

Approximation Algorithms for Streamed Sparse Graphs

by

Xiuge Chen

Student Number: 961392

ORCID: 0000-0002-2342-5795

Submitted to the

School of Computing and Information Systems

In partial fulfillment of the requirements for the degrees of

Master of Science (Computer Science)

at the

School of Computing and Information Systems

Faculty of Engineering and Information Technology

THE UNIVERSITY OF MELBOURNE

June 2021

Thesis Supervisor: Tony Wirth

Title: Professor at School of Computing and Information Systems

Thesis Co-supervisor: Patrick Eades

Title: Research Fellow at School of Computing and Information Systems

Project Type: Research project

Credit: 75 points

Subjects: COMP90060, COMP90065, COMP90070

Approximation Algorithms for Streamed Sparse Graphs

by

Xiuge Chen

Submitted to the

School of Computing and Information Systems

June 2021

In partial fulfillment of the requirements for the degrees of
Master of Science (Computer Science)

Abstract

Dominating Set and Independent Set are well-known NP-complete problems. Each has been extensively studied in the classical RAM model, but studying them in the data stream model is relatively less developed. In this work, we demonstrate algorithms that approximate these problems or their solution size in streamed sparse graphs.

The Caro-Wei Bound states that for every graph G , the cardinality of maximum independent set is lower-bounded by $\sum_{v \in V(G)} \frac{1}{d(v)+1}$, where $d(v)$ is the degree of v . For graphs with average degree \bar{d} and maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d}+1)^3}$, we show that the Caro-Wei Bound can be $(1 \pm \epsilon)$ -approximated in insertion-only streams using $\mathcal{O}(\bar{d}\epsilon^{-2} \log n \log \delta^{-1})$ space and one pass. The space usage can be further reduced if the stream is vertex-arrival and random-order. Moreover, with a slightly worse approximation ratio and additional space, the maximum-degree constraints can be removed. Significantly, this algorithm can be modified to report an actual independent set in the online streaming model with $\mathcal{O}(\log \epsilon^{-1})$ update time and $\mathcal{O}(\log \epsilon^{-1} \log n \log \delta^{-1})$ working space.

For streamed trees and forests, Chitnis and Krauthgamer showed that the independence number can be $3/2$ -approximated using the number of leaves [84], but no streaming algorithm is given. We show that the number of leaves can be $(1 \pm \epsilon)$ -approximated in one pass and $\text{polylog}(n)$ -space. Also, by including the number of support vertices (vertices that are adjacent to leaves, a notion from discrete mathematics), the approximation ratios can be further improved. Using two passes and $\tilde{\mathcal{O}}(\sqrt{n})$ space, our algorithm can achieve a $2(1 \pm \epsilon)$ approximation to the domination number and a $4/3(1 \pm \epsilon)$ -approximation to the independence number. This technique can similarly be applied to other classic graph problems, such as the matching number and the covering number.

Thesis Supervisor: Tony Wirth

Title: Professor at School of Computing and Information Systems

Thesis Co-supervisor: Patrick Eades

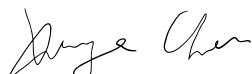
Title: Research Fellow at School of Computing and Information Systems

Declaration of Authorship

I certify that

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the School
- the thesis is 27573 words in length (exclusive of tables, maps, bibliographies and appendices).

Signed:



Date:

2021.06.07

Acknowledgements

First and foremost, I would like to thank my supervisor, Tony Wirth. Tony introduced me to theoretical computer science and helped me realize the beauty and the power of algorithms. He has always been very supportive and has provided many insightful ideas and comments about my work. I appreciate his patience and advice, as well as his enthusiasm and open-mind. It was a pleasure being his research student.

Next, I would like to thank my co-supervisor, Patrick Eades for his support, patience, encouragement, and valuable advice. This has helped me improve my research, and has kept my passion for algorithms alive.

I would also like to thank Rajesh Chitnis and Robert Krauthgamer for pointing out the research gap in studying streamed sparse graphs and kindly sharing their notes with me. The quality of my research is considerably enhanced by the enlightening discussions with Rajesh.

Being part of the theory group, I met many inspiring and fascinating people – Hao Wu, William Holland, Xin Zhang, Peaker Guo, Philip Cervenjak, and those whose names I cannot remember at the moment. I benefit a lot from our discussions, and I really enjoy our weekly theory talk and reading group.

I would like to thank Tilman Dingler and Vassilis Kostakos for introducing me to Human-Computer Interaction and recruiting me as a research assistant. It was a wonderful experience being part of the lab, and I greatly appreciate your support, guidance, and help. I would also like to thank my coauthor – Chaofan Wang and Zhanna Sarsenbayeva; my collaborator – Benjamin Tag, Namrata Srivastava, Rajiv Jain, and Jennifer Healey; as well as others – Allen Mari Pilaes, Difeng Yu, Qiushi Zhou.

I am also very grateful for a few fantastic working experiences during my study. I thank Renata Borovica-Gajic, Farah Khan, Olya Ohrimenko, Lachlan Andrew, Peter Schachte, and Michelle Blom for recruiting me as one of your teaching assistants. I had a great time learning and teaching your subjects. I also thank Optiver for providing an internship opportunity and a returned offer to me.

I would like to thank all my friends at Guiyang, Nanjing University and University of Melbourne for the wonderful friendship and memories we had together.

I would also like to thank my families for their unconditional love and support, including my parents, Jun Yan and Hong Chen, and my grandparents, Zongfeng Wang and Xijin Yan, as well as other family members. They have set a great example for me, gave me confidence, and helped me become who I am.

Finally, I would like to thank my partner, Yihan Sun, for being the most amazing person in my life. Words cannot describe how thankful and how lucky I am to have her. I very much appreciate everything she did for me.

Contents

Abstract	i
Declaration of Authorship	ii
Acknowledgements	iii
List of Figures	viii
List of Tables	ix
Abbreviations	x
Symbols	xi
1 Introduction	1
1.1 Dominating Set and Independent Set	1
1.2 Computing <i>MDS</i> and <i>MIS</i>	3
1.3 The Data Stream Model	6
1.4 Streaming <i>MDS</i> and <i>MIS</i>	7
1.5 Research Questions	10
1.6 Our Contributions	10
1.7 Organizations	14
2 Preliminaries	16
2.1 Problem Definitions	16
2.1.1 Graph Definitions	16
2.1.2 Graph Problems	17
2.1.3 Randomized and Approximation Algorithms	17
2.1.4 Complexity Notations	18
2.2 Probability Theory	19
2.3 Hash Functions	21
2.4 Streaming Models	23
2.5 Elementary Streaming Algorithms	25
3 Further Related Works	28
3.1 Dominating Set	28
3.1.1 Approximating the Dominating Set	28

