this purpose (16, 19).

One future step could be to characterize unbinned genomic content from metagenomes by looking at Plass-assembled marker genes in the metagenome that do not belong to any bin's query neighborhood. This would provide an estimate of the extent of metagenome content remaining unbinned.

Conclusions

The assembly-independent approach described in this work provides an alternative window into metagenome content. We extend previous work showing that assembly-based methods are fragile to strain variation, and provide an alternative workflow that substantially broadens our ability to characterize metagenome content. This first investigation focuses on only one mock and one real data set, but the neighborhood indexing approach should be broadly applicable to all shotgun metagenomes.

In this initial investigation of neighborhood indexing, we have focused on using neighborhood queries with a genome bin. We recognize that this approach is of limited use in regions where no genome bin is available; spacegraphcats is flexible and performant enough to support alternative approaches such as querying with k-mers belonging to genes of interest.

Potential applications of spacegraphcats in metagenomics include developing metrics for genome binning quality, analyzing pangenome neighborhood structure, exploring r-dominating sets for $r > 1$, extending analyses to colored De Bruijn graphs, and investigating *de novo* extraction of genomes based on neighborhood content. We could also apply spacegraphcats to analyze the neighborhood structure of assembly graphs overlayed with physical contact information

[†] That is, graphs about which we make no structural assumptions.

(from e.g. HiC), which could yield new applications in both metagenomics and genomics (39, 40).

More generally, the graph indexing approach developed here may be applicable well beyond metagenomes and sequence analysis. The exploitation of bounded expansion to efficiently compute r-dominating sets on large graphs makes this technique applicable to a broad array of problems.

Materials and Methods

Data. We use two data sets: SRR606249 from `podarV` (41) and SRR1976948 (sample SB1) from `hu` (36). Each data set was first preprocessed to remove low-abundance k-mers as in (42), using `trim-low-abund.py` from khmer v2.1.2 (43) with the parameters `-C 3 -Z 18 -M 20e9 -V -k 31`. We build compact De Bruijn graphs using BCALM v2.2.0 (44). Stringent read trimming at low-abundance k-mers was done with `trim-low-abund.py` from khmer, with the parameters `-C 5 -M 20e9 -k 31`.

Software. The source code for the index construction and search is available at https://github.com/spacegraphcats/spacegraphcats (20). It is implemented in Python 3 under the 3-Clause BSD License.

Snakemake (45) workflows to reproduce all of the analysis are available at https://github.com/dib-lab/2018-paper-spacegraphcats/, and Jupyter Notebooks to recreate all of the figures are in that same repository (46). The notebooks rely on the numpy, matplotlib, pandas, scipy, and Vega-Lite libraries (47–51).

Benchmarking. We measured time and memory usage for Algorithms 1-3 by executing the following targets in the spacegraphcats `conf/Snakefile`: `catlas.csv` for `rdomset`, `contigs.fa.gz.mphf` for `indexPieces`, and `search` for `search`. We report wall time and maximum resident set size, running under Ubuntu 18.04 on an NSF Jetstream virtual machine with 10 cores and 30 GB of RAM (52, 53). To measure maximum resident set size, we used the memusg script (Jaeho Shin, https://gist.github.com/netj/526585).

Graph denoising. For each data set, we built a compact De Bruijn graph (cDBG) for k=31 by computing the set of unitigs with BCALM (54), removing all vertices of degree one with a mean k-mer abundance of 1.1 or less, and then contracting long degree-two paths when possible.

Neighborhood indexing and search. We use spacegraphcats to build an r-dominating set for the denoised cDBG and index it. We then perform neighborhood queries with spacegraphcats, which produces a set of cDBG nodes and reads that contributed to them. The full list of query genomes for the *Proteiniclasticum* query is available in Supp Material A, and the NCBI accessions for the *P. gingivalis*, *T. denticola*, and *B. thetaiotamicron* queries are in the directory `pipeline-base` of the paper repository, files `strain-gingivalis.txt`, `strain-denticola.txt`, and `strain-bacteroides.txt`, respectively.

Search results analysis. Query neighborhood size, Jaccard containment, and Jaccard similarity was estimated using modulo hash signatures with a k-mer size of 31 and a scaled factor of 1000, as implemented in sourmash v2.0a9 (55).

Assembly and genome bin analysis. We assembled reads using MEGAHIT v1.1.3 (28) and Plass v2-c7e35 (33), treating the reads as single-ended. Bin completeness was estimated with CheckM 1.0.11, with the `-reduced_tree` argument (35). Amino acid identity between bins and genomes was calculated using CompareM commit 7cd51276 (https://github.com/dparks1134/CompareM).

