Sparse dominating sets and balanced neighborhood partitioning

Yosuke Mizutani∗ Annie Staker∗ Blair D. Sullivan∗

December 22, 2021

Abstract
Recent work in metagenomics constructs a partition of the assembly graph using an $r$-dominating set to enable scalable data representation and rapid approximate queries. In this paper, we consider two problems that arise in this setting: selection of a dominating set that minimizes uncertainty in partitioning, and reducing the amount of variation in piece sizes to improve scalability.

First, we introduce a notion of sparse dominating sets which minimize the number of vertices with multiple closest dominators as measured using a new congestion parameter. Although identifying the least congested dominating set is NP-hard, we present an algorithm that finds one with approximately minimum congestion.

In the second setting, we consider the problem of balanced neighborhood partitioning: given an $r$-dominating set, find the partition which assigns each vertex to one of its closest dominators and achieves the “most balanced” piece sizes. We consider the variant which minimizes the variance of piece sizes, and show that it is NP-hard iff $r$ is greater than 1. We design and analyze several algorithms, including a polynomial-time approach which is exact when $r = 1$ (and heuristic otherwise).

We complement our theoretical results with extensive computational experiments on a corpus of real-world networks showing that sparse dominating sets lead to more balanced neighborhood partitionings. Further, on the metagenome HuSB1, our approach maintains high neighborhood query containment and similarity while improving piece size variance.

1 Introduction
Microbial communities play a critical role in many aspects of human health (e.g. gut microbiomes) and ecosystems (e.g. marine ecology), and understanding their composition and function has been increasingly important in biological and medical research. Much of the work on these communities focuses on analyzing the genomic material (DNA and RNA) of the constituent microorganisms, a research area called metagenomics. A primary challenge in the field is reconstructing individual genomes from the mixture of short fragments created by shotgun sequencing.

One practical approach that has gathered significant recent attention utilizes a metagenome assembly graph to guide analyses. Commonly, this is done with a (compact) De Bruijn graph, or (c)DBG, where vertices correspond to DNA subsequences called $k$-mers and edges indicate potential compatibility in an assembly (almost complete overlap). Since the graphs corresponding to real-world metagenomic datasets may have tens of millions of vertices, scalable methods for analysis are imperative. Recent work of Brown et al. [2], implemented in spacegraphcats [4], leveraged the sparsity of these graphs to enable efficient indexing and querying using partial information about suspected constituent microbes. Their approach relies on finding an $r$-dominating set (using Dvorak’s approximation algorithm for sparse graphs [4]), then partitioning the assembly graph into pieces by assigning each vertex to one of its closest dominators. The process is repeated on the piece graph to form a hierarchy of dominating sets to enable effective navigation and categorization of the data. Initial experiments demonstrated the approach can improve the completeness of partial genomes for microbes present in a community and also reveal significant strain variation in real-world microbiomes [2].

Despite these promising results, several key challenges remain. Here, we focus on those related to the algorithm for partitioning the metagenome assembly graph into pieces around the dominators. In particular, since the current algorithm finds an approximately minimum-size dominating set, it may choose several dominators that are very close together in the (c)DBG, leading to uncertainty in how a region should be “carved up” into pieces. Further, the sizes of the pieces may be imbalanced, counteracting any advantage gained by using the hierarchy to prune away irrelevant sequences.

In this work, we tackle these challenges by (a) introducing a notion of sparse dominating sets which rewards “scattering” dominators, with the aim of generating

∗University of Utah, yos@cs.utah.edu, annie.staker@utah.edu, sullivan@cs.utah.edu

1https://github.com/spacegraphcats/spacegraphcats
pieces which are biologically more meaningful and inherently more stable (as there will be less tie-breaking during partitioning), and (b) considering algorithms for balanced neighborhood partitioning: assigning vertices to dominators which minimize variation in the resulting piece sizes.

While we defer formal definitions to Section 2, our approach to avoid needing to choose between multiple closest dominators is based on minimizing congestion. We show that Minimum Congestion r-Dominating Set is NP-hard and present an $O(\sqrt{\Delta} \cdot n)$-approximation algorithm running in $O(\Delta^{2r} n \log n)$ time. We compare this with the $O(r \log \Delta)$ standard approximation algorithm for $r$-Dominating Set, which asks for the smallest $r$-dominating set. Unlike minimum dominating sets, we do not require sparse dominating sets to be small in size, and we discuss trade-offs between the solution size and congestion.

Once we have an $r$-dominating set, the problem becomes one of partitioning the vertices into pieces so that (a) each vertex is assigned to a piece containing one of its closest dominators, and (b) the pieces are as equal in size as possible. For the latter condition, we choose to minimize the variance of the piece size distribution in Balanced Neighborhood Partitioning. We show this is polynomial-time solvable when $r = 1$ and NP-hard when $r \geq 2$, even when there are only two dominators (proofs in Appendix A). We show the problem is fixed parameter tractable (FPT) in graphs of bounded degree when parameterized by the number of vertices equidistant from multiple dominators ($k$), giving an $O(\Delta^{2r} n^4)$ FPT algorithm Prt-Branch. Further, when $r = 1$, we give an exact $O(n^4)$ algorithm Prt-Layer in general graphs using flow-based techniques; when $r \geq 2$, this yields a heuristic. Finally, we compare with a linear-time greedy heuristic, Prt-Weight. These algorithms are described in Section 2.

We implemented all of the above algorithms using C++ in an open-source repository and tested their performance on a large corpus of real-world networks, including a variety of metagenome graphs. Experimental results demonstrate that the choice of dominating set can significantly impact the runtime and solution quality of balanced neighborhood partitioning algorithms, with sparse sets outperforming their smaller but more congested analogs. Finally, we present preliminary results indicating that low-congestion dominating sets do not significantly degrade the fidelity of queries using partial genome bins on HuSB1, one of the metagenome datasets analyzed in 2, a critical requirement for their downstream adoption.

## 2 Sparse Dominating Sets

In this section, we formulate the problem of finding sparse dominating sets, establish hardness, and describe several heuristic algorithms along with an integer (linear) programming (ILP) formulation. Graph theory notation is detailed in Appendix A.

To define a sparse dominating set, we first introduce the notion of congestion which measures how frequently a given set of vertices overlaps with the $r$-neighborhoods in a graph.

**Definition 2.1.** Given a graph $G = (V, E)$, vertex set $S \subseteq V$, and radius $r \in \mathbb{N}$, the $r$-congestion of $S$ at a vertex $v \in V$, denoted $\text{cong}_r(S, v)$, is $|N^r(v) \cap S|$. The **average $r$-congestion** of $S$ in $G$ is then $\text{cong}_r(S) = \frac{1}{|V|} \sum_{v \in V} \text{cong}_r(S, v)$.

We observe that the average congestion of a given set $S$ can also be computed by obtaining the neighborhood size for each vertex in $S$.

**Lemma 2.1.** Average congestion can be computed as $\text{cong}_r(S) = \frac{1}{|V|} \sum_{u \in S} |N^r[u]|$.

We say an $r$-dominating set is sparse when it achieves low average $r$-congestion; that is, a typical vertex has few dominators in its $r$-neighborhood. This naturally leads to the following minimization problem.

**Minimum Congestion $r$-Dominating Set (MCDS)**

**Input:** A graph $G = (V, E)$ and radius $r \in \mathbb{N}$.

**Problem:** Find an $r$-dominating set $D \subseteq V$ such that $\text{cong}_r(D)$ is minimized.

We remark that this is distinct from the class of problems studied in [7, 5] which put uniform local constraints on each vertex (e.g., that they are dominated at least $\lambda$ and at most $\mu$ times).

We write $\text{mac}^r(G)$ ($\text{mac}(G)$ when $r = 1$) for the minimum average congestion attainable by any $r$-dominating set on $G$. By Lemma 2.1, $|V| \cdot \text{cong}_r(D)$ equals the weighted sum over $D$ of $\omega(v) = |N^r(v)|$. Thus, MCDS is a specialization of Minimum Weighted $r$-Dominating Set. Furthermore, like other dominating set problems [13], MCDS ($G, r$) is equivalent to MCDS ($G^r, 1$), where $G^r$ denotes the $r$th power of $G$. We now establish that minimizing average congestion is NP-hard.