3.1.2	Computing the Domination Number	30
3.1.2.1	Upper Bounds of Domination Number	30
3.1.2.2	Lower Bounds of Domination Number	32
3.1.2.3	Approximating the Domination Number	32
3.1.2.4	Bounds of Connected Domination Number	33
3.2	Streaming Dominating Set	34
3.3	Independent Set	35
3.3.1	Approximating the Independent Set	35
3.3.2	Computing the Independent Number	36
3.3.2.1	Upper bounds of Independence Number	36
3.3.2.2	Lower bounds of Independence Number	36
3.4	Streaming Independent Set	38
4	Streaming Dominating Set	40
4.1	Tree Approximation	40
4.1.1	3-approximation of Domination Number	41
4.1.2	2-approximation of Domination Number	41
4.1.3	Exact Estimate of Connected Domination Number	48
4.2	Tree Streaming Algorithms	48
4.2.1	Estimating the Number of Non-leaf Vertices	49
4.2.2	Estimating the Number of Support Vertices	50
4.2.2.1	Two-pass $(1 \pm \epsilon)$ Approximation	50
4.2.2.2	Two-pass Exact Estimate	53
4.2.3	Estimating Domination Number	56
4.3	Hardness Results	58
5	Streaming Independent Set	60
5.1	Sparse Graph Streaming Algorithms	61
5.1.1	An Edge-Arrival Algorithm for Independence Number	61
5.1.2	Modifications to <i>Online Streaming Model</i>	70
5.1.3	Removing the Maximum Degree Constraint	72
5.1.4	Vertex-Arrival Random-Order Algorithm	76
5.2	Tree Approximation	77
5.2.1	3/2 Approximation of Independence Number	77
5.2.2	4/3 Approximation of Independence Number	78
5.3	Tree Streaming Algorithms	82
5.3.1	Estimating the Number of Leaves	82
5.3.2	Estimating Independence Number	85
5.4	Hardness Results	87
6	Other Streaming Graph Problems	89
6.1	Trees Approximation	90
6.1.1	2 Approximation of Matching and Vertex Cover	90
6.1.2	3/2 Approximation of Matching and Vertex Cover	90
6.2	Tree Streaming Algorithms	95
6.2.1	Estimating the Matching Number and the Covering Number	95
6.3	Hardness Results	96

7	Conclusions and Future Work	97
7.1	Conclusions	97
7.2	Future Work	99
	Bibliography	100

List of Figures

1.1	Example of dominating set and independent set	2
1.2	Counter example for the independent set greedy algorithm	5
2.1	Example of graph streams arrival order	23
4.1	Sharp example of our upper bound on tree domination number	43
4.2	Tight example of our lower bound on tree domination number	46
6.1	Sharp example of our upper bound on tree matching number	92
6.2	Sharp example of our upper bound on tree matching number	94

List of Tables

1.1	Streaming Caro-Wei Bound approximation in graphs with average degree \bar{d}	12
1.2	Summary of streaming tree and forest domination number approximation	13
1.3	Summary of streaming forest independence number approximation	14
1.4	Summary of streaming forest matching/covering number approximation .	14

Abbreviations

<i>DS</i>	Dominating Set
<i>MDS</i>	Minimum Dominating Set
<i>CDS</i>	Connected Dominating Set
<i>MCDS</i>	Minimum Connected Dominating Set
<i>IS</i>	Independent Set
<i>MIS</i>	Maximum Independent Set
<i>Mat</i>	Matching
<i>MMat</i>	Maximum Matching
<i>VC</i>	Vertex Cover
<i>MVC</i>	Minimum Vertex Cover

Symbols

$V(G)$ or V	the vertex set of graph G
$E(G)$ or E	the edge set of graph G
$n(G)$ or n	the number of vertices of graph G
$m(G)$ or m	the number of edges of graph G
$\Delta(G)$ or Δ	the maximum vertex degree of graph G
$\delta(G)$ or δ	the minimum vertex degree of graph G
	the fail rate of a randomized algorithm
$\bar{d}(G)$ or \bar{d}	the average degree of graph G
$d(v)$	the degree of vertex v
$N(S)$	the set of vertices that are neighbours of vertices in S (excluding S)
$N[S]$	the union of S and $N(S)$
$Deg_i(G)$ or Deg_i	the set of G 's vertices with degree i
$H_i(G)$ or H_i	the set of G 's vertices with degree at least i
$S(G)$ or S	the set of G 's support vertices
$\gamma(G)$ or γ	G 's domination number
$\gamma_C(G)$ or γ_C	G 's connected domination number
$\beta(G)$ or β	G 's independence number
$\phi(G)$ or ϕ	G 's matching number
$\tau(G)$ or τ	G 's covering number
$E[X]$	Mean of a random variable X
$Var(X)$	Variance of a random variable X

Chapter 1

Introduction

1.1 Dominating Set and Independent Set

The dominating set problem and the independent set problem are two well-known combinatorial optimization problems on graphs. Given a graph, both problems can be viewed as selecting a subset of its vertices set that satisfies certain properties. To illustrate, we say a vertex can cover another vertex if there is an edge between them, and two vertices are independent of each other if there is no edge between them. With this definition, a subset of vertices is a *dominating set* if they can cover all other remaining vertices in the same graph. And a subset of vertices is an *independent set* if every pair of vertices in this subset are independent of each other. Trivially, every vertex itself is an independent set, and the full vertex set is a dominating set. Typically, instead of finding just one valid solution, we are more interested in finding the optimal one. That is, the independent set with the largest size (i.e., maximum independent set *MIS*), and the dominating set with the smallest size (i.e., minimum dominating set, *MDS*). And we use domination number (γ) and independence number (β) to denote the size of the optimal solution to these two problems.

Take the graph in Figure 1.1 as an example. In this graph, vertices *E* and *F* form a minimum dominating set, as all of the other vertices are dominated/covered by them and there does not exist a smaller dominating set. Also, the set of vertices *A*, *C*, *D*, and *G* is the maximum independent set, as there are no edges among them and there does not exist a larger independent set.

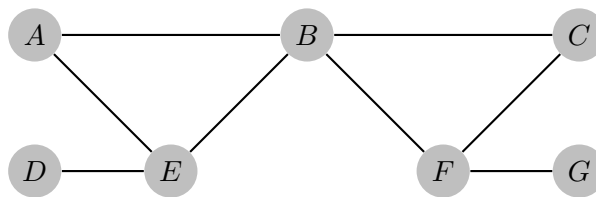


FIGURE 1.1: Example of dominating set and independent set

The origin of the dominating set and the independent set problem can be traced back to the k -Queens problem in the 18th century, which asks whether we could place k chess queens on a $k \times k$ chessboard so that no two queens are attacking each other. Later in 1957, Berge [12] formally introduced both problems as “cover” and “internally stable set”¹. Since then, these two problems have drawn lots of attention not only because of their wide range of applications, but also their close connections with other graph problems.

The dominating set problem and its variants are very suitable to describe problems where the main goal is to select a group of supervisors/guardians, or a group of sources for information dissemination and interaction. For example, in large wireless sensor networks, it is very energy-consuming to keep all sensors activated. And in reality, sensors do not have to be active at all times, keeping every sensor turned on will unnecessarily waste lots of energy. To resolve this, one can let only a small set of nodes be “awake” (i.e., responsible for sensing and gathering data) and make other nodes “sleep” [107]. By doing this, most of sensors are off during waiting time, thus it helps significantly reduce the energy consumption. Moreover, in order to be able to notify all the sleeping nodes when some jobs arrive, the chosen awake nodes must form a dominating set. Other applications of the dominating set include modeling routing and broadcasting problems in multi-hop ad-hoc networks [28, 120, 124], addressing the controllability of networks [98, 124], describing protein interactions in biological networks [96], and generating summarization for multi-document in information retrieval [111].