Gene targeted analysis. Analysis of specific genes was done with HMMER v3.2.1 (56). Plass amino acid assemblies were queried with HMMER `hmmscan` using the PFAM domains listed in Supp Material 8, using a threshold score of 100 (57). Matching sequences were then extracted from the assemblies for further analysis. To overcome problems associated with comparing non-overlapping sequence fragments, only sequences that overlapped 125 of the most-overlapped 200 residues of the PFAM domain were retained (all sequences shared a minimum overlap of 50 residues with all other sequences). These sequences were aligned with MAFFT v7.407 with the `-auto` argument (58). Pairwise similarities were calculated using HMMER where the final value represented the number of identical amino acids in the alignment divided by the number of overlapping residues between the sequences. Pairwise distances were visualized using a multidimensional scaling calculated in R using the `cmdscale` function. To visualize the assembly graph structure underlying these amino acid assemblies, we use paladin v1.3.1 to map abundance-trimmed reads back to the Plass amino acid assembly, with the `-f` argument set to 125 to set the minimum ORF length accepted (59). We extracted the reads that mapped to the gene of interest, created an assembly graph using BCALM v2.2.0 (54), and visualized the graph using Bandage v0.8.1 (60).

KEGG Analysis. We annotated the Plass assemblies using Kyoto Encyclopedia of Genes GhostKOALA v2.0 (61). To assign KEGG ortholog function, we used methods implemented at https://github.com/edgraham/GhostKoalaParser release 1.1.

ACKNOWLEDGMENTS. This work is funded in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through Grants GBMF4551 to C. Titus Brown, GBMF4553 to Jeffrey Heer, and GBMF4560 to Blair D. Sullivan. This work arose from the Barnraising for Data-Intensive Discovery at Mt. Desert Island Biological Lab in May 2016. We thank Erich Schwarz, Martin Steinegger, Johannes Söding, Mark Blaxter, and members of the Data Intensive Biology lab at UC Davis for discussion and feedback.

1. Pell J, , et al. (2012) Scaling metagenome sequence assembly with probabilistic de bruijn graphs. *PNAS* 109(33):13272–13277.
2. Laczny CC, et al. (2017) Busybee web: metagenomic data analysis by bootstrapped supervised binning and annotation. *Nucleic Acids Research* p. gkx348.
3. Lin H, Liao Y (2016) Accurate binning of metagenomic contigs via automated clustering sequences using information of genomic signatures and marker genes. *Scientific Reports* 6:24175.
4. Parks DH, et al. (2017) Recovery of nearly 8, 000 metagenome-assembled genomes substantially expands the tree of life. *Nature Microbiology* 2(11):1533–1542.
5. Tully BJ, Graham ED, Heidelberg JF (2018) The reconstruction of 2, 631 draft metagenome-assembled genomes from the global oceans. *Scientific Data* 5:170203.
6. Stewart RD, et al. (2018) Assembly of 913 microbial genomes from metagenomic sequencing of the cow rumen. *Nature Communications* 9(1).
7. Delmont TO, et al. (2018) Nitrogen-fixing populations of planctomycetes and proteobacteria are abundant in surface ocean metagenomes. *Nature Microbiology* 3(7):804–813.
8. Hug LA, et al. (2016) A new view of the tree of life. *Nature Microbiology* 1(5).
9. Sczyrba A, et al. (2017) Critical assessment of metagenome interpretation—a benchmark of metagenomics software. *Nature Methods* 14(11):1063–1071.
10. Awad S, Irber L, Brown CT (2017) Evaluating metagenome assembly on a simple defined community with many strain variants. https://www.biorxiv.org/content/early/2017/07/03/155358.
11. Brown CT (2015) Strain recovery from metagenomes. *Nature Biotechnology* 33(10):1041–1043.
12. Brito IL, Alm EJ (2016) Tracking strains in the microbiome: Insights from metagenomics and models. *Frontiers in Microbiology* 7.
13. Alneberg J, et al. (2018) Genomes from uncultivated prokaryotes: a comparison of metagenome-assembled and single-amplified genomes. *Microbiome* 6(1).
14. Barnum TP, et al. (2018) Genome-resolved metagenomics identifies genetic mobility, metabolic interactions, and unexpected diversity in perchlorate-reducing communities. *The ISME Journal* 12(6):1568–1581.
15. Olekhnovich EI, Vasilyev AT, Ulyantsev VI, Kostryukova ES, Tyakht AV (2017) MetaCherchant: analyzing genomic context of antibiotic resistance genes in gut microbiota. *Bioinformatics* 34(3):434–444.
16. Quince C, et al. (2017) DESMAN: a new tool for de novo extraction of strains from metagenomes. *Genome Biology* 18(1).
17. Nayfach S, Rodriguez-Mueller B, Garud N, Pollard KS (2016) An integrated metagenomics pipeline for strain profiling reveals novel patterns of bacterial transmission and biogeography. *Genome Research* 26(11):1612–1625.
18. Garrison E (2018) Ph.D. thesis (Cambridge University). As submitted, awaiting viva (defense) and further revision.
19. Onate FP, et al. (2018) MSPminer: abundance-based reconstitution of microbial pan-genomes from shotgun metagenomic data. *Bioinformatics*.
20. Brown CT, Moritz D, O'Brien MP, Reidl F, Sullivan BD (2018) spacegraphcats, v1.0 (http://dx.doi.org/10.5281/zenodo.1478025).
21. Reidl F (2016) Structural sparseness and complex networks. Aachen, Techn. Hochsch., Diss., 2015.
22. Karp RM (1972) Reducibility among combinatorial problems in *Complexity of computer computations*. (Springer), pp. 85–103.