**Theorem 2.1.** MCDS is NP-hard.

**Proof.** We show that MCDS with $r = 1$ is equivalent to Minimum Dominating Set (MDS) when $G$ is regular. By Lemma 2.1, if $G$ is $d$-regular, then $\text{cong}_r(S) = \frac{1}{|V|} \sum_{u \in S} |N_G[u]| = (1 + d) |S|/|V|$. Thus, any minimum
congestion dominating set must also be minimum in size. The result follows directly, since MDS is NP-hard in regular graphs.

Additionally, we observe that determining the value of \( \text{mac}_c(G) \) is hard through a connection to perfect codes. An \( r \)-perfect code is a dominating set \( D \) such that every vertex \( v \in V \) satisfies \( |N[v] \cap D| = 1 \). It is NP-complete to determine whether a graph has a perfect code \(^{[8]} \). Since \( G \) admits an \( r \)-perfect code if and only if \( \text{mac}_c(G) = 1 \), this completes the proof.

The minimum congestion dominating set therefore indicates the “distance” to a perfect code. We turn that notion into a graph editing problem and obtain the following result.

**Theorem 2.2.** Let \( \text{PCE}(G) \) be the minimum \( k \) so \((G,k)\) is a yes-instance of Perfect Code Editing. Then \( \text{PCE}(G) \leq (\text{mac}(G) - 1)n \leq 2 \cdot \text{PCE}(G) \).

**Proof.** Given a dominating set \( D \subseteq V \) that attains \( \text{mac}(G) \), let \( v \in V \) be a vertex such that \( \text{cong}(D,v) > 1 \). Then, removing an edge \( uv \) for any \( u \in D \setminus \{v\} \) will decrease \( \text{mac}(G) \cdot n \) by at least 1, so \( \text{PCE}(G) \leq (\text{mac}(G) - 1)n \). Given a perfect code \( D' \subseteq V \) of \( G' \), any edge addition can increase \( (\text{mac}(G) \cdot n) \) by at most 2, so \( (\text{mac}(G) - 1)n \leq 2 \cdot \text{PCE}(G) \). \( \square \)

**2.1 Properties of Minimum Congestion Dominating Sets** In general, a minimum congestion dominating set will not also be a minimum dominating set, and in Figure 1 we give a construction proving their sizes can diverge arbitrarily. Further, by definition, we have \( \text{mac}(G) \geq 1 \) for any graph; we give a degree-based upper bound below.

**Theorem 2.3.** \( \text{mac}(G) \leq (\bar{d} + 1)/2 \) for every graph \( G \), where \( \bar{d} \) is the average degree of \( G \).

**Proof.** Let \( D \subseteq V \) be a minimal dominating set (i.e. every proper subset \( D' \subset D \) is not dominating). Then, \( \mathcal{D} := V \setminus D \) is also a dominating set. Now, \( \text{cong}(D) + \text{cong}(\mathcal{D}) = 1/2 \sum_{v \in D} |N[v]| + 1/2 \sum_{v \in \mathcal{D}} |N[v]| \). Rewriting the right-hand side as \( 1/2 \sum_{v \in V} |N[v]| = \bar{d} + 1 \), we have \( \text{mac}(G) \leq \min\{\text{cong}(D), \text{cong}(\mathcal{D})\} \leq (\bar{d} + 1)/2. \) \( \square \)

It is natural to ask how close this is to best-possible; we leave this as an open question.

**Conjecture 2.1.** There is no constant \( c \) such that \( \text{mac}(G) \leq c \) for every graph \( G \).

**2.2 Algorithms** We now describe several greedy algorithms along with an ILP formulation for MCDS.

**2.2.1 Greedy Algorithms** We first recall the standard greedy algorithm for Minimum Dominating Set which we call Dom-Degree. At each step, the algorithm chooses a vertex \( v \in V \) such that the number of undominated vertices in \( N^+[v] \) is maximized. To target MCDS, we adapt this to prioritize based on the ratio of the undominated vertices. Specifically, Dom-Ratio chooses a vertex \( v \in V \) such that \( \frac{|N^+[v]|}{|N^-[v]|} \) is maximized at each step given a partial dominating set \( D \subseteq V \). In both algorithms, ties are broken arbitrarily.

While Dom-Degree is an \( \mathcal{O}(r \log \Delta) \)-approximation for \( r \)-Dominating Set, it is not for MCDS (Theorem A.2). In contrast, Dom-Ratio can produce sets which are arbitrarily bigger than the minimum dominating set. (Figure 1 (right) is an example of this), yet we prove it is an approximation for MCDS.

**Theorem 2.4.** Dom-Ratio is an \( \mathcal{O}\left(\sqrt{\Delta r}\right) \)-approximation algorithm for MCDS.

To evaluate smarter tie-breaking strategies, we define Dom-Degree-Plus to be Dom-Degree with ties broken.

---

\(^{[8]}\) An abridged proof of Theorem 2.4 is in Appendix A.
using Dom-Ratio’s criteria (the ratio of undominated vertices), and Dom-Ratio-Plus analogously using Degree’s criteria (the number of undominated vertices). Further ties are randomly broken.

2.2.2 Integer Programming We observe that one may obtain optimal solutions to r-DOMINATING SET and MCDS using an ILP solver allowing empirical evaluation of approximation ratios. We use the following ILP formulation for r-DOMINATING SET:

\[
\text{Dom-MDS} \\
\text{Let } x_v \in \{0, 1\} \text{ be variables for all } v \in V. \\
\text{Minimize } \sum_{v \in V} x_v \\
\text{Subject to } \sum_{w \in N^r[v]} x_w \geq 1 \text{ for all } v \in V.
\]

Similarly, we formulate MCDS as follows:

\[
\text{Dom-MAC} \\
\text{Let } x_v \in \{0, 1\} \text{ be variables for all } v \in V. \\
\text{Minimize } \sum_{v \in V} |N^r[v]| x_v \\
\text{Subject to } \sum_{w \in N^r[v]} x_w \geq 1 \text{ for all } v \in V.
\]

This is based on the fact that MCDS is a specialization of MINIMUM WEIGHTED r-DOMINATING SET. In both cases, the solution set is provided by \(\{v \in V : x_v = 1\}\). In Section 4, we use these exact solutions to evaluate the quality of our algorithms.

3 Neighborhood Partitioning

We now turn to the second problem arising in our metagenomics application: partitioning the vertex set into pieces around a set of shallow dominating sets as evenly as possible. We first formalize the notion of a neighborhood partitioning and use variance to define its balance. After establishing that the resulting problem is NP-hard (and remains so under very restrictive conditions), we show that for radius 1, a flow-based approach gives a polynomial-time solution. Finally, we describe several algorithmic approaches for obtaining both exact and heuristic solutions.

We begin by considering the more general setting where we are given \(L\), a set of landmarks\(^4\) and ask for a partition of \(V\) into \(|L|\) disjoint sets so that (a) each piece contains exactly one landmark, (b) every vertex is assigned to a piece with one of its closest landmarks from \(L\) and (c) for every piece \(A\), the induced subgraph on \(A\) preserves the distance between the landmark and other vertices in \(A\).

**Definition 3.1.** Given a graph \(G = (V, E)\) and landmarks \(L = \{u_1, \ldots, u_\ell\} \subseteq V\), we say \(A = \{A_1, \ldots, A_\ell\}\) is a neighborhood partitioning of \(G\) with respect to \(L\) if and only if \(d_{G[A]}(v, u_i) = d_G(v, L)\) for every \(1 \leq i \leq \ell\), \(v \in A_i\).

We note that if \(L\) is an \(r\)-dominating set of \(G\), the resulting pieces will necessarily have radius at most \(r\), making \(G/A\) an \(r\)-shallow minor\(^5\). This is essential for spacegraphcats to maintain efficiency guarantees when computing the dominating sets in their hierarchy, since it ensures graphs remain within the assumed sparse class (bounded expansion) \([2]\).

We now define the problem of finding a set of pieces whose size distribution is as even as possible.