Similarly, the independent set problem has many applications in optimization and scheduling, especially when the “conflicts” or the “collisions” among elements should be avoided. For example, automatic label placement is the task of automatically placing texts or labels on a diagram (e.g., a map or a chart), and we do not want the placed texts overlapping with each other. This problem can be easily modelled as an independent

¹Some works have also used “externally stable set” to refer to the dominating set

set problem, where each node represent an available placement, and two nodes share an edge if they overlap with each other [56]. By utilizing the maximum independent set algorithm, we can find the maximum set of texts to be placed without interfering with each other. Other applications of the independent set route/path planning [80], detecting collisions in voting pools [3], and identifying non-repetitive genetic sequences in genetic systems design [68].

Besides its wide applications, the independent set problem is closely related to other graph problems. For instance, a subset of vertices is a vertex cover (or hitting set) if for every edge, at least one of its endpoints is in the set. In Figure 1.1, for example, the set $\{B, E, F\}$ is the minimum possible vertex cover. It is known that the vertex complement of the independent set (i.e., the set of vertices that are not in the independent set) is itself a vertex cover. To illustrate, assuming there is an edge that is not covered by vertices other than the independent set, then this edge must have both endpoints in the independent set, violating its definition, hence a contradiction. Also, a matching of a graph is a subset of its edges, such that none of them share a common vertex. For example, in Figure 1.1, there are multiple matchings of size 3 (e.g. edges (D, E) , (F, G) , and (A, B)). Denote the number of vertices in the graph by n , the size of the maximum matching by ϕ , and the size of maximum independent set by β . It is known that $n - \phi \geq \beta \geq n - 2\phi$. The lower bound can be proved by taking a maximum matching and adding all vertices that are not in the matching as an independent set. And the upper bound can be proved via arguing that no more than ϕ vertices in the maximum matching can be in the independent set, hence we have $\beta \leq \phi + (n - 2\phi) = n - \phi$.

1.2 Computing *MDS* and *MIS*

The minimum dominating set problem (*MDS*) and the maximum independent set problem (*MIS*) have been extensively studied in the classical random-access machine (RAM) model, where the whole graph is stored in the memory and the primary resource of concern is time. Given an arbitrary input $k > 0$, deciding whether there exists a dominating set of size at most k in general graphs is NP-complete [79]. Finding the minimum dominating set and computing the domination number in general graphs cannot be easier than this decision problem, hence they are both NP-hard. And their hardness even

hold in planar graphs with maximum degree of three [55] and in unit disk graphs² [34]. Similarly, in general graphs, given an arbitrary input $k > 0$, deciding whether there is a maximum independent set of size at least k is also NP-complete [79]. Thus, finding the maximum independent set and computing the independence number are both NP-hard, and it is even hard to approximate them within factor of $n^{1-\epsilon}$, for any $\epsilon > 0$ [62, 125]. Currently, the best brute-force algorithm can compute the *MIS* in $\mathcal{O}(1.1996^n)$ time [122], and the exponential base can be further reduced (but remains greater than 1) if the maximum degree is bounded by a constant [121, 122]. Hence, researchers have been seeking efficient algorithms under compromises, such as computing approximate results, restricting the input to specific graph classes, or establishing general upper/lower bounds on the domination/independence number.

Like many optimization problems, *MDS* and *MIS* can be well approximated via simple, but powerful greedy algorithms. For the dominating set, the greedy algorithm iteratively picks the node that dominates the most undominated nodes, and removes it and all of its edges from the graph. It can achieve an $(H_\Delta + 1) \leq (\ln \Delta + 2)$ approximation in every graph, where Δ is the maximum vertex degree and H_Δ is the Δ -th harmonic number [73, 91]. This approximation ratio is nearly optimal: as shown by reduction from the set cover problem, it is NP-hard to achieve an approximation ratio better than $(1 - \epsilon) \ln \Delta$ for every $\epsilon > 0$ [43, 78]. To see why the greedy algorithm fails to find an optimal solution, consider Figure 1.1 above. In this graph, the minimum dominating set has size two (E and F), while the greedy algorithm outputs a dominating set of size 3 (B , E , and F). For actual graph examples where the analysis of the greedy algorithm is tight (i.e. outputs an $\mathcal{O}(\ln \Delta)$ approximation), see Example 1.1 and Example 1.2 in Li et al. [88].

As for the independent set, its greedy algorithm also constructs a solution iteratively until there are no vertices left. In each iteration, it picks the vertex with the smallest degree and removes all of its neighbors. Halldórsson and Radhakrishnan [58] have shown that this greedy algorithm can achieve an approximation ratio of $\frac{\Delta+2}{3}$, where Δ is the maximum vertex degree. Figure 1.2 is an example of why the greedy algorithm might fail to output the optimal solution. In the first iteration, the algorithm might choose either one of B , C , D , F . If it chooses B , D , or F , it outputs the optimal solution

²The intersection graph of unit disks in the Euclidean plane, where each node is a unit disk, and two nodes share an edge if they intersect with each other

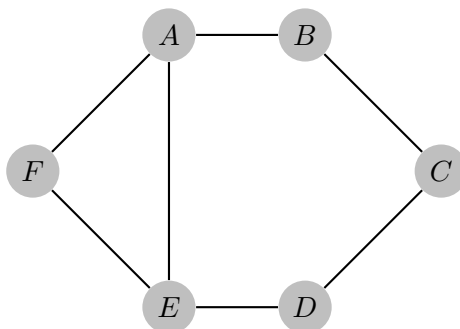


FIGURE 1.2: Counter example for the independent set greedy algorithm

(which has size 3). However, if it chooses C , then its output solution is at most of size 2.

In the real world, most of graphs are sparse (i.e., have $\mathcal{O}(n)$ edges or constant average degree) [109]. In these relatively sparse graphs, we can approximate these two problems with approximation ratios proportional to their average degree. For instance, in graphs with bounded arboricity³, α , an $2\alpha - 1$ approximation of dominating set can be achieved based on solving its relaxed linear program [10, 46]. Also, it is known that for a graph G , its independence number $\beta \geq \sum_{v \in V(G)} \frac{1}{1+d(v)} \geq \frac{n}{\bar{d}+1}$, where $d(v)$ is the degree of vertex v and \bar{d} is the average degree of G . The former term (i.e., $\beta \geq \sum_{v \in V(G)} \frac{1}{1+d(v)}$) is known as the Caro-Wei Bound [29, 118], and the later term (i.e., $\beta \geq \frac{n}{\bar{d}+1}$) is known as the Turán Bound [115]. Clearly, either bound gives a constant approximation for the independence number when the average degree is bounded by a constant (i.e., when the graph is relatively sparse). It is worthwhile to mention that there is a greedy algorithm that outputs an independent set of size at least the Caro-Wei Bound in expectation. Given a permutation (i.e., ordering) of the vertices, this permutation-based greedy algorithm adds a vertex to the independent set if it has the smallest order among it and its neighbours. By choosing the permutation randomly, each vertex has the smallest order with probability $\frac{1}{d_v+1}$, one can then show that the expected size of the chosen independent set is exactly the Caro-Wei Bound. Moreover, by utilizing the results of semi-definite programming, the approximation ratio of the independent set can be improved to $\tilde{\mathcal{O}}(\frac{\bar{d}}{\log^2 \bar{d}})$ [11], where \bar{d} is the average degree and we use $\tilde{\mathcal{O}}$ to suppress the logarithmic factors. This result has an almost matching lower bound, it is NP-hard to achieve an approximation ratio of $\mathcal{O}(\frac{\Delta}{\log^4 \Delta})$.