23. Chlebík M, Chlebíková J (2008) Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation* 206(11):1264–1275.
24. Downey RG, Fellows MR (2012) *Parameterized complexity*. (Springer Science & Business Media).
25. de Mendez PO, , et al. (2012) *Sparsity: graphs, structures, and algorithms*. (Springer Science & Business Media) Vol. 28.
26. Limasset A, Rizk G, Chikhi R, Peterlongo P (2017) Fast and scalable minimal perfect hashing for massive key sets. *CoRR* abs/1702.03154.
27. Shakya M, et al. (2013) Comparative metagenomic and rrna microbial diversity characterization using archaeal and bacterial synthetic communities. *Environmental Microbiology* 15(6):1882–1899.
28. Li D, et al. (2016) MEGAHIT v1.0: A fast and scalable metagenome assembler driven by a dvanced methodologies and community practices. *Methods* 102:3–11.
29. Seah BKB, Gruber-Vodicka HR (2015) gbtools: Interactive visualization of metagenome bins in r. *Frontiers in Microbiology* 6.
30. Nurk S, Meleshko D, Korobeynikov A, Pevzner PA (2017) metaspades: a new versatile metagenomic assembler. *Genome Research* 27(5):824–834.
31. Sharon I, et al. (2015) Accurate, multi-kb reads resolve complex populations and detect rare microorganisms. *Genome Research* 25(4):534–543.
32. Hu P, et al. (2016) Genome-resolved metagenomic analysis reveals roles for candidate phyla and other microbial community members in biogeochemical transformations in oil reservoirs. *mBio* 7(1).
33. Steinegger M, Mirdita M, Soding J (2018) Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold.
34. Yang Y, Yooseph S (2013) SPA: a short peptide assembler for metagenomic data. *Nucleic Acids Research* 41(8):e91–e91.
35. Parks DH, Imelfort M, Skennerton CT, Hugenholtz P, Tyson GW (2015) CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome Research* 25(7):1043–1055.
36. Hu P, et al. (2016) Genome-resolved metagenomic analysis reveals roles for candidate phyla and other microbial community members in biogeochemical transformations in oil reservoirs. *MBio* 7(1):e01669–15.
37. Demaine ED, et al. (2014) Structural sparsity of complex networks: Random graph models and linear algorithms. *CoRR* abs/1406.2587.
38. Nadara W, Pilipczuk M, Rabinovich R, Reidl F, Siebertz S (2018) Empirical evaluation of approximation algorithms for generalized graph coloring and uniform quasi-wideness in *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, LIPIcs, ed. D'Angelo G. (Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik), Vol. 103, pp. 14:1–14:16.
39. Marbouty M, et al. (2014) Metagenomic chromosome conformation capture (meta3c) unveils the div ersity of chromosome organization in microorganisms. *eLife* 3.
40. Beitel CW, et al. (2014) Strain- and plasmid-level deconvolution of a synthetic metagenome by sequencing proximity ligation products. *PeerJ* 2:e415.
41. Shakya M, , et al. (2013) Comparative metagenomic and rrna microbial diversity characterization using archaeal and bacterial synthetic communities. *Environ. microbiol.* 15(6):1882–1899.
42. Zhang Q, Awad S, Brown CT (2015) Crossing the streams: a framework for streaming analysis of short DNA sequencing reads. https://doi.org/10.7287/peerj.preprints.890v1.
43. Standage D, et al. (2017) khmer release v2.1: software for biological sequence analysis. *The Journal of Open Source Software* 2(15):272.
44. Chikhi R, Limasset A, Medvedev P (2016) Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics* 32(12):i201–i208.
45. Koster J, Rahmann S (2012) Snakemake–a scalable bioinformatics workflow engine. *Bioinformatics* 28(19):2520–2522.
46. Kluyver T, et al. (2016) Jupyter notebooks-a publishing format for reproducible computational workflows. in *ELPUB*. pp. 87–90.
47. van der Walt S, Colbert SC, Varoquaux G (2011) The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering* 13(2):22–30.
48. Hunter JD (2007) Matplotlib: A 2d graphics environment. *Computing in Science & Engineering* 9(3):90–95.
49. McKinney W (2011) pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing* pp. 1–9.
50. Jones E, Oliphant T, Peterson P, , et al. (2001–) SciPy: Open source scientific tools for Python. [Online; accessed <today>].
51. Satyanarayan A, Moritz D, Wongsuphasawat K, Heer J (2017) Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics* 23(1):341–350.
52. Stewart CA, et al. (2015) Jetstream in *Proceedings of the 2015 XSEDE Conference on Scientific Advancements Enabled by Enhanced Cyberinfrastructure - XSEDE '15*. (ACM Press).
53. Towns J, et al. (2014) XSEDE: Accelerating scientific discovery. *Computing in Science & Engineering* 16(5):62–74.
54. Chikhi R, Limasset A, Medvedev P (2016) Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics* 32(12):i201–i208.
55. Brown CT, Irber L, Cohen L (2016) dib-lab/sourmash: v1.0 (https://doi.org/10.5281/zenodo.153989).
56. Eddy SR, Team HD (2018) Hmmer v3.2.1. http://hmmer.org.
57. Finn RD, et al. (2015) The pfam protein families database: towards a more sustainable future. *Nucleic Acids Research* 44(D1):D279–D285.
58. Katoh K, Standley DM (2013) MAFFT multiple sequence alignment software version 7: Improvements in performance and usability. *Molecular Biology and Evolution* 30(4):772–780.
59. Westbrook A, et al. (2017) PALADIN: protein alignment for functional profiling whole metagenome shotgun data. *Bioinformatics* 33(10):1473–1478.
60. Wick RR, Schultz MB, Zobel J, Holt KE (2015) Bandage: interactive visualization ofde novogenome assemblies: Fig. 1. *Bioinformatics* 31(20):3350–3352.
61. Kanehisa M, Sato Y, Morishima K (2016) BlastKOALA and GhostKOALA: KEGG tools for functional characterization of genome and metagenome sequences. *Journal of Molecular Biology* 428(4):726–731.