**Balanced Neighborhood Partitioning (BNPart)**

**Input:** A graph \(G = (V, E)\) and landmarks \(L = \{u_1, \ldots, u_\ell\} \subseteq V\)

**Problem:** Find a neighborhood partitioning \(A = \{A_1, \ldots, A_\ell\}\) of \(G\) on \(L\) such that the piece-size population variance \(\text{Var}(\{|A| : A \in \mathcal{A}\})\) is minimized.

A key observation is that since \(V\) and \(L\) are given to us, the average size of \(A\) is \(|V|/|L|\), which is fixed. This results in the following equivalence:

**Theorem 3.1.** A neighborhood partitioning \(A\) gives the minimum piece-size variance if and only if the square sum of the piece-sizes is minimized.

The hardness of BNPart is considered in detail in Appendix A.2 and summarized in the following theorem.

**Theorem 3.2.** BNPart is NP-hard. It remains hard even if \(\max_{v \in V} d(v, L) = 2\) or \(|L| = 2\).

However, if \(\max_{v \in V} d(v, L) \leq 1\) (equivalently, \(L\) is a \((1, 1)\)-dominating set of \(G\)), then BNPart becomes tractable.

**Theorem 3.3.** BNPart can be solved in \(O(|L|n^3)\) if \(\max_{v \in V} d(v, L) \leq 1\).

Our proof relies on Theorem 3.1 and the fact that the problem is equivalent to SIZE BALANCED ASSIGNMENT (SBA), defined in Appendix A.2 which can be transformed into an instance of the maximum-cost minimum flow problem (proof in Appendix A.2).
The underlying ideas of our preprocessing are orienting the following data reduction, noting that by definition, all assigned to the same piece. We encapsulate this idea in If \( v \) thus \( v \) landmark must include one of neighborhood kernel preprocessing a removing unnecessary edges. We call the output of this edges outward from the landmarks, creating layers of then we can safely remove \( e \) of length \( BNPart \)

3.1 Algorithms

We now present several algorithms for computing neighborhood partitions along with a quadratic programming formulation for BNPART. In all cases, we work with a preprocessed instance which we refer to as a (compact) neighborhood kernel which can be computed in linear time. Our first two algorithms apply greedy strategies and are complemented by an exact branch-and-bound algorithm based on ideas from an FPT algorithm on bounded-degree instances. We require the following conditions.

- Each vertex \( v \in V_c \) is the representative for its bag, and must be the closest to \( L \) in \( G \) among \( \phi(v) \).
- All bag members must be assigned to the same landmark in any valid neighborhood partitioning.
- \( \phi(v) \) is maximal subject to these conditions.

We visualize this process in Figure 2. In Appendix A.3, we describe algorithms (Nbr-Kernel and Compact-Nbr-Kernel) for creating (compact) neighborhood kernels and show that they run in \( O(n + m) \) time. Finally, we note that the maximum degree is closed under these transformations (Lemma A.3).

**Algorithm 1. Prt-Layer \((G, L)\)**

**Input** Graph \( G = (V, E) \), landmarks \( L = \{u_1, \ldots, u_\ell\} \subseteq V \)

**Output** Neighborhood partitioning \( \mathcal{A} \)

1. \( H \leftarrow Nbr-Kernel \((G)\) \)
2. Partition \( V \) into layers \( \{V_i \mid v \in V : d(v, L) = i \} \)
3. Initialize map \( f : V \rightarrow L \) to \( f(u) = u \) for all \( u \in L \)
4. for \( i \leftarrow 1 \) to max\(_{v \in V} d(v, L) \) do
5. \( R \leftarrow \{(f(x), x) \mid x \in \bigcup_{j=0}^{i-1} V_j\} \) \( \triangleright \) initialize relations for assigned vertices
6. \( R \leftarrow R \cup \{(f(w), v) \mid v \in V_i, w \in N_H(v)\} \)
7. update \( f \) with SBAP\((L, \bigcup_{i=0}^\ell V_j, R)\)
8. return \( f^{-1} \) \( \triangleright \) map from \( L \) to assigned vertices

\*By definition, \( L \subseteq V_c \subseteq V \).
3.1.2 Heuristic Algorithms We have developed two heuristics for BNPart. \texttt{Prt-Weight} is a linear-time $O(n + m)$ greedy algorithm that works on the compact neighborhood kernel. From landmarks, it traverses all bags in BFS order, assigning each bag to the smallest piece among its candidates.

\texttt{Prt-Layer} is a polynomial-time $O((L)n^3)$ algorithm that is exact when $L$ is a $(1,1)$-dominating set and heuristic otherwise. It works on the neighborhood kernel and solves SBA at each layer starting from the one closest to landmarks (a layer $V_i$ is a set of vertices such that the distance to the closest landmark is $i$).

3.1.3 Branch-and-bound Algorithm Before elaborating on our exact algorithm for BNPart, we show the following motivating result.

**Theorem 3.4.** BNPart is fixed parameter tractable (FPT) in graphs of bounded degree parameterized by $k$, the number of vertices equidistant from multiple landmarks.

*Proof.* Given a graph $G = (V,E)$, landmarks $L \subseteq V$, and their compact neighborhood kernel $(H_c,\phi)$, let $\Delta$ be the maximum degree of $G$. Let $X$ be $V(H_c) \setminus L$. Since all vertices in $X$ are equidistant from multiple landmarks, $|X| \leq k$. From Lemma \ref{lem:degree_bound}, each vertex in $H_c$ has at most $\Delta$ in-neighbors. Every vertex in $X$ must be in the same piece as one of its in-neighbors for a valid neighborhood partitioning, giving at most $\Delta^k$ possible assignments. Other computations can be done in polynomial time in $n$, so the brute-force approach results in time $O(\Delta^k n^{O(1)})$. \hfill \Box

Our algorithm \texttt{Prt-Branch} (see Appendix A.3 for details) reinforces this idea by combining efficient base-case handling with naive branch-cut functionality. For a base case, we apply \texttt{Prt-Layer} if possible; specifically, if there is only one layer $X_i$ left and all bags are of size 1, \texttt{Prt-Layer} gives an exact solution in $O((L)n^3)$ time. To obtain a lower-bound for a branch cut, we exploit the fact that our partial solutions $A'$ satisfy the following property: any solution $A$ extending $A'$ has $\sum_{A \in A'}|A|^2 \geq \left(\sum_{A \in A'}|A'|^2\right) + (n - |\cup A'axter|)|/|L|$. The total running time of this algorithm is $O(\Delta^k |L|n^3)$.

3.1.4 Quadratic Programming Finally, we give a quadratic programming formulation. Given a graph $G = (V,E)$ and landmarks $L \subseteq V$, let $(H_c = (V_c,E_c),\phi)$ be the compact neighborhood kernel of $G$ on $L$. Let $x_{u,v} \in \{0,1\}$ be variables for all $u \in L, v \in V_c$; set $x_{u,v} = 1$ if and only if $v$ is assigned to landmark $u$.

\begin{align*}
\text{Minimize} & \quad \sum_{u \in L} \left(\sum_{v \in V_c} |\phi(v)| x_{u,v}\right)^2 \\
\text{Subject to} & \quad \sum_{u \in L} x_{u,v} = 1 \quad \text{for all } v \in V_c, \\
& \quad \sum_{w \in N_{H_c}(v)} x_{u,w} \geq x_{u,v} \\
& \quad \text{for all } u \in L, v \in V_c \setminus L.
\end{align*}

By Theorem \ref{thm:bounded_degree}, the objective function guarantees minimum variance. The first constraint enforces that every vertex must be assigned to exactly one landmark, and the second constraint guarantees that every non-landmark must be in the same piece as one of its in-neighbors. The balanced neighborhood partitioning is given by $A = \{A_u, \ldots, A_{w}\}$ where $A_u = \bigcup_{v \in V_c} x_{u,v}=1$

4 Experiments

To complement our theoretical results, we implemented and evaluated our algorithms on a diverse corpus of networks including cDBGs constructed from real metagenomes, small instances from DIMACS10 \cite{dimacs}, and graphs from \cite{dbgs}. We categorized each graph as small, medium, or large based on the number of edges; detailed statistics are shown in Table \ref{tab:stats}. All algorithms were implemented in C++ and experiments were conducted on a 40-core Linux machine with 190 GB of memory (details in Appendix B).