³The minimum number of forests that this graph can be decomposed into

The approximation ratio can be further improved in even sparser graphs. For instance, the minimum dominating set has a polynomial-time approximation scheme (PTAS) in planar graphs [7], unit disk graphs [69], and graphs with polynomial expansion [60]. That is, the minimum dominating set can be $1 - \epsilon$ approximated in these graphs, for every small constant ϵ . Similarly, a PTAS has been found for finding the maximum independent set. Moreover, in trees or forests, one can compute the dominating set and the independent set exactly in linear time using dynamic programming. Briefly speaking, we recursively examine the nodes in the tree and construct our solution. For example, in the dominating set problem, for each node in the tree, we need to consider whether it is already dominated or not. If not, we need to either choose itself or choose one of its children (if available) to cover it. Similarly, in the independent set problem, for each node, we can choose to add it (if its parent is not in the independent set) or not add it to the independent set. Because there are no cycles in trees, this strategy is guaranteed to terminate and output the optimal dominating/independent set.

1.3 The Data Stream Model

With the evolution of information technology, massive graphs with enormous amount of nodes and edges have been generated (e.g., webpages, wireless networks, and protein networks). For example, currently, the number of nodes in a hyperlink web graph can be of the order 2×10^9 [109]. The traditional offline algorithms described above can not be used to analyze such huge graphs when limited computational space available (e.g., in a sensor), as they all require storing the whole graphs in memory. This challenge has motivated studies under many new computational models, such as the data stream model and the distributed model. In the data stream model, the universe of data elements is known in advance, and data elements arrive sequentially in arbitrary order. In the case of graphs, each data element could be either a single vertex together with all its neighbors that arrived before (i.e., the vertex-arrival model), or a single edge (i.e., the edge-arrival model). Also, if deletions of previously arrived vertices/edges are allowed, then it is a turnstile stream. Otherwise, we refer it as a insertion-only stream ⁴.

⁴If pre-deletions of vertices/edges (i.e., deleting a vertex/edge that has not appeared before), it is a dynamic stream

The data stream model receives much attention during the last two decades for several reasons. Firstly, it nicely captures the natural ways of receiving data in the real world. For example, the streaming processing of data might occur when monitoring data traffic, or receiving partial data from multiple clients. Secondly, the data stream model helps address the situation where we want to perform some computations but only have limited space resource available. This is because in the data stream model, the primary resource of concern is space rather than time. Typically, the space available is much less than the space required to store the whole stream. For example, consider the problem of finding the number of distinct packages that pass through a sensor, the trivial solution would be maintaining a hash map to store packages. However, it becomes very challenging when there are huge amount of different packages passing through, as a sensor usually does not have a sufficiently large memory to store everything. For most problems, their streaming variants only allow $o(n)$ bits of memory. This restriction is relaxed to $\mathcal{O}(n \log n)$ for some graph problems as a general graph can cost $\mathcal{O}(n^2 \log n)$ bits to store all of its edges, which is also known as the semi-streaming model [52]. However, for sparse graphs with a linear number of edges, semi-streaming model actually allows us to store the whole graph, hence it is more reasonable to limit the space usage to $o(n)$. Moreover, the algorithms developed under the data stream model are also very useful when an approximate answer or occasional incorrect output is acceptable. Many industries have already benefited from using streaming techniques to help significantly reduce their time and space cost, see [1] for a public software library and its users (e.g., Yahoo, Druid, etc.)

The study of streaming graph problems was initiated by Henzinger et al. [65] who study following paths and testing connectivity. In the last decade, many graph problems have been extensively studied in the streaming model, including matching [26, 32, 33, 39, 49], and vertex cover [6, 31–33, 94]. See [93] for a comprehensive survey.

1.4 Streaming *MDS* and *MIS*

The size of a dominating set or an independent set can be as large as $\Theta(n)$ (consider a path of length n). If we restrict the space usage to be sublinear in the number of vertices (i.e., $o(n)$), we often cannot store the entire solution, hence making all algorithms infeasible. Therefore, when only sublinear space is available, we are often interested

in just finding the size of the optimal solution (i.e., the domination number and the independence number), rather than the actual solution. The optimal size approximation has been extensively studied in matching [26, 39, 49]. However, very little research has been done in the streaming domination number and independence number. Generally, current algorithms for both problems can be summarized into two types, based on the techniques they have used.

On the one hand, people have relaxed the stream order from arbitrary to random, and studied techniques of converting known constant-query graph property testing and constant-time approximation algorithms into constant-space streaming algorithms. Based on this, Monemizadeh et al. [97] designed a streaming algorithm that approximated the domination number in bounded-degree graphs, with an additive error term of ϵn . Their algorithm relied on approximating the distributions of the k -disc of the graph, the subgraph induced by a vertex and its neighbors that are at most k away. Peng and Sohler [106] further extended this result to general graphs and graphs with bounded average degree (e.g., planar graphs). In the same paper, Peng and Sohler [106] showed that the independence number in planar graphs can be approximated within a multiplicative factor of $1 + \epsilon$ using only a constant amount of space. However, it is worth mentioning that the constant in space is very large, i.e., $\mathcal{O}(2^{(1/\epsilon)^{(1/\epsilon)^{\log^{\mathcal{O}(1)}(1/\epsilon)}}})$. One crucial fact they have relied on is that β in planar graphs is $\Theta(n)$, so that it is fine to ignore the vertices with degree higher than $\Theta(1/\epsilon)$ as there are at most $\Theta(\epsilon n)$ of them.

On the other hand, some works have focused on approximating well-established upper and lower bounds in the streaming settings. Halldórsson et al. [57] studied general hypergraphs and gave a one-pass randomized insertion-only streaming algorithm, which uses $\mathcal{O}(n)$ space and outputs an independent set with size no less than the Caro-Wei bound in expectation. Their core algorithm is based on the fact that given a independent set I , $V \setminus I$ is a hitting set (or a vertex cover). Hence, for every hyper-edge, assuming we have a priority function f that maps each vertex into a natural number $\in \mathbb{N}$, we could add the vertex with largest value to form a hitting set, and the remaining graph itself is a independent set. A trivial choice of the function f would be a random permutation of the vertex set, which can be stored in $\mathcal{O}(n \log n)$ space. By choosing f carefully such that vertices are mapped to $\log n$ levels with exponentially decreasing probability (i.e., a vertex is mapped to the i th level with probability $\frac{1}{2^i}$), the space usage could be further reduced to $\mathcal{O}(n)$. Moreover, their algorithm can be applied to an even stricter model, the

online streaming model, which combines the properties of data streams model and online model. In online streaming model, elements (i.e., edges) arrive one by one in arbitrary order, the algorithm must maintain a valid solution at any given time of the stream, and every decision is irrevocable. For example, in online streaming independent set, initially the solution is the set of all vertices. As an edge arrives, the algorithm must decide which vertex should be removed from the solution to maintain a valid independent set. Once a vertex is removed from the solution, it cannot be added back later. The solution can be stored locally, or can be stored in a remote server such that the algorithm “reports” to the server about its decisions.