A. Appendix

Approximation guarantee. Let us introduce some notation for the analysis of Algorithm 1. We first partition the vertices of D according to whether they were added in line 10 (denoted by D_1) or in line 15 (denoted by D_2). Let $v_1, \ldots, v_n$ be the vertex- order in which they are iterated over in the loop starting at line 6. We will use the notation D_1^i, D_2^i, d^i, and c^i to represent the states of the respective sets and data structures during the ith iteration of said loop. Let $\tau := \texttt{domThreshold}(r)$ be the chosen threshold (we discuss a good value for τ on cDBGs below).

Lemma 1. *After the for-loop at line 8 has finished,*

$$d^i[v_i] = \begin{cases} dist_G(v, D^i) & \text{if } dist_G(v, D^i) \leq r, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

Proof. The statement trivially holds while $D^i = \emptyset$, so assume otherwise. Let $u_h \in D^i$ be the vertex closest to v_i and let $h < i$ be the iteration in which u_h was added to D (either in line 10 or line 15 of that iteration).

If $d := \text{dist}_G(v_i, u_h) > r$, then $d^i[v_i]$ has not been changed yet and is still set to ∞. Otherwise, consider the three possible scenarios promised by the distance-property of dtf-augmentations:

Case 1: $v_i u_h \in \vec{G}_d$. Then $\omega(v_i u_h) = d$ and in iteration h the value of $d^h[v_i]$ is set to the correct value d in the loop at line 8. By assumption this distance remains minimal until iteration i and hence $d^i[v_i] = d^h[v_i] = d$.

Case 2: $u_h v_i \in \vec{G}_d$. Then $\omega(v_i u_h) = d$ and in iteration i the value of $d^i[v_i]$ is set to the correct value d in the loop at line 8.

Case 3: $xu_h, xv_i \in \vec{G}_d$ with $\omega(xu_h) + \omega(xv_i) = d$. During iteration h the value of $d^h[x]$ is set to $\omega(xu_h)$ in the loop at line 8 and subsequently retrieved in iteration i when $d^i[v_i]$ is set to

$$d^i[x] + \omega(xu_h) = \omega(xu_h) + \omega(xv_i) = d.$$

We conclude that after the execution of the loop at line 8. $d^i[v_i]$ is set to ∞ if v_i is not dominated by D_i and is otherwise set to $\text{dist}_G(v_i, D^i)$, as claimed. □

As an immediate consequence, we see the conditional statement at the end of the loop at line 8 accurately determines whether v_i is dominated by D_i or not. Accordingly, lines 15 of the loop are only executed if v_i is *not* dominated by D^i. Another consequence is that all vertices in D_1 have large distance to each other:

Corollary 1. *The set D_1 is $(r+1)$-scattered in G.*

We need one more important property of the algorithm in order to derive the approximation factor.

Lemma 2. *For every $w \in G$ it holds that $|D_1 \cap N_r^-(w)| \leq \tau + 1$.*

Proof. Assume towards a contradiction that $\tau + 2$ such vertices $v_{i_1}, \ldots, v_{i_\tau+2}$, $i_1 < i_2 < \ldots < i_{\tau+2}$ exist in $D_1 \cap N_r^-(w)$. Since every such vertex v_i, $i \in \{i_1, \ldots i_{\tau+2}\}$, was added to D in part (2), part (3) of the algorithm was executed during iteration i as well. Thus $c[w]$ was increased in each iteration i and during iteration $i_{\tau+1}$ we have that $c[w] \geq \tau + 1$ after the increment of $c[w]$. Therefore part (4) must have been executed for w, including w into D. Hence $w \in D^s$ for $s > i_{\tau+1}$ and in particular $w \in D^{i_{\tau+2}}$. But then $v_{i_{\tau+2}}$ was dominated by w at the beginning of iteration $i_{\tau+2}$ since we assumed that $\omega(rv_{i_{\tau+2}}) \leq r$, thus $v_{i_{\tau+2}}$ would not have been included in D at step (2). This contradicts our assumption of $v_{i_{\tau+2}} \in D_1$ so the claim must hold. □