4.1 Sparse Dominating Sets To evaluate our algorithms for finding a low-congestion dominating set, we first tested the effectiveness of tie-breaking strategies by running \texttt{Dom-Degree} (denoted \texttt{deg}), \texttt{Dom-Ratio (ratio)}, \texttt{Dom-Degree-Plus (deg+)} as well as \texttt{Dom-Ratio-Plus (ratio+)} on small instances. The algorithms with a tie-breaking strategy consistently outperformed those without one, finding smaller dominating sets in 78% of the experiments and less-congested dominating sets in 93% of them. Running time was also improved by tie-breaking strategies in 55% of instances, due to its proportionality to solution size.

Based on these findings, we restricted our attention to \texttt{deg+} and \texttt{ratio+} for the remaining experiments. We evaluated them, along with \texttt{Dom-SGC (sgc-d)}, \texttt{Dom-MDS (mds)}, and \texttt{Dom-MAC (mac)} on the full corpus with radii $\{1, 2\}$ (plus radii $\{3, 5, 10\}$ on small and medium instances). We used a default 3-hour timeout, reduced to 1 hour for \texttt{mds} and \texttt{mac} when the radius was $>1$ or the instance was medium/large.

Figure \ref{fig:runtime} (left) shows the distribution of running times sorted by $\|G\|^r$, the number of edges in the $r$-th...
power of $G$. We observe a linear trend for $sgc-d$, $deg+$, and $ratio+$, establishing efficiency. While the exact algorithms $mds$ and $mac$ are prone to timing out, they did finish on some larger instances. We hypothesize this success is directly related to reduction rules in the ILP solver related to vertices with degree 1 and 2, as supported by Table 2.

To evaluate solution quality, we plotted the relationship between the solution size and average congestion, both relative to the best-known (Figure 3, right). As might be expected, $deg+$ and $mds$ find smaller, more congested dominating sets, while $ratio+$ and $mac$ find larger, sparser ones. The algorithm used in the prior metagenomic analysis ($sgc-d$) intermediates between the two, but often at the cost of being much larger or more congested.

### 4.2 Balanced Neighborhood Partitioning

Turning to the problem of generating similarly-sized pieces around a set of dominators, we tested both the scalability and solution quality of the algorithms for BNPART as well as the impact of choosing smaller versus sparser dominating sets. To this end, we ran algorithms $Prt-SGC$ ($sgc-p$), $Prt-Weight$ ($wt$), $Prt-Layer$ ($layer$), $Prt-Branch$ ($branch$), and $Prt-QP$ ($qp$) using dominating sets produced by $sgc-d$, $deg+$, and $ratio+$. All runs were subject to a 1-hour timeout.

We first evaluated the running time and timeout rate for all BNPART algorithms (Table 1) on the corpus of small instances. The algorithm $wt$ was consistently fastest, followed by $sgc-p$; both had no timeouts. The polynomial-time $layer$ was slower than the linear-time approaches but still completed nearly all small and medium instances. It is particularly notable that when $r = 1$, it successfully output optimal solutions, which $qp$ could not find. We also observed that $qp$’s performance improves when given a cDBG and dominators from $deg+$ or $ratio+$. Lastly, $branch$ was unable to finish with larger radii, even on these small instances, eliminating it from use in additional experiments.

| Config | Algorithm | Rel. Runtime | %Timeout |
|--------|-----------|--------------|----------|
| All    | $wt$      | 1.0          | —        |
|        | $sgc-p$   | 28.1         | —        |
|        | $layer$   | 49.8         | —        |
|        | $branch$  | 135.6        | 75.0%    |
|        | $qp$      | 667.5        | 36.1%    |
| cDBG   | $wt$      | 1.0          | —        |
|        | $sgc-p$   | 35.4         | —        |
|        | $layer$   | 28.9         | —        |
|        | $branch$  | 71.8         | 80.0%    |
|        | $qp$      | 149.3        | 13.3%    |

Table 1: Average running time relative to $wt$ (Rel. Runtime) and timeout rate for BNPART algorithms on the corpus of small instances.

For both remaining experiments, we ran on all small and medium instances. To assess the impact of dominator selection on the variance of piece sizes, we restrict our attention to $wt$ and measure the standard deviations of the piece sizes (Figure 4 (left)). In general, dominating sets from $ratio+$ tend to result in more balanced neighborhood partitionings than $deg+$, with some outliers at larger radii. We note that while $sgc-p$ achieves the least variance (except at $r = 10$), this is...
explained by its dominating sets being larger (resulting in a smaller mean piece size).

Finally, we compare the solution quality of our greedy algorithms for neighborhood partitioning using fixed dominating sets found by \textit{ratio+}. In Figure 4 (right), we see that \textit{wt} achieves more balanced pieces than \textit{sgc-p} at all radii while keeping the same asymptotic running time. The more expensive \textit{layer} performs even better, achieving nearly optimal results. With larger radii (e.g. \( r \geq 3 \)), \textit{wt} performs as well as \textit{layer} does.

### 4.3 Metagenome Neighborhood Queries

Given the motivation for this work is improving the metagenomics analysis pipeline from [2], we also assessed the impact of our techniques in this setting. Specifically, we tested whether using sparse dominating sets and balanced neighborhood partitionings from our algorithms affected the containment and similarity of neighborhood queries in HuSB1, a large real-world metagenome [6]. We used Brown et al.’s replication pipeline\footnote{https://github.com/ctb/2020-rerun-hu-s1} and compared our algorithms (\textit{deg+} and \textit{ratio+} for MCDS; \textit{wt} for \textit{BNPART}) to those used in [2] (\textit{sgc-d} and \textit{sgc-p}, respectively). While [2] restricted their attention to radius 1, we also include results from radius 2, as this is of interest in ongoing related work by the Brown lab.

Before running neighborhood queries, we evaluated the piece size distributions as shown in Figure 5 (left, center). We observe that independent of dominator selection, \textit{wt} successfully reduced the piece size variance. When comparing dominating sets, we see a trend consistent with other experiments. \textit{sgc-d} chooses many dominators, resulting in smaller and more uniform piece sizes. \textit{deg+} has the fewest dominators and higher variance, larger pieces and \textit{ratio+} gives larger, balanced pieces.

Finally, we re-ran the neighborhood queries from [2]. While the containment was completely preserved, the similarities\footnote{Jaccard similarity to original query} exhibited mild variation, shown in Figure 5 (right). While there is a slight overall reduction (partly explainable by having fewer pieces), the difference is marginal at \( r = 2 \). These preliminary results indicate that our techniques can significantly improve the balance of piece sizes (leading to more efficient downstream analysis when a hierarchy of dominating sets is constructed) without significantly degrading fidelity of queries. Additional experiments are needed to better understand the impact of this work in the metagenomic setting.

### 5 Conclusions & Future Work

This paper tackles two problems arising in a recent approach to metagenome analysis: finding \((r\text{-})\)dominating sets which minimize the number of vertices with multiple closest dominators, and partitioning the vertices of a graph into connected pieces around a set of landmarks minimizing variance of the piece-sizes while guaranteeing every vertex is assigned to some closest landmark.

We formalize the first using the notion of \textit{congestion}, and show that finding minimum congestion \(r\text{-}\)dominating sets (MCDS) is NP-hard. We introduce linear-time algorithms for finding low-congestion dominating sets and evaluate their effectiveness on a large corpus of real-world data, showing trade-offs between the solution size and average congestion. It remains open whether there is a constant upper bound on the minimum average congestion (Conjecture 2.1; the largest value observed in our experiments was 3.7351); further, we believe our approximation bound on MCDS can be tightened. We are intrigued by the connection of this problem to \textit{Perfect Code Editing}, and believe this...
could be a fruitful direction for future work.

Turning to the partitioning problem (BNPart), we show that at radius 1, flow-based techniques give exact solutions in polynomial time, but the problem becomes NP-hard even in very restricted cases as soon as \( r \geq 2 \). Our heuristics, however, produce nearly optimal results in our experiments. Further, we show that using sparse dominating sets (such as those from ratio+) improves both the running time and solution quality of algorithms for BNPart. A natural question is whether there are alternative measures of partition balance in real-world data which avoid the inverse relationship between dominating set size and the variance of the piece size distribution.