Moreover, it is not hard to see the Caro-Wei Bound is very similar to the -1 frequency moment⁵ (or the harmonic mean) of the degree vector. Although $(1 \pm \epsilon)$ -approximating the -1 frequency moment in general graphs requires $\Omega(\epsilon^{-1/2}n)$ bits [18], it is achievable in sublinear space when the value of the frequency moment is sufficiently large. Hence, Cormode et al. [40] designed an algorithm that $(1 \pm \epsilon)$ -approximates the Caro-Wei Bound with constant success probability using $\mathcal{O}(\frac{n \log n}{\lambda_L \epsilon^2})$ space, where λ_L is a known lower bound of the Caro-Wei Bound. By letting $\lambda_L = \frac{n}{d+1}$ (the Turán Bound), the space usage becomes $\mathcal{O}(\epsilon^{-2} \bar{d} \log n)$. In the algorithm, they sampled vertices with probability $p = \frac{3}{\epsilon^2 \lambda_L}$ beforehand and record the degree of the sampled vertices during the stream. They return $\frac{1}{p} \sum_{v \in S} \frac{1}{d(v)+1}$ as their estimate of the Caro-Wei Bound, where S is the set of sampled vertices. By simple analysis, one can show that the returned estimate is an unbiased estimate of the Caro-Wei Bound and its variance is bounded by $1/p$ of the Caro-Wei Bound.

As for the lower bound, Chitnis and Cormode [31] adapted the lower-bound reduction technique for the set cover problem [5] and demonstrated that for graphs with arboricity $\beta + 2$ for $\beta \geq 1$, any randomized $\frac{\beta}{32}$ -approximation algorithm for the minimum dominating set problem would require $\Omega(n)$ space. This lower bound even holds under the vertex arrival model. Cormode et al. [40] have proved a lower bound for approximating the Caro-Wei Bound. They showed that every randomized algorithm with constant error probability would require $\Omega(\frac{n}{\lambda_L \epsilon^2 p})$ space to ϵ -approximate the Caro-Wei Bound, where λ_L is a known lower bound of the Caro-Wei Bound and p is the number of passes allowed. This implies a $\Omega(\frac{\bar{d}}{\epsilon^2})$ space lower bound for every one-pass algorithm that approximates the independence number with a ratio of $\bar{d}(1 \pm \epsilon)$.

⁵The sum of the reciprocal of each vertex’s degree, i.e., $\sum_{v \in V(G)} \frac{1}{d(v)}$

1.5 Research Questions

Although results have been obtained for both the dominating set and the independent set in sparse streamed graphs, there are still some gaps between the upper bounds and the lower bounds.

For dominating set, no upper bound has been established for graphs with bounded arboricity α , and the lower bound from [31] does not rule out the possibility of $\mathcal{O}(\alpha)$ approximation algorithms, which can be achieved in offline algorithms [10, 46]. Besides, the current best result for bounded-degree graphs [97] only works in randomly ordered streams and its error rate is ϵn . Can we achieve the same approximation (or even better) in arbitrary ordered streams? Lastly, no research has been done to study streamed graphs that are even sparser, such as the planar graphs and trees.

For independent set, although positive results have been obtained on approximating the Caro-Wei Bound [40, 57], there is still a $\log n$ gap between the upper bound and the lower bound. Moreover, there are still several remaining problems on even sparser streamed graphs (like planar graphs and trees), which either costs constant but enormously large space [106], or has not been studied before.

Hence, we propose our research question as follows.

Given sub-linear space and constant-number passes of a sparse graph stream, can we design streaming algorithms that approximate the domination number and independence number with better ratio? If so, what is the best approximation ratio we can achieve for different graph classes and space classes?

1.6 Our Contributions

In this work, we have studied algorithms for approximating various graph optimization problems (e.g., the independence number) in streamed sparse graphs, especially in graphs with low average degree (including graphs with bounded arboricity and degeneracy, as well as planar graphs) and streamed forests.

For streamed graphs with average degree \bar{d} and maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d} + 1)^3}$ ⁶, our

⁶which is $\mathcal{O}(n)$ if \bar{d} and ϵ are constants

algorithm can $(1 \pm \epsilon)$ -approximate the Caro-Wei bound in small space. Since the Caro-Wei Bound is at least the Turán Bound ⁷, our result gives a $\bar{d}(1 \pm \epsilon)$ -approximation of the independence number. Our algorithm fails with probability at most δ , and its space usage is $\mathcal{O}(\bar{d}\epsilon^{-2} \log n \log \delta^{-1})$ in arbitrary-order edge-arrival streams, $\mathcal{O}(\log(\bar{d}\epsilon^{-2}) \log \delta^{-1})$ in random-order vertex-arrival streams. It simulates the permutation-based greedy algorithm for independent set in data streams with the help of ϵ -min wise hash function. Recall that given a random vertex permutation, the greedy algorithm outputs an independent set of size at least the Caro-Wei Bound in expectation, but storing such permutation would require $\Omega(n \log n)$ bits. An ϵ -min wise hash function can mimic a random permutation in small space as it maps input elements to values $\in \mathbb{R}$, such that for each element, the probability that it has the smallest hash value across a set of elements, A , is $\frac{1 \pm \epsilon}{|A|}$. However, the error rate of this algorithm cannot be bounded in general graphs because of the positive correlation between a vertex and its neighbours' neighbours. To illustrate, consider a path with vertices v, u, w and edges $(v, u), (u, w)$. If v has smaller hash value than u , than it provides a lower bound on u 's hash value. Hence, w is more likely to be smaller than u as well. We show that, this positive correlation can be bounded if the maximum degree is not too large.

In addition, our algorithm can be modified to report an actual independent set with fast update time (i.e., time used to process each edge) and small working space (i.e., space other than used to store the solution) in the online streaming model. Also, the maximum degree limitation in our edge-arrival algorithm can be removed using the heavy hitter sketch and an extra post-processing step. This is because according to the Turán Bound, the independence number is at least $\frac{n}{\bar{d}+1}$. Hence, the independence number of the subgraph after removing all high-degree vertices (i.e., vertices with degree higher than $\frac{\epsilon^2 n}{3(\bar{d}+1)^3}$) is still within ϵ away from the original independence number. Therefore, it is sufficient to use heavy hitter sketch to obtain and remove all high-degree vertices, and estimate the independence number of the subgraph. The space usage of the new algorithm is $\mathcal{O}(\bar{d}^4 \epsilon^{-2} \log^2 n)$ (when failing with low probability), and the returned results is slightly biased (i.e., $(1 - \epsilon)(1 \pm \epsilon)$ -approximation). Its update time is $\mathcal{O}(\log n \log(\frac{\bar{d}^4 n \log n}{\epsilon^2}))$ and its post-processing time is $\mathcal{O}(\bar{d}^5 \epsilon^{-4})$. Note that since post-processing is required, the new algorithm cannot be used in online streaming model.