Lemma 3. *There exists a subset $A \subseteq D_1$ such that A is $(2r+1)$-scattered in G and*

$$|A| \geq \frac{|D|}{2(\tau+2)\Delta^-(\vec{G}_{2r}))\Delta^-(\vec{G}_r)}.$$

Proof. We construct an auxiliary graph H with vertices D_1 by adding arcs $v_i v_j$ for $v_i, v_j \in D_1$ with $i < j$ whenever $\text{dist}_G(v_i, v_j) \leq 2r$. Let $\vec{G}_{2r}$ be a $2r$th dtf-augmentation of G and let us create a digraph $\vec{H}$ by orienting every edge $uv \in H$ as follows:

1. If of $uv, vu \in \vec{G}_{2r}$, then orient uv in $\vec{H}$ according to the corresponding arc in $\vec{G}_{2r}$ (if both arcs exists choose an arbitrary orientation),
2. otherwise there exists $w \in N_{2r}^-(u) \cap N_{2r}^-(v)$ with $\omega_{2r}(u) + \omega_{2r}(v) = \text{dist}_G(u, v) \leq 2r$. Orient the edge uv towards that vertex $x \in \{u, v\}$ for which $\omega_{2r}(x)$ is larger.

We now argue that $\Delta^-(\vec{H})$ is small. Consider any vertex $v \in \vec{H}$. Every in-arc $uv \in \vec{H}$ either is of type 1, of which we have at most $\Delta^-(\vec{G}_{2r})$, or of type 2. Consider a group of in- arcs $u_i v$, $1 \leq i \leq \ell$ of type 2 that are all present because of a common vertex w. Since $w \in N_{2r}^-(u)$, we have at most $\Delta^-(\vec{G}_{2r})$ such groups. By construction, $\omega_{2r}(wu_i) \leq \omega_{2r}(wv)$ and since both weights sum to less than $2r$, this means that $\omega_{2r}(wu_i) \leq r$. Lemma 2 now tells us that $\ell \leq \tau + 1$. Therefore v has at most $(\tau+1)\Delta^-(\vec{G}_{2r})$ in-arcs of type 2 and we conclude that

$$\Delta^-(\vec{H}) \leq \Delta^-(\vec{G}_{2r}) + (\tau+1)\Delta^-(\vec{G}_{2r}) = (\tau+2)\Delta^-(\vec{G}_{2r}).$$

This finally implies that H is $2(\tau+2)\Delta^-(\vec{G}_{2r})$-degenerate and therefore contains an independent set $A \subseteq V(H)$ of size at least $|A| \geq |H|/(2(\tau+2)\Delta^-(\vec{G}_{2r}))$. Taken together with the fact that $|H| = |D_1| \geq |D|/\Delta^-(\vec{G}_r)$ (every vertex added to D_1 will cause at most $\Delta^-(\vec{G}_r)$ many vertices to be added to D_2 in the loop at line 11 and $D = D_1 \cup D_2$), we find that

$$|A| \geq \frac{|D|}{2(\tau+2)\Delta^-(\vec{G}_{2r}))\Delta^-(\vec{G}_r)}$$

By construction of H we conclude that A is $(2r+1)$-scattered in G of the claimed sized. □

Since a $(2r+1)$-scattered set provides a lower bound for an r-dominating set, we conclude that Algorithm 1 computes a $2(\tau+2)\Delta^-(\vec{G}_{2r})\Delta^-(\vec{G}_r)$-approximation of an optimal r-dominating set, which is a constant-factor approximation in graphs of bounded expansion.

In practice one could, depending on the value of $\Delta^-(\vec{G}_r)$ and $\Delta^-(\vec{G}_{2r})$, compute the optimal value for τ. However, this would necessitate the computation of $2r$ augmentation, the expensive step we want to avoid. Alternatively, we can choose a 'good enough' value for τ that still guarantees a constant-factor approximation while being easy to determine in practice. In the context of cDBGs, we found that $\tau := (2r)^2$ yields reliably good results.

Computational Runtimes. See "Benchmarking" in Materials and Methods for benchmarking methods.

The `podarV` data set was retrieved from the NCBI SRA using accession SRR606249. The full build and indexing of the 103 million error-trimmed reads (10.3 Gbp in total) took approximately 23 minutes and required 12.8 GB of RAM. Loading the indices for search required 4.3 GB of RAM and a search with a 3 Mbp genome took approximately 32 seconds.

The `HuSB1` data set was retrieved from the NCBI SRA using accession SRR1976948. The full build and indexing of the 34 million error-trimmed reads (8.5 Gbp in total) required approximately 217 minutes and required 24.4 GB of RAM. Loading the indices for search required 18 GB of RAM and a search with a 3 Mbp genome took approximately 80 seconds.

For data set complexity (number of k-mers, number of cDBG nodes) please see Table 1.

Query genome accession numbers for *Proteiniclasticum* search. See Table 3.

Amino Acid Identity results for *Proteiniclasticum*. See Table 4.

Genome bin completeness improvements for `HuSB1`. See Table 5.

K-mer inclusion of reads by MEGAHIT assemblies. See Table 7.

gyrA alignment. See Figure 5

gyrA by neighborhood. See Table 6.