Finally, we integrate our algorithms into the metagenomic analysis pipeline used in [2] and demonstrate that on a large real metagenome (HuSB1), sparse dominating sets and balanced neighborhood partitionings reduce piece size variability relative to the prior approaches without significantly degrading the fidelity of neighborhood queries. It remains to see how these preliminary results extend to other metagenomic datasets and queries.

Acknowledgements

Special thanks to Rachel Walker for early discussions on notions of sparse dominating sets & related terminology, and to Titus Brown & Taylor Reiter for assistance with metagenomic datasets and the spacegraphcats software. This work was supported by grant GBMF4560 to Blair D. Sullivan from the Gordon and Betty Moore Foundation.

References

[1] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. Graph partitioning and graph clustering, volume 588. American Mathematical Society, Providence, RI, 2013.

[2] C. T. Brown, D. Moritz, M. P. O’Brien, F. Reidl, T. Reiter, and B. D. Sullivan. Exploring neighborhoods in large metagenome assembly graphs using spacegraphcats reveals hidden sequence diversity. Genome Biology, 21(1):164, Jul 2020.

[3] R. Chikhi, A. Limasset, and P. Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics, 32(12):i201–i208, 06 2016.

[4] Z. Dvořák. Constant-factor approximation of the domination number in sparse graphs. European Journal of Combinatorics, 34(5):833–840, 2013.

[5] C. Einarson and F. Reidl. A General Kernelization Technique for Domination and Independence Problems in Sparse Classes. In 15th International Symposium on Parameterized and Exact Computation (IPEC 2020), volume 180 of Leibniz International Proceedings in Informatics (LIPIcs), pages 11:1–11:15, 2020.

[6] P. Hu, L. Tom, A. Singh, B. C. Thomas, B. J. Baker, Y. M. Piceno, G. L. Andersen, J. F. Banfield, and N. Dubilier. 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, 2016.

[7] L. Jaffke, O. joung Kwon, T. J. Strømme, and J. A. Telle. Mim-width iii. graph powers and generalized distance domination problems. Theoretical Computer Science, 796:216–236, 2019.

[8] J. Kratochvıl. Regular codes in regular graphs are difficult. Discrete Mathematics, 133(1):191–205, 1994.

[9] M. Livingston and Q. Stout. Perfect dominating sets. Congressus Numerantium, 79, 08 1997.

[10] M. Nguyen, M. Hà, D. Nguyen, and T. Tran. Solving the k-dominating set problem on very large-scale networks. Comp. Social Networks, 7, 07 2020.

[11] C. Quince, A. W. Walker, J. T. Simpson, N. J. Loman, and N. Segata. Shotgun metagenomics, from sampling to analysis. Nature biotechnology, 35(9):833–844, 2017.

[12] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In AAAI, 2015.

[13] P. J. Slater. R-domination in graphs. J. ACM, 23(3):446–450, July 1976.
A Theoretical Results

A.1 Notation & Terminology Given a graph \( G = (V,E) \), we write \( n = |G| = |V| \) for the number of vertices and \( m = |G| = |E| \) for the number of edges. The distance between two vertices \( x, y \in V \), denoted by \( d_G(x,y) \), equals the minimum number of edges in an \( x,y \)-path in \( G \); if no such path exists, we set \( d(x,y) := \infty \). The distance between a vertex \( v \in V \) and vertex set \( X \subseteq V \), denoted by \( d_G(v,X) \), is defined as \( d_G(v,X) := \min_{x \in X} d_G(v,x) \). We use \( N(v) \) and \( N[v] \) to denote the open and closed neighborhoods of a vertex \( v \), respectively. We write \( N^r[v] \) for the \( r \)-neighborhood, the set of all vertices at distance at most \( r \) from \( v \). For a vertex set \( X \subseteq V \), \( N^r[X] \) denotes the union of \( N^r[x] \) for all \( x \in X \). The \( r \)-th power \( G^r \) of \( G \) is defined as \( G^r := (V, \{ uw | u, v \in V, d_G(u,v) \leq r \}) \).

We write \( \deg_G(v) = \deg(v) \) for the degree of a vertex \( v \), and \( \delta \) and \( \Delta \) for the minimum and maximum degree of \( G \), respectively. We denote the induced subgraph of \( G \) on a set \( X \subseteq V \) by \( G[X] \). For an edge \( e \in E \), we write \( G/e \) for the graph formed from \( G \) by contracting the edge \( e \); for a vertex set \( X \subseteq V \), \( G/X \) denotes the graph obtained by contracting all edges in \( G[X] \).

A.2 Neighborhood Partitioning Proofs First, we prove Theorem 3.1 which relates the minimum variance to the minimum sum of squares.

Proof of Theorem 3.1 Let \( X = \{x_1, \ldots, x_|X|\} \) be the piece sizes of \( A \). By definition of population variance, \( \text{Var}(X) = \frac{1}{|X|} \sum_{i=1}^{\ell} (x_i - \mu)^2 = \left( \frac{1}{|X|} \sum_{i=1}^{\ell} x_i^2 \right) - \mu^2 \), where \( \mu = \frac{1}{|X|} \sum_{i=1}^{\ell} x_i = |V|/\ell \). Since \( \mu \) is independent of \( A \), the minimum variance is always achieved with the minimum square sum. \( \square \)

Next, we define Size Balanced Assignment, which can be viewed as a generalization of an assignment problem where every agent is required to have the same number of tasks.

Size Balanced Assignment (SBA)

Input: Agents \( A \), tasks \( T \), and a binary relation \( R \subseteq A \times T \).

Problem: Find a function \( \phi : T \rightarrow A \) such that \( (\phi(t), t) \in R \) for every \( t \in T \), and the population variance of the number of assigned tasks for each agent \( \text{Var}(\{ |\phi^{-1}(a)| : a \in A \}) \) is minimized.

We use the following results to prove Theorem 3.3.

Theorem A.1. SBA can be solved in time \( \mathcal{O}(|A|^2|T|^2 + |A||T|^3) \).

Proof. The same argument as Lemma 3.3 applies here, and SBA is equivalent to asking for an assignment minimizing the square sum of \( |\phi^{-1}(a)| \) for each \( a \in A \).

We transform this problem to the minimum-cost maximum flow problem with integral flows. Construct a multi-digraph \( G = (V,E) \). The set of vertices \( V \) includes \( A, T, \) the source \( \alpha \) and sink \( \beta \) of the flow. We construct the edges \( E \) as follows. \( A \) and \( T \) preserve their relation \( R \) as edges, oriented from \( A \) to \( T \). For every task \( t \in T \), add one edge from \( t \) to \( \beta \). For every agent \( a \in A \), add multiple edges from \( \alpha \) to \( a \) so that the number of incoming edges matches the number of outgoing edges at \( a \). Only these edges from \( \alpha \) have costs. For the \( i \)th edge to each \( a \), denoted \( e_{a,i} \), we set the cost to \( 2i - 1 \). All edges have capacity \( 1 \). There are at most \( 2|R| + |T| \) edges in this graph, so the construction can be done in \( \mathcal{O}(|A||T|) \) time. See Figure 6 for an example.

Figure 6: Transformation of SBA into a flow problem. The binary relation \( R \) between agents \( A \) and tasks \( T \) is shown as blue edges. Every edge has unit capacity; non-zero edge costs are shown as numbers.

Let \( c(e) = 1 \) be the capacity and \( w(e) \) be the cost of each edge. Also, let \( \hat{w} \) be the minimum cost and \( \hat{f} \) be the maximum flow amount of \((G,c,w)\) from \( \alpha \) to \( \beta \). We claim that for any \( k \in \mathbb{Z} \), we get \( \hat{w} \leq k \) and \( \hat{f} = |T| \) if and only if the SBA instance \((A,T,R)\) admits a valid assignment \( \phi : T \rightarrow A \) with the square sum of the number of assigned tasks \( \sum_{a \in A} |\phi^{-1}(a)|^2 \leq k \).