⁷See [17] for a simple proof

Difference between our results and the results in [40, 57]: Halldórsson et al. [57] gave an online streaming algorithm that outputs an actual independent set of size at least the Caro-Wei Bound in expectation. Their algorithm has update time $\log n$ and working space usage $\mathcal{O}(n)$. Cormode et al. [40] designed an turnstile algorithm that $(1 \pm \epsilon)$ -approximate the Caro-Wei Bound with probability at least $1 - \delta$ using $\mathcal{O}(\bar{d}\epsilon^{-2} \log n \log \delta^{-1})$ space. However, it cannot report an actual set. Our algorithm has advantages from both works. Firstly, when the stream is insertion-only and the maximum degree is not too large, our algorithm achieves asymptotically the same approximation ratio as [40] using asymptotically the same space. If we restrict the stream to be vertex-arrival and random-order, the space usage can be further reduced. Moreover, it can be modified to the online streaming model with faster update time ($\mathcal{O}(\log 1/\epsilon)$) and less extra working space ($\mathcal{O}(\log \epsilon^{-1} \log n \log \delta^{-1})$). And both its error rate and its fail rate are bounded by our analysis.

Our results for Caro-Wei Bound approximation in streamed sparse graphs with bounded average degree are summarized in Table 1.1. The “Stream” column indicates the stream type, we use *Edge* for edge-arrival model and *Vertex* for vertex-arrival model. Also, we use *Turn.* and *Ins.* to indicate the turnstile and the insertion-only streams, respectively. The “Order” column specifies the stream order, where *Arb.* stands for arbitrary-order streams, and *Ran.* stands for random-order streams. “Approx.”, “Update”, and “Space” columns contains the approximation ratio, update time and space usage of each algorithm. The “Online” column has value “Yes” if the algorithm fits into the online streaming model. All algorithms in Table 1.1 have constant success probability, except result on the first row does not have error bound, and result on the fifth row has high success probability (i.e., $1 - n^{-c}$ for some constants c). And rows marked with * require the maximum vertex degree to be no more than $\frac{\epsilon^2 n}{3(\bar{d}+1)^3}$.

Stream	Order	Approx.	Update	Space	Online	Ref
Edge Ins.	Arb.	<i>exp.</i>	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	Yes	[57]
Edge Turn.	Arb.	$(1 \pm \epsilon)$	$\mathcal{O}(1)$	$\mathcal{O}(\bar{d}\epsilon^{-2} \log n)$	No	[40]
Edge Ins.	Arb.	$(1 \pm \epsilon)$	$\mathcal{O}(\log \epsilon^{-1})$	$\mathcal{O}(\bar{d}\epsilon^{-2} \log n)$	No	This*
Edge Ins.	Arb.	$(1 \pm \epsilon)$	$\mathcal{O}(\log \epsilon^{-1})$	$\mathcal{O}(\log \epsilon^{-1} \log n)$	Yes	This*
Edge Ins.	Arb.	$(1 - \epsilon)(1 \pm \epsilon)$	$\mathcal{O}(\log n \log(\frac{\bar{d}^4 n \log n}{\epsilon^2}))$	$\mathcal{O}(\bar{d}^2 \epsilon^{-2} \log^2 n)$	Yes	This
Vertex Ins.	Ran.	$(1 \pm \epsilon)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(\bar{d}\epsilon^{-2}))$	No	This*
Any	Any	c	-	$\Omega(\frac{\bar{d}}{\epsilon^2})$	-	[40]

TABLE 1.1: Streaming Caro-Wei Bound approximation in graphs with average degree \bar{d}

For a streamed forest, we introduce the idea of support vertices in approximating the domination number, the independence number, as well as other classical graph problems (i.e., the matching number and the covering number). A vertex is a support vertex if it is adjacent to (share an edge with) at least one leaf. It is known that in trees, by knowing the cardinality of non-leaf vertices or the leaves, one can 3-approximate the domination number [47, 84, 86, 95], 3/2-approximate the independence number [84], and 2-approximate of the matching number as well as the covering number [25, 49]. We show that, with additional knowledge about the number of support vertices, we could improve the approximation ratio to 2 for domination number, to 4/3 for independence number, and to 3/2 for matching number and covering number.

We also present $\text{poly}(\log n)$ -space streaming algorithms that $(1 \pm \epsilon)$ -approximate the size of leaves and the size of non-leaf vertices. Moreover, we show that the number of support vertices can be $(1 \pm \epsilon)$ -approximated using 2 passes and $\tilde{O}(\sqrt{n})$ space when it is large. When the number of support vertices is small, either the number of non-leaf vertices is also small (so that we can estimate both of them accurately), or the number of non-leaf vertices is too large such that there is no need to use support vertices. We remark that the cardinality of the three quantities we used (i.e., support vertices, leaves, and non-leaf vertices) can be easily estimated in other models. For example, in distributed settings, every vertex can exchange information with its neighbours at each round, a graph problem is solved via analysing and integrating the information obtained at each node across certain number of rounds. We could use one round to check whether the vertex is a leaf or not, and another round to check whether it is a support vertex or not. Each communication round can be done using only $\mathcal{O}(1)$ message size.

Our results for streamed forests are summarized in Table 1.2, Table 1.3, and Table 1.4 respectively. In these tables, we use the same notations as the Table 1.1.

For domination number and connected domination number approximation, our result and other relevant studies are summarized in Table 1.2.

Prob.	Graph	Stream	Order	Pass	Approx.	Space	Citation
γ	Tree	Edge Turn.	Arb.	1	$3(1 \pm \epsilon)$	$\text{poly}(\log n)$	[84]
	Forest	Edge Turn.	Arb.	2	$2(1 \pm \epsilon)$	$\tilde{O}(\sqrt{n})$	This
	Forest	Any	Any	1	$3/2 - \epsilon$	$\Omega(\sqrt{n})$	[49]
γ_C	Tree	Edge Turn.	Arb.	1	$(1 \pm \epsilon)$	$\text{poly}(\log n)$	This

TABLE 1.2: Summary of streaming tree and forest domination number approximation

And for independence number, Table 1.3 compares our results with other studies.

Graph	Stream	Order	Pass	Approx.	Space	Citation
Forest	Edge Turn.	Arb.	1	$3/2(1 \pm \epsilon)$	$poly(\log n)$	[84], This
	Edge Turn.	Arb.	2	$4/3(1 \pm \epsilon)$	$\tilde{O}(\sqrt{n})$	This
	Any	Any	1	$4/3 - \epsilon$	$\Omega(\sqrt{n})$	[49]

TABLE 1.3: Summary of streaming forest independence number approximation

Lastly, for both tree/forest matching and vertex cover, Table 1.4 summarizes our result in comparison with other related results.

Graph	Stream	Order	Pass	Approx.	Space	Citation
Forest	Edge Ins.	Arb.	1	$2(1 \pm \epsilon)$	$\tilde{O}(\sqrt{n})$	[49]
	Edge Ins.	Arb.	1	$2(1 \pm \epsilon)$	$poly(\log n)$	[39]
	Edge Turn.	Arb.	1	$2(1 \pm \epsilon)$	$poly(\log n)$	[25]
	Edge Turn.	Arb.	2	$3/2(1 \pm \epsilon)$	$\tilde{O}(\sqrt{n})$	This
	Any	Any	1	$3/2 - \epsilon$	$\Omega(\sqrt{n})$	[49]

TABLE 1.4: Summary of streaming forest matching/covering number approximation

1.7 Organizations

In Chapter 2, we introduce some preliminaries, including definitions about the graphs and the problems, as well as some fundamental streaming and probability tools that are used in our algorithm.