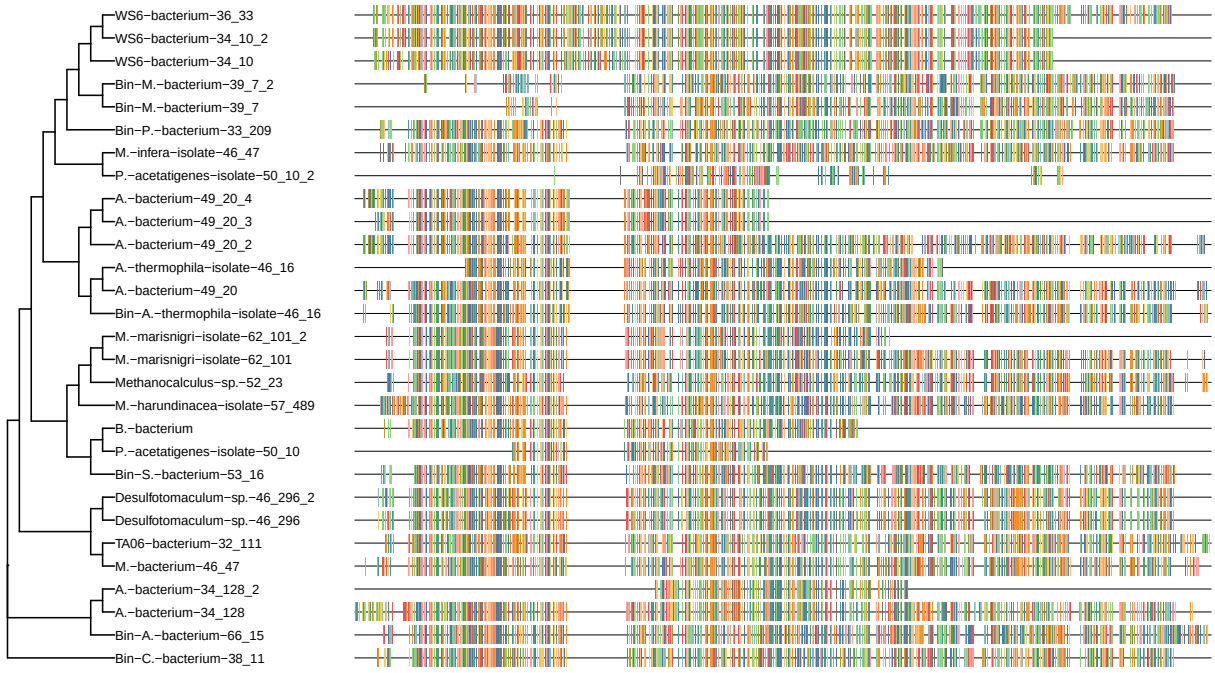

Fig. 5. A multiple sequence alignment and neighbor joining tree of representative gyrA amino acid fragments assembled by Plass from the genome neighborhoods in `HuSB1`.

Other genes. See bin and neighborhood content results for *alaS* in Table 9, *gyrB* in Table 10, *recA* in Table 11, rpb2d6 in Table 12, *rplB* in Table 13, and *rpsC* in Table 14.

Name	NCBI accession
P. ruminis CGMCC	GCA_900099635.1
P. ruminis DSM	GCA_000701905.1
P. ruminis ML2	GCA_900115135.1

Table 3. Accession numbers for genomes used in *Proteiniclasticum* neighborhood query.

Genome A	Genome B	Orthologous Genes	Mean AAI
P. ruminis ML2	*P. ruminis shakya*	2546	95.74
P. ruminis DSM	*P. ruminis shakya*	2391	93.47

Table 4. CompareM results for *Proteiniclasticum* genomes. *P. ruminis shakya* is the result of assembling the reads extracted from `podarV` with the neighborhood search.

Bin name	Bin completeness	MEGAHIT (Δ)	Plass (Δ)
P. bacterium 34_609	33.5%	42.4% (9.0)	55.0% (21.5)
WS6 bacterium 36_33	42.2%	51.4% (9.2)	46.1% (3.8)
P. bacterium 33_209	47.9%	51.5% (3.6)	55.7% (7.8)
M. bacterium 39_7	50.1%	50.9% (0.9)	66.5% (16.5)
P. acetatigenes isolate 50_10	56.7%	60.1% (3.4)	60.5% (3.8)
WS6 bacterium 34_10	61.9%	67.3% (5.5)	66.2% (4.3)
M. infera isolate 46_47	63.8%	67.5% (3.8)	66.8% (3.0)
A. bacterium 34_128	64.4%	75.0% (10.6)	68.1% (3.7)
A. thermophila isolate 46_16	67.2%	78.3% (11.1)	76.6% (9.4)
A. bacterium 49_20	69.5%	72.3% (2.7)	72.4% (2.9)
M. marisnigri isolate 62_101	72.1%	80.8% (8.6)	85.7% (13.6)
M. bacterium 46_47	72.9%	81.0% (8.1)	81.0% (8.1)
B. bacterium	80.0%	79.5% (-0.5)	83.7% (3.7)
Methanocalculus sp. 52_23	82.7%	87.4% (4.7)	91.7% (9.0)
Desulfotomaculum sp. 46_80	83.5%	93.1% (9.6)	95.8% (12.3)
S. bacterium 57_84	90.8%	92.0% (1.2)	90.8% (0.0)
S. bacterium 53_16	91.5%	94.8% (3.3)	94.8% (3.3)
Desulfotomaculum sp. 46_296	91.5%	94.0% (2.4)	100.0% (8.5)
A. bacterium 66_15	94.2%	95.3% (1.1)	98.1% (4.0)
C. bacterium 38_11	94.4%	96.2% (1.7)	96.2% (1.8)
TA06 bacterium 32_111	94.5%	94.5% (0.0)	96.4% (1.8)
Methanobacterium sp. 42_16	97.6%	98.4% (0.8)	97.7% (0.1)
M. harundinacea isolate 57_489	100.0%	100.0% (0.0)	95.8% (-4.2)