To prove the correctness in the forward direction, suppose there exists a flow of amount \( |T| \) and cost \( \hat{w} \leq k \). The flow at \( t \in T \) is 1, and there is incoming flow from only one agent \( a \in A \), which corresponds to assignment \( t \mapsto a \). Likewise, the flow at \( a \in A \) must equal \( |\phi^{-1}(a)| \). The minimum cost can be achieved by \( \sum_{a \in A} \sum_{i=1}^{\ell} (2i - 1) = \sum_{a \in A} |\phi^{-1}(a)|^2 \leq k \). On the other hand, assume that there exists a valid assignment \( \phi : T \rightarrow A \). Then, we can construct the following flow \( f : E \rightarrow \mathbb{Z} \) with amount \( |T| \) and cost \( k \).
Thus, $d \leq k$, as desired.

$$f(e_{a,i}) = \begin{cases} 1 & \text{if } i \leq \lfloor \phi^{-1}(a) \rfloor \\ 0 & \text{otherwise} \end{cases}$$

$$f((a,t)) = \begin{cases} 1 & \text{if } a = \phi(t) \\ 0 & \text{otherwise} \end{cases} \quad \text{for every } a \in A$$

$$f((t, \beta)) = 1 \quad \text{for every } t \in T$$

The minimum-cost maximum flow problem with integral capacities can be solved using the Bellman-Ford algorithm, which has running time $O(\ell_{\max} |V| |E|)$. In our case, that is $O(|T|(\ell_A + |T|)|A| |T|) = O(|A|^2 |T|^2 + |A||T|^3)$. \hfill \Box

### A.2.1 Hardness of BNPart

**Theorem 3.3** states that BNP is tractable when the landmarks form a dominating set of $G$.

**Proof of Theorem 3.3**. Since non-landmarks can be assigned to any neighboring landmark, the problem is equivalent to SBA with agents $L$, tasks $V$, and relation $\{(u, u) : u \in L\} \cup \{(u, v) : u \in L, v \in N(u) \setminus L\}$. By Theorem A.1, running time is $O(|L|^2 |V|^2 + |L||V|^3) = O(|L|n^3)$.

We break the proof of Theorem 3.2, which states the hardness of BNPart, into two lemmas, first showing the case when the landmarks are a 2-dominated-set by reduction from Exact Cover by 3-sets.

#### Exact Cover by 3-sets (X3C)

**Input:** Set $X$ with $|X| = 3q$ and a collection $C$ of 3-element subsets of $X$.

**Problem:** Is there a subcollection $C' \subseteq C$ such that every element of $X$ occurs in exactly one member of $C'$?

**Lemma A.1.** BNPart is NP-hard when the landmarks are distance at most two from all vertices.

**Proof.** Consider an instance of X3C with $n = |C|$. If $n < q$, then output “no”. Assuming $n \geq q$, we construct a graph $G = (V, E)$ as follows. The set of vertices $V$ includes $n$ landmarks $L = \{u_i : 1 \leq i \leq q\}$, $C$, $X$, and attached leaves; 3 leaves are attached to each of the $n-q$ vertices in $L$. The edges are constructed as follows: $L$ and $C$ form a biclique; for each $C \subseteq C$, $C$ is connected to all elements of $X$ if it contains. This construction can be done in time $O(n^3)$. See Figure 7 (b) for an example.

We return “yes” if and only if BNPart returns a partition with pieces of equal size (by construction, this must be 5). Let $L' \subseteq L$ be the landmarks that do not have attached leaves. If the given X3C instance admits an exact cover $C'$, then for each $C \subseteq C'$, we assign one landmark in $L'$ to $C$ and all elements of $C$. Then, we assign each of $L \setminus L'$ to $n-q$ unused sets in $C$, resulting in pieces of equal sizes.

Conversely, suppose BNPart returns an equally-sized partition. Because each piece must have 5 vertices, $n-q$ sets in $C$ are assigned to $L \subseteq L'$. Also, since $|L'| = q$, each landmark in $L'$ must have exactly one vertex of $C$ in its piece. Then, each piece contains exactly 3 elements in $X$, and this is an exact cover by 3-sets. \hfill \Box

**Lemma A.2.** BNPart is NP-hard when there are only two landmarks.

**Proof.** We reduce from SATISFIABILITY (SAT). Let $x_1, \ldots, x_n$ be the variables and $\phi_1, \ldots, \phi_m$ be the clauses appearing in a SAT instance. We construct a graph $G = (V, E)$ as follows. The set of vertices $V$ includes two landmarks $L = \{u_1, u_2\}$, $2n$ vertices $X = \{x_1, x_1', \ldots, x_n, x_n'\}$ representing SAT literals, $n$ variables $Y = \{y_1, \ldots, y_n\}$, $m$ clauses $\Phi = \{\phi_1, \ldots, \phi_m\}$, and attached leaves. A total of $n-1$ leaves are attached to each of $Y \cup \Phi$, and $n(m+n)$ leaves are attached to $u_2$. The set of edges $E$ is constructed as follows: $L$ and $X$ form a biclique; for each $y \in Y$, $y$ is connected to $x_i$ and $x_i'$; for each $\phi \in \Phi$, $\phi$ is connected to all literals in clause $\phi$. This construction can be done in time $O(n(n+m))$. See Figure 7 (b) for an example.

We return “yes” if and only if BNPart returns a partition of equal sizes. Let $\{A, B\}$ be a partition such that $u_1 \in A$ and $u_2 \in B$. If the given SAT instance is satisfiable, then we include $Y$, $\Phi$, and all true literals of $X$ in $A$. $B$ takes the others, and we get $|A| = |B| = 1 + n + n(n+m)$. Conversely, if BNPart returns an equally-sized partition, then $B$ cannot include any of $\Phi$ or $Y$, as it forces $|B| > 1 + n + n(n+m)$ to $|A|$. Because $Y \subseteq A$ and $|X \cap A| = n$, exactly one of $x_i, x_i'$ is in $A$ for every $i$. Thus, $X \cap A$ is a valid assignment for the SAT instance. \hfill \Box

### A.3 Algorithm Analysis

#### A.3.1 Dominating Set Algorithms

**Theorem A.2.** Dom-Degree is not an approximation algorithm for MDSDS.

**Proof.** Let $r = 1$. Consider a graph $G = (V, E)$ that has a biclique $(A, B)$ with $|A| = |B| = k$ and one attached leaf $x_v$ for each $v \in A \cup B$. Thus, $n = |V| = 4k$. In the first two iterations, Dom-Degree should choose vertices $u \in A$ and $v \in B$. But then, all $4(k-1)$ vertices in $V \setminus \{u, v, x_u, x_v\}$ equally have one undominated vertex.
in their neighborhoods. Thus, Dom-Degree may choose $D := A \cup B$, which gives $\text{Dom}(D) = n/8 + 1$. However, $D := V \setminus D$ is a perfect code, assuring $\text{mac}(G) = 1$. □

**THEOREM A.3.** Dom-Degree, Dom-Degree-Plus, Dom-Ratio, and Dom-Ratio-Plus find an $r$-dominating set in time $O(\Delta^2 n \log n)$.

**Proof.** All of these algorithms continue to choose vertices until the entire graph becomes dominated, so the resulting set is an $r$-dominating set.

The running time is also the same among these algorithms. Consider a max heap of $n$ elements, which stores values of greedy criteria for each $v \in V$. We can pick and remove the maximum element in $O(\log n)$ time. After adding a vertex $v$ to a partial dominating set $D$, we need to update at most $N^2[v]$ elements in the heap because all criteria are based on the number of undominated vertices in the $r$-neighborhood. There are at most $n$ iterations, and since $|N^2[v]| \leq \Delta^2 r$, the total running time is $O(\Delta^2 n \log n)$. □

**A.3.2 Proof Sketch of Dom-Ratio Approximation Guarantee** In the interest of space, we sketch here only the main ideas of the proof of Theorem 2.4.

**Proof Sketch of Theorem 2.4.** First, observe that Dom-Ratio behaves in the same way when a graph $G'$ and radius 1 are given instead of $(G, r)$. Since $\Delta(G') \leq (\Delta(G))^r$, it suffices to consider $r = 1$.