Chapter 3 is about some other related works, including typical offline algorithms for computing the optimal dominating and independent set, upper and lower bounds for the domination and independence number, and additional results in the streaming dominating set and independent set.

Then in Chapter 4, we present our work on tree/forest streaming domination number. To begin with, in Section 4.1, we recap some existing 2-approximation schemes for tree domination number that use only easily estimable graph characteristics, such as the number of non-leaf vertices. And we show that by adding other graph characteristics in the scheme, we could further improve the approximation ratio to $3/2$. Next in Section 4.2, we cover streaming algorithms that can well-approximate these characteristics.

Our results on independence number are shown in Chapter 5. Firstly, in Section 5.1, we demonstrate our algorithms for approximating the Caro-Wei Bound in sparse graphs.

And then we introduce our results on tree/forest. Similarly, Section 5.2 includes a previous $3/2$ -approximation for trees and our new $4/3$ -approximation results with the help of additional graph characteristics. In the last section (Section 5.3), we rely on some results from Section 4.2 and present our streaming algorithm to approximate the independence number.

The organization of Chapter 6 is very similar to Chapter 4 and the last two sections of Chapter 5. We start this chapter with some known results on approximating the matching number, and we show how support vertices can help improve this approximation ratio. And lastly, we conclude that the streaming algorithms introduced in 6 can also be used to approximate the matching number.

Finally, in the last section (Section 7), our results are summarized in terms of their usefulness, novelty, and limitations. Also, we propose several potential directions for future studies.

Chapter 2

Preliminaries

2.1 Problem Definitions

2.1.1 Graph Definitions

For a undirected graph G , let $V(G)$ be the set of G 's vertices, and let n represent the number of G 's vertices, $|V(G)|$. Also, we let $E(G)$ be the set of edges in G , and m be the total number of G 's edges. A graph is a tree if it is connected and has no cycle, i.e., every two vertices is connected via exactly one path. If a graph is acyclic but not connected, then it is a forest, which can be viewed as a group of disjoint trees. It is known that in tree, $m = n - 1$. Similarly, in a forest, $m = n - k$, where k is the number of tree components. For a subset of vertices, $S \subseteq V(G)$, let $N(S)$ denote the set of vertices whose neighbours are in S (excluding S itself), and $N[S]$ to represent $S \cup N(S)$. Moreover, let $d(v)$ be the degree of a vertex v , δ be the minimum vertex degree of G , Δ be the maximum vertex degree of G , and \bar{d} be the average degree of G .

The quantities $Deg_i(G)$, $H_i(G)$, and $S(G)$ are defined as follows:

Definition 2.1. (*Deg_i and H_i*) $Deg_i(G)$ is the set of vertices in G that have degree equal to i . Similarly, $H_i(G)$ is the set of vertices in G that have degree $\geq i$. When the context is clear, we use Deg_i and H_i .

Definition 2.2. (*S*) The support vertex set of a graph G is the subset of G 's vertices which are adjacent to at least one leaf (or end vertex). Denote the support vertex set as $S(G)$, or S if the context is clear. A similar definition appears in [42].

2.1.2 Graph Problems

The **dominating set**, $DS(G)$, of a graph is a subset of $V(G)$, such that each vertex in G is either in $DS(G)$ or adjacent to at least one vertex in $DS(G)$. Let $MDS(G)$ be the minimum dominating set of graph G , and $\gamma(G)$ be the domination number (i.e., the size of $MDS(G)$).

Similarly, the **connected dominating set**, $CDS(G)$, of graph G is a variant of the dominating set. Besides the domination requirement, in a connected dominating set, the subgraph induced by the chosen set needs to be connected. That is, every vertex in $CDS(G)$ must be adjacent to at least one other vertex in $CDS(G)$. We let $MCDS(G)$ be the minimum connect dominating set and $\gamma_C(G)$ be the connected domination number (i.e., the size of the minimum connected dominating set).

A subset of vertices is an **independent set** (or a stable set) if and only if the subgraph induced by it contains no edges. Denote such a set as $IS(G)$, and as $MIS(G)$ if it has the maximum cardinality across all valid sets. Let $\beta(G)$ be the cardinality of $MIS(G)$, which is also known as the independence number.

A **matching** of G is a subset of G 's edges, such that none of them shares a common endpoint. Let $Mat(G)$ represent a valid matching and $MMat(G)$ represent the maximum cardinality matching. Let ϕ be the size of the maximum matching, the matching number.

A **vertex cover** (or $VC(G)$) of a graph G is a subset of G 's vertices, such that for every edge in G , at least one of its endpoint is in $VC(G)$. Denote the minimum cardinality vertex cover as $MVC(G)$ and its cardinality as $\tau(G)$ (also known as the covering number).

2.1.3 Randomized and Approximation Algorithms

An algorithm is **randomized** if it has access to a sequence of random bits, and it utilizes them to guide its behavior. There are generally two types of randomized algorithms, Las Vegas and Monte Carlo. A randomized algorithm is a Las Vegas algorithm if it always returns the correct solution, but some resource used (e.g., time, space) is only bounded in expectation. For example, the Quicksort algorithm is guaranteed to sort correctly; its running time is $\mathcal{O}(n \log n)$ in expectation, but $\mathcal{O}(n^2)$ in the worst-case scenario. And

an algorithm is Monte Carlo if it has worst-case guarantees on the resources usage, but it might return an incorrect solution with bounded probability.

Approximation algorithms are often found with NP-hard problems. It is known that if $P \neq NP$, then NP-hard problems cannot be solved efficiently in polynomial time. Approximation algorithms help address this problem by trading accuracy for efficiency. Approximation algorithm runs efficiently, but instead of finding the exact or optimal solution, the output of the algorithm is only guaranteed to be within a fixed distance from it. And such distance is often referred to as the approximation ratio.

Approximation ratio: Let A be an algorithm for a maximisation problem P , and Π be an instance of P . We use $A(\Pi)$ to represent the solution returned by running A on Π , and $OPT(\Pi)$ be the optimal solution of Π . For $c > 1$, we say that algorithm is a factor c -approximation algorithm if

$$\frac{OPT(\Pi)}{c} \leq A(\Pi) \leq OPT(\Pi) \quad (2.1)$$

Similarly, if P is a minimisation problem, the algorithm A is a c -approximation algorithm if

$$OPT(\Pi) \leq A(\Pi) \leq cOPT(\Pi) \quad (2.2)$$

2.1.4 Complexity Notations

We use the classic complexity notations to represent the resources required by our algorithms. In big-O notation, \mathcal{O} , denotes the asymptotic upper bound of a function. And for functions $f(n)$ and $g(n)$, $f(n) \in \mathcal{O}(g(n))$ if there exist two positive constants c and n_0 , such that

$$|f(n)| \leq c|g(n)| \text{ for all } n \geq n_0 \quad (2.3)$$

Similarly, we could define the asymptotic lower bound of a function. For two functions $f(n)$ and $g(n)$, $f(n) \in \Omega(g(n))$ if $g(n) \in \mathcal{O}(f(n))$. And two functions are asymptotically equivalent ($f(n) \in \Theta(g(n))$) if both $f(n) \in \mathcal{O}(g(n))$ and $f(n) \in \Omega(g(n))$ hold.