Table 5. Bin and neighborhood completeness, as estimated by CheckM. "Bin completeness" is the result of running CheckM on the genome sequence from GenBank; MEGAHIT is the result of running CheckM on the MEGAHIT nucleotide assembly of the neighborhood reads; Plass is the result of running CheckM on the Plass amino acid assembly of the neighborhood reads. Δ is the difference between the column and the bin completeness.

Species	gyrA (bin)	gyrA (Plass)
Methanobacterium sp. 42_16	0	0
P. bacterium 34_609	0	0
Desulfotomaculum sp. 46_80	0	0
S. bacterium 57_84	0	0
B. bacterium	0	1
P. acetatigenes isolate 50_10	0	2
WS6 bacterium 34_10	0	2
M. marisnigri isolate 62_101	0	2
C. bacterium 38_11	1	1
M. infera isolate 46_47	1	1
S. bacterium 53_16	1	1
M. bacterium 46_47	1	1
TA06 bacterium 32_111	1	1
P. bacterium 33_209	1	1
A. bacterium 66_15	1	1
Methanocalculus sp. 52_23	1	2
WS6 bacterium 36_33	1	2
A. bacterium 34_128	1	2
A. thermophila isolate 46_16	1	2
M. harundinacea isolate 57_489	1	2
M. bacterium 39_7	2	0
Desulfotomaculum sp. 46_296	2	2
A. bacterium 49_20	2	3

Table 6. Bin and neighborhood gyrA protein content. gyrA count for each bin is the number of gyrA amino acid sequences part of the original bin. gyrA count by Plass is the minimum number of gyrA amino acid sequences supported by at least one position with at least 10 copies of each variant, e.g. "3" indicates that there is at least one position in the multiple sequence alignment of gyrA sequences for that neighborhood that has 3 distinct variants in 10 distinct sequences.

Species	MEGAHIT assembly containment
M. harundinacea isolate 57_489	4.2%
Desulfotomaculum sp. 46_296	12.7%
M. marisnigri isolate 62_101	13.7%
S. bacterium 57_84	19.4%
P. bacterium 34_609	19.6%
A. bacterium 66_15	20.5%
Desulfotomaculum sp. 46_80	24.0%
P. bacterium 33_209	26.5%
S. bacterium 53_16	30.9%
A. bacterium 49_20	31.8%
Methanocalculus sp. 52_23	33.4%
P. acetatigenes isolate 50_10	36.5%
M. bacterium 46_47	36.6%
A. bacterium 34_128	36.8%
M. infera isolate 46_47	37.9%
Methanobacterium sp. 42_16	38.1%
A. thermophila isolate 46_16	38.6%
TA06 bacterium 32_111	44.1%
C. bacterium 38_11	44.4%
WS6 bacterium 34_10	53.3%
WS6 bacterium 36_33	53.8%
B. bacterium	54.1%
M. bacterium 39_7	55.7%

Table 7. Containment of neighborhood k-mer content in MEGAHIT nucleotide assemblies.

Name	PFAM accession
recA	PF00154
rplB	PF00181
rpsC	PF00189
gyrB	PF00204
gyrA	PF00521
rpb2d6	PF00562
alaS	PF01411

Table 8. Protein names and PFAM accessions for targeted analyses.

Species	alaS (bin)	alaS (Plass)
P. acetatigenes isolate 50_10	0	0
A. bacterium 49_20	0	0
P. bacterium 34_609	0	0
B. bacterium	0	0
S. bacterium 53_16	0	0
A. bacterium 34_128	0	0
M. infera isolate 46_47	0	2
M. marisnigri isolate 62_101	0	2
M. bacterium 39_7	1	0
Methanobacterium sp. 42_16	1	1
C. bacterium 38_11	1	1
S. bacterium 57_84	1	1
TA06 bacterium 32_111	1	1
P. bacterium 33_209	1	1
A. bacterium 66_15	1	1
M. harundinacea isolate 57_489	1	1
Methanocalculus sp. 52_23	1	2
WS6 bacterium 36_33	1	2
Desulfotomaculum sp. 46_80	1	2
M. bacterium 46_47	1	2
Desulfotomaculum sp. 46_296	1	2
A. thermophila isolate 46_16	2	1
WS6 bacterium 34_10	2	2

Table 9. Bin and neighborhood alaS protein content.