Consider a partial dominating set $D \subseteq V$ and chosen vertex $v \in V$ during an iteration of Dom-Ratio. Let us call $t = |N[D]|/|V|$ the global domination ratio, and $\rho(v) = |N[v] \cap N[D]|/|N[v]|$ the local domination ratio. By definition, $v$ has the minimum local domination ratio among all vertices at this point.

A key idea is to consider the minimum possible $t$ for the given $0 \leq \rho \leq 1$. We get the following relationship:

$$
\rho = 0 \quad \text{if } 0 \leq t < \frac{\Delta + 1}{\Delta + 2},
\rho \leq 1 - \frac{1 - t}{\Delta + 1} \quad \text{if } \frac{\Delta + 1}{\Delta + 2} \leq t < \frac{\Delta}{\Delta + 2},
\rho \leq 1 - f^{-1}(t) \quad \text{if } \frac{\Delta}{\Delta + 2} \leq t < \frac{\Delta + 1}{\Delta + 2},
\rho \leq \frac{\Delta + 1}{\Delta + 2} \quad \text{if } t = 1,
\rho = 1 \quad \text{if } t = 1,
$$

where $f(x) = ((1 + \Delta x^2/(1 - x))^{-1}$, and $g^{-1}$ is the inverse function of $g$. Next, we relate $\text{Dom}(G)$ to $t$ and obtain the following: $\text{Dom}(G) \leq \int_0^t h(t)dt$, where

$$
h(t) = \begin{cases} 
1 & \text{if } 0 \leq t < \frac{\Delta + 1}{\Delta + 2}, \\
\frac{\Delta + 1}{1 - t} & \text{if } \frac{\Delta + 1}{\Delta + 2} \leq t < \frac{\Delta}{\Delta + 2}, \\
\frac{2}{1 - t} & \text{if } \frac{\Delta}{\Delta + 2} \leq t < \frac{\Delta + 1}{\Delta + 2}, \\
\Delta + 1 & \text{if } \frac{\Delta + 1}{\Delta + 2} \leq t < 1, \\
0 & \text{if } t = 1.
\end{cases}
$$

By evaluation, we get $\text{Dom}(G) \leq \frac{\Delta}{\Delta + 2} + \frac{\Delta}{\Delta + 2}$. Since $1 \leq \text{mac}(G) \leq \text{Dom}(G)$ for all $G$, the approximation ratio is bounded by $O(\sqrt{\Delta})$. □

**A.3.3 Neighborhood Partitioning Algorithms**

Given a graph and landmarks, Algorithm 2 computes the neighborhood kernel in linear time.

**THEOREM A.4.** Nbr-Kernel correctly gives the neighborhood kernel of the graph on the given landmarks in $O(n + m)$ time.
**Algorithm 2. Nbr-Kernel** \((G, L)\)

**Input** Graph \(G = (V, E)\), landmarks \(L \subseteq V\)
**Output** Neighborhood kernel \(H\)

1: \(E' \leftarrow \emptyset\)
2: for \(e \in E\) do
3: \(vw \leftarrow e\) such that \(d(v, L) \leq d(w, L)\)
4: if \(d(v, L) + 1 = d(w, L)\) then
5: \(E' \leftarrow E' \cup \{vw\}\)
6: return \(H = (V, E')\)

**Proof.** The check in Line 3 is consistent with Definition 3.2. For every vertex \(v \in V\), the distance from the closest landmark can be computed by BFS from \(L\) in \(\mathcal{O}(n+m)\) time. The algorithm checks all edges in \(\mathcal{O}(m)\), resulting in \(\mathcal{O}(n+2m) = \mathcal{O}(n+m)\). □

**Algorithm 3. Compact-Nbr-Kernel** \((G, L)\)

**Input** Graph \(G = (V, E)\), landmarks \(L \subseteq V\)
**Output** Compact neighborhood kernel \((H_c, \phi)\)

1: \(H \leftarrow \text{Nbr-Kernel} \ (G, L)\)
2: Initialize \(\psi : V \rightarrow V\) to the identity map
3: for \(v \in V\) sorted by ascending \(d_G(v, L)\) do
4: if \(\text{deg}_H(v) = 1\) then
5: let \(w \in N_H(v)\) \(\triangleright\) only one in-neighbor
6: set \(\psi(v) = w\) \(\triangleright\) \(v\) belongs to bag \(w\)
7: \(H \leftarrow H/\{vw\}\) \(\triangleright\) contract edge \(vw\)
8: return \((H, \psi^{-1})\)

**Theorem A.5.** *Compact-Nbr-Kernel* correctly gives the compact neighborhood kernel of the graph on the given landmarks in \(\mathcal{O}(n+m)\) time.

**Proof.** This algorithm traverses all vertices in BFS order from the landmarks, which guarantees each bag’s representative to be the closest to \(L\). By the definition of BNPart, Line 2 ensures that all bag members must be assigned to the same landmark because a vertex must follow its in-neighbor’s assignment. Lastly, \(\phi(v)\) is maximal for every \(v \in V_c\) because otherwise, there exists a vertex \(x \in V_c\) with at least two in-neighbors \(y, z\) such that \(v = \psi(y) = \psi(z) \neq \psi(x)\). This cannot happen because the algorithm visits \(y, z\) before \(x\) and contracts \(y, z\) into one vertex in \(H_c\).

This modified BFS does not add any extra asymptotic running time to \(\text{Nbr-Kernel}\). Thus, the total running time is \(\mathcal{O}(n+m)\). □

**Lemma A.3.** Given a graph \(G = (V, E)\), landmarks \(L \subseteq V\), and their neighborhood kernel \(H = (V, E')\) and compact neighborhood kernel \((H_c = (V_c, E_c), \phi)\), \(\text{deg}_{H_c}(v) \leq \text{deg}_H(v) \leq \Delta(G)\) for every \(v \in V_c\).

**Theorem A.6.** *Prt-Weight* gives a neighborhood partitioning in \(\mathcal{O}(n+m)\) time.

**Proof.** The algorithm traverses all bags of the compact neighborhood kernel in BFS order from the landmarks. When it processes a bag \(v\), all of \(v\)’s in-neighbors have already been assigned to one of their closest landmarks. Thus, regardless of the choice at \(v\), the result is a valid neighborhood partitioning.

The overall running time is \(\mathcal{O}(n+m)\) because this algorithm performs BFS, and the piece size of each landmark can be updated in \(\mathcal{O}(1)\). □

**Theorem A.7.** *Prt-Layer* gives a neighborhood partitioning in \(\mathcal{O}(\ell |L|^3)\) time. Further, it gives an exact solution to BNPart when \(L\) is a dominating set of \(G\).

**Proof.** If each assigned piece is connected in the neighborhood kernel, then it is a valid neighborhood partitioning because vertices \(V_i\) at layer \(i\) are distance \(i\) away from their closest landmarks, both in the original graph and their assigned piece. The algorithm examines vertices in level order emanating from the landmarks, and by the formulation of SBA, each of the resulting pieces must be connected. Also, from Theorem A.3, this algorithm gives an exact solution when \(L\) is a dominating set.

The running time (other than constructing a neighborhood kernel) is the sum of the time taken for SBA at each layer. At layer \(i\), only the vertices in \(V_i\) have choices (all vertices in earlier layers already have an assigned landmark). In a flow problem, we can efficiently pre-process already assigned vertices, thus resulting in time \(\mathcal{O}(\ell |L||V_i|^3)\). Because of the fact \(V = \bigcup_i V_i\), the total running time is \(\sum_i \mathcal{O}(\ell |L||V_i|^3) = \mathcal{O}(|L|^3)\). □

**Theorem A.8.** *Prt-Branch* gives an optimal solution to BNPart in \(\mathcal{O}(\Delta^k|L|^3)\) time, where \(k\) is the number of vertices equidistant from multiple landmarks.