Lastly, we use the little- o notation, o , to denote a function which grows strictly asymptotically slower than another function. So $f(n) \in o(g(n))$ if and only if for every positive constant ϵ , there exist a positive constant n_0 , such that

$$|f(n)| \leq \epsilon |g(n)| \text{ for all } n \geq n_0 \quad (2.4)$$

Similarly, we define $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$, meaning that $f(n)$ grows asymptotically faster than $g(n)$.

Sometimes we use \tilde{O} , $\tilde{\Omega}$, $\tilde{\Theta}$, \tilde{o} , and $\tilde{\omega}$ to suppress the logarithm factors. For example, for every constant $c > 0$, we have $\mathcal{O}(n \log^c n) = \tilde{O}(n)$.

2.2 Probability Theory

In probability theory, it is known that for a countable set of events, the probability that at least one of them happens is no larger than sum of the probabilities that each event happens by their own. That is, let A be a countable set of events (of size k) and A_i be the i -th event,

$$\Pr[A_1 \vee \dots \vee A_k] \leq \sum_{i=1}^k \Pr[A_i] \quad (2.5)$$

Inequality (2.5) is also known as the **union bound** or Boole's inequality.

A **random variable** is a variable whose outcome is determined by some random events [15]. Formally speaking, the random variable could be viewed as a measurable function that maps from the sample space (the set of all possible outcomes of a non-deterministic event) to the real numbers. Random variables can be used to describe a large range of physical phenomena and random processes. For example, we could define a random variable X as the face value a fair dice with six faces. The value of X depends on the uncertain outcome of each roll. The outcomes of a random variable can be discrete or continuous, the following definitions are all referring to the discrete case (i.e., there is a finite number of outcomes).

In probability and statistics, **Mean** and **variance** are two commonly used terms to describe the characteristics of a random variable or a set of random variables. For a discrete random variable X , let I denote the sample space, which is the set of all possible outcomes of X . Its mean ($E[X]$, also known as the expected value or the first moment) is defined as

$$E[X] = \sum_{i \in I} x_i \Pr[X = x_i] \quad (2.6)$$

where x_i is the value of X when event i occurs. The variance of X , $Var(X)$, is

$$Var(X) = E[(X - E[X])^2] = \sum_{i \in I} (x_i - E[X])^2 \Pr[X = x_i] \quad (2.7)$$

There are many useful properties of mean and variance. For example, the linearity of expectation states that the expectation of a sum of random variables is equal to the sum of their individual expectations. Moreover, the expectation of a random variable times a constant is equal to the constant times the expectation of that random variable. Combining them, we have

$$E[\sum c_i X_i] = \sum E[c_i X_i] = \sum c_i E[X_i] \quad (2.8)$$

Hence, the variance of a random variable can be expressed in terms of its expectation

$$Var(X) = E[(X - E[X])^2] = E[X^2] - E^2[X] \quad (2.9)$$

Moreover, if X is non-negative, it is upper bounded by a constant factor of its expectation with constant probability. This is known as **Markov's inequality**, which states that

$$\Pr[X \geq k] \leq \frac{E[X]}{k} \quad (2.10)$$

for non-negative X and $k > 0$. Alternatively, if both the expectation and the variance are finite, one can bound the probability that X is not concentrated around its expectation. **Chebyshev's inequality** guarantees that

$$\Pr[|X - E[X]| \geq k] \leq \frac{\text{Var}(X)}{k^2} \quad (2.11)$$

For the sum of a series of random variables, $X = \sum X_i$, one can obtain a stronger concentration bound if these random variables are independent of each other. The **Chernoff Bound** below is obtained by firstly applying Markov's inequality to e^{tX} for every $t > 0$, and optimizing over t by assuming that X_i are independent. For the Chernoff Bound, we have

$$\Pr[X \geq (1 + \delta)E[X]] \leq e^{-\frac{\delta^2}{2+\delta}E[X]} \quad (2.12)$$

and

$$\Pr[X \leq (1 - \delta)E[X]] \leq e^{-\frac{\delta^2}{2}E[X]} \quad (2.13)$$

In addition, if the value of each random variables X_i is bounded by $[a_i, b_i]$ and they are still independent, Hoeffding [67] generalized the Chernoff Bound to

$$\Pr[|X - E[X]| \geq k] \leq 2e^{-\frac{2n^2k^2}{\sum(b_i - a_i)^2}} \quad (2.14)$$

which is known as the **Chernoff-Hoeffding inequality**. Panconesi and Srinivasan [103] have extended the Chernoff-Hoeffding inequality to negatively correlated Boolean random variables.

Theorem 2.3. [103] *For r negatively correlated Boolean random variables X_1, X_2, \dots, X_r , let $X = \sum_{i=1}^r X_i$, $\mu = E[X]$ and $0 < \delta < 1$, then we have:*

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\frac{\mu\delta^2}{2}} \quad (2.15)$$

2.3 Hash Functions

A hash function h is a mathematical function that maps elements from one universe, \mathcal{U} , to another universe, \mathcal{U}' , i.e., $h : \mathcal{U} \rightarrow \mathcal{U}'$. For example, $h(x) = x \bmod 5$ is a hash function

that maps elements from \mathbb{R} to $[0, 4]$. A family of hash functions, $\mathcal{H} = \{h : \mathcal{U} \rightarrow \mathcal{U}'\}$, is a group of hash functions with same input and output universe, and all of them satisfy certain mathematical property. Since hash function must be deterministic (i.e., the same input must be mapped to the same output), a common technique to introduce randomness is by selecting one hash function uniformly at random from a family of hash functions.

Often, a good hash function should achieve both uniformity and fully independence. That is, the hash function should map its inputs uniformly to its output universe, and the mapping decision should be fully independent. However, such family of hash functions would require $m2^n$ to store, where m is the output universe size and n is the input universe size. The space usage can be reduced if we relax the requirements on independence. Let $[n] = \{0, \dots, n - 1\}$. A family of hash functions, $\mathcal{H} = \{h : [n] \rightarrow [m]\}$, is **k -wise** if for every $(x_1, \dots, x_k) \in [n]^k$ and $(y_1, \dots, y_k) \in [m]^k$, we have

$$\Pr_{h \in \mathcal{H}} [h(x_1) = y_1 \wedge \dots \wedge h(x_k) = y_k] \leq \frac{1}{m^k}, \quad (2.16)$$

where $h \in \mathcal{H}$ denotes that h is choosing randomly from \mathcal{H} . By randomly select h from \mathcal{H} , the hash values of x_1, \dots, x_k are independent of each other. Without loss of generality, assuming m is a prime number. A common k -wise hash function, given by Wegman and Carter[117], is constructed as

$$h(x) = \left(\sum_{i=1}^k a_i x^{i-1} \right) \bmod m, \quad (2.17)$$

where the coefficients, a_i , are k random numbers selected from m . In this construction, a randomly selected hash function requires $\mathcal{O}(k \log m)$ bits to store all of its coefficients, and its computation time is $\mathcal{O}(k)$ ¹.

Moreover, the uniformity