Species	gyrB (bin)	gyrB (Plass)
M. bacterium 39_7	0	0
P. acetatigenes isolate 50_10	0	0
Methanobacterium sp. 42_16	0	0
WS6 bacterium 36_33	0	0
P. bacterium 34_609	0	0
Desulfotomaculum sp. 46_80	0	0
S. bacterium 57_84	0	0
S. bacterium 53_16	0	0
A. thermophila isolate 46_16	0	2
P. bacterium 33_209	1	0
C. bacterium 38_11	1	1
M. infera isolate 46_47	1	1
M. bacterium 46_47	1	1
TA06 bacterium 32_111	1	1
A. bacterium 66_15	1	1
M. harundinacea isolate 57_489	1	1
WS6 bacterium 34_10	1	2
Methanocalculus sp. 52_23	1	2
M. marisnigri isolate 62_101	1	2
A. bacterium 34_128	1	2
A. bacterium 49_20	2	2
B. bacterium	2	2
Desulfotomaculum sp. 46_296	2	2

Table 10. Bin and neighborhood gyrB protein content.

Species	recA (bin)	recA (Plass)
M. bacterium 39_7	0	0
WS6 bacterium 34_10	0	0
Methanocalculus sp. 52_23	0	0
A. bacterium 49_20	0	0
WS6 bacterium 36_33	0	0
P. bacterium 34_609	0	0
M. marisnigri isolate 62_101	0	0
S. bacterium 53_16	0	0
M. bacterium 46_47	0	0
A. thermophila isolate 46_16	0	1
B. bacterium	1	0
P. acetatigenes isolate 50_10	1	1
Methanobacterium sp. 42_16	1	1
C. bacterium 38_11	1	1
M. infera isolate 46_47	1	1
S. bacterium 57_84	1	1
A. bacterium 34_128	1	1
TA06 bacterium 32_111	1	1
P. bacterium 33_209	1	1
A. bacterium 66_15	1	1
M. harundinacea isolate 57_489	1	1
Desulfotomaculum sp. 46_80	1	2
Desulfotomaculum sp. 46_296	1	2

Table 11. Bin and neighborhood recA protein content.

Species	rpb2d6 (bin)	rpb2d6 (Plass)
P. acetatigenes isolate 50_10	0	0
P. bacterium 34_609	0	0
S. bacterium 57_84	0	1
M. bacterium 46_47	0	1
C. bacterium 38_11	1	0
A. bacterium 49_20	1	0
M. bacterium 39_7	1	1
Methanobacterium sp. 42_16	1	1
Methanocalculus sp. 52_23	1	1
M. infera isolate 46_47	1	1
B. bacterium	1	1
S. bacterium 53_16	1	1
A. bacterium 34_128	1	1
TA06 bacterium 32_111	1	1
A. bacterium 66_15	1	1
A. thermophila isolate 46_16	1	1
WS6 bacterium 36_33	1	2
Desulfotomaculum sp. 46_80	1	2
M. marisnigri isolate 62_101	1	2
Desulfotomaculum sp. 46_296	1	2
P. bacterium 33_209	1	2
M. harundinacea isolate 57_489	1	2
WS6 bacterium 34_10	2	2

Table 12. Bin and neighborhood rpb2d6 protein content.

Species	rplB (bin)	rplB (Plass)
M. bacterium 39_7	0	0
Methanobacterium sp. 42_16	0	0
Methanocalculus sp. 52_23	0	0
WS6 bacterium 36_33	0	0
M. marisnigri isolate 62_101	0	0
M. harundinacea isolate 57_489	0	0
P. acetatigenes isolate 50_10	1	1
C. bacterium 38_11	1	1
A. bacterium 49_20	1	1
M. infera isolate 46_47	1	1
P. bacterium 34_609	1	1
Desulfotomaculum sp. 46_80	1	1
S. bacterium 57_84	1	1
B. bacterium	1	1
S. bacterium 53_16	1	1
A. bacterium 34_128	1	1
M. bacterium 46_47	1	1
Desulfotomaculum sp. 46_296	1	1
TA06 bacterium 32_111	1	1
P. bacterium 33_209	1	1
A. bacterium 66_15	1	1
A. thermophila isolate 46_16	1	1
WS6 bacterium 34_10	1	2

Table 13. Bin and neighborhood rplB protein content.

Species	rpsC (bin)	rpsC (Plass)
M. bacterium 39_7	0	0
P. acetatigenes isolate 50_10	0	0
WS6 bacterium 34_10	0	0
Methanobacterium sp. 42_16	0	0
Methanocalculus sp. 52_23	0	0
WS6 bacterium 36_33	0	0
M. marisnigri isolate 62_101	0	0
B. bacterium	0	0
P. bacterium 33_209	0	0
M. harundinacea isolate 57_489	0	0
M. infera isolate 46_47	0	1
C. bacterium 38_11	1	1
A. bacterium 49_20	1	1
P. bacterium 34_609	1	1
S. bacterium 57_84	1	1
S. bacterium 53_16	1	1
A. bacterium 34_128	1	1
M. bacterium 46_47	1	1
TA06 bacterium 32_111	1	1
A. bacterium 66_15	1	1
A. thermophila isolate 46_16	1	1
Desulfotomaculum sp. 46_80	1	2
Desulfotomaculum sp. 46_296	1	2

Table 14. Bin and neighborhood rpsC protein content.