**Proof.** The algorithm implements the idea presented in the proof of Theorem A.3. The number of possible partitions is bounded by \(\Delta^k\). The correctness of the base case stems from Theorem A.7. It remains for us to prove the correctness of our lower-bound function. Let \(
\ell = |L|, X = \{x_1, \ldots, x_\ell\}\) be the piece sizes of the partial assignment, and \(Y = \{y_1, \ldots, y_\ell\}\) be the optimal extension of \(X\) \((x_i \leq y_i\) for every \(1 \leq i \leq \ell)\). We want to prove \(\sum_{i=1}^{\ell} y_i^2 \geq \left(\sum_{i=1}^{\ell} x_i^2\right) + z^2/\ell\), where
Algorithm 4. Prt-Branch \((G, L)\)

\[
\text{Input}\;\text{Graph} \; G = (V, E), \text{landmarks} \; L = \{u_1, \ldots, u_\ell\} \subseteq V
\]

\[
\text{Output}\;\text{Neighborhood partitioning} \; A
\]

1: let \(A\) be a global variable to store the best solution

2: \((H_c \leftarrow (V_c, E_c), \phi) \leftarrow \text{Compact-Nbr-Kernel} (G)\)

3: \(T \leftarrow \{v \mid v \in V_c \setminus L, |\phi(v)| = 1, N^+(v) = \emptyset\}\)

4: return BranchAndBound\((\{u_1\}, \ldots, \{u_\ell\})\)

5: procedure BranchAndBound\((A')\)

6: if \(\sum_{A \in A} |A|^2 \leq \text{LowerBound}(A')\) then

7: return \(\triangleright \text{branch cut}\)

8: \(U \leftarrow V_c \setminus \bigcup A'\) \(\triangleright \text{unassigned bags}\)

9: if \(U = T\) then \(\triangleright \text{base case}\)

10: find a partition of \(U\) using Prt-Layer

11: update \(A'\) based on the result of Prt-Layer

12: if \(\sum_{A' \in A'} |A'|^2 < \sum_{A \in A} |A|^2\) then

13: \(A \leftarrow A'\)

14: return

15: \(v \leftarrow \{v \mid v \in U \setminus T, N^-(v) \cap U = \emptyset\}\)

\(\triangleright \text{branch on} \; v \; \text{having no unassigned in-neighbors}\)

16: for \(w \in N^-(v)\) do

17: update \(A'\); include \(\phi(v)\) in \(w\)'s piece

18: BranchAndBound\((A')\)

19: procedure LowerBound\((A')\)

20: return \((\sum_{A' \in A'} |A'|^2) + (n - |\bigcup A'|)^2 / \ell\)

\[
z = \sum_{i=1}^\ell y_i - x_i.
\]

This can be shown by the following algebra.

\[
\left(\sum_{i=1}^\ell y_i^2\right) - \left(\sum_{i=1}^\ell x_i^2\right) = \sum_{i=1}^\ell y_i^2 - x_i^2
\]

\[
\geq \sum_{i=1}^\ell (y_i - x_i)^2 = z^2 / \ell + \sum_{i=1}^\ell (y_i - x_i - z / \ell)^2
\]

\[
\geq z^2 / \ell
\]

The running time is determined by the search tree whose depth is at most \(k\) and branching factor is \(\Delta\). The base case requires computation of time \(O(|L|n^3)\), so the overall running time is \(O(\Delta^k |L|n^3)\).

B Experimental Setup

B.1 Dataset Table 2 summarizes the corpus including several graph invariants. All metagenomic cDBGs are generated by BCALM 2 (v2.2.1) with k-size 31. We obtained other graphs from the Network Data Repository [12]. We used the largest connected component if a graph is disconnected.

B.2 Hardware We used identical hardware for all experiments. Each machine is equipped with 40 CPUs (Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz) and 190 GB of memory.

B.3 Software We ran all experiments on CentOS Linux release 7.9.2009. spacegraphcats is written in Python 3. We used Gurobi Optimizer 9.1.2 for ILP and QP. We used Plotly for creating charts. Our algorithms are written in C++ with OpenMP and compiled with gcc 10.2.0.
| Dataset          | Category | $|G|$    | $||G||$ | $\delta$ | $\bar{d}$ | $\Delta$ | degn | $\%V_2$ | diam | $\gamma$ | mac       |
|-----------------|----------|--------|--------|--------|---------|--------|------|--------|------|---------|-----------|
| p8808mo10       | cDBG     | 28,793 | 34,534 | 1      | 2.40    | 7      | 3    | 61%    | 469  | 8,830   | 1.08      |
| 5032-01         | cDBG     | 45,070 | 66,527 | 1      | 2.95    | 8      | 4    | 41%    | 103  | ?       | ?         |
| twofoo          | cDBG     | 49,043 | 59,648 | 1      | 2.43    | 8      | 3    | 62%    | 1,455 | 14,606  | 1.11      |
| ca-GrQe         | collab   | 4,158  | 13,422 | 1      | 6.46    | 81     | 43   | 35%    | 17   | 776     | 1.32      |
| ca-CondMat      | collab   | 21,363 | 91,286 | 1      | 8.55    | 279    | 25   | 21%    | 15   | 2,990   | 1.46      |
| delaunay-n13    | DIM      | 8,192  | 24,547 | 3      | 5.99    | 12     | 4    | 0%     | 49   | ?       | ?         |
| wing-tnodal     | DIM      | 10,937 | 75,488 | 5      | 13.80   | 28     | 8    | 0%     | 26   | ?       | ?         |
| fe-feat2        | DIM      | 11,143 | 32,818 | 3      | 5.89    | 12     | 4    | 0%     | 121  | ?       | ?         |
| delaunay-n14    | DIM      | 16,384 | 49,122 | 3      | 6.00    | 16     | 4    | 0%     | 65   | ?       | ?         |
| fe-sphere       | DIM      | 16,386 | 49,152 | 4      | 6.00    | 6      | 5    | 0%     | 128  | ?       | ?         |
| cti             | DIM      | 16,840 | 48,232 | 3      | 5.73    | 6      | 4    | 0%     | 64   | ?       | ?         |
| cs4             | DIM      | 22,499 | 43,858 | 2      | 3.90    | 4      | 3    | 0%     | 75   | ?       | ?         |
| delaunay-n15    | DIM      | 32,768 | 98,274 | 3      | 6.00    | 18     | 4    | 0%     | 87   | ?       | ?         |
| hugebubbles     | DIM      | 44,224 | 50,121 | 1      | 2.27    | 3      | 2    | 60%    | 1,065 | 14,424  | 1.05      |
| t60k            | DIM      | 60,005 | 89,440 | 2      | 2.98    | 3      | 2    | 2%     | 649  | ?       | ?         |
| 6015-01         | cDBG     | 108,141| 154,774| 1      | 2.86    | 8      | 4    | 44%    | 315  | ?       | ?         |
| PSM6XBSE        | cDBG     | 219,634| 261,016| 1      | 2.38    | 3      | 6    | 64%    | 2,055| 65,444  | 1.09      |
| HSMA331E        | cDBG     | 486,485| 575,888| 1      | 2.37    | 8      | 3    | 64%    | 2,222| 145,211 | 1.09      |
| ca-HepPh        | collab   | 11,204 | 117,619| 1      | 21.00   | 491    | 238  | 24%    | 13   | 1,662   | 1.51      |
| ca-AstroPh      | collab   | 17,903 | 196,972| 1      | 22.00   | 504    | 56   | 13%    | 14   | 2,055   | 1.74      |
| email-enron-large| email   | 33,696 | 180,811| 1      | 10.73   | 1,383  | 43   | 38%    | 13   | 1,972   | 1.78      |
| inf-luxembourg-osm| DIM   | 114,599| 119,666| 1      | 2.09    | 6      | 2    | 89%    | 1,337| 3,7752  | 1.02      |

Table 2: This table gives details on the network corpus used in experiments, including category (cDBG: metagenomic cDBG, collab: collaboration, social: social networks, DIM: DIMACS10, email), number of vertices ($|G|$), number of edges ($||G||$), minimum degree ($\delta$), average degree ($\bar{d}$), maximum degree ($\Delta$), degeneracy (degn), percentage of the vertices with degree at most 2 ($\%V_2$), diameter of the graph (diam), and the smallest size ($\gamma$)/average congestion (mac) of any dominating set. An entry of "?" indicates Gurobi timed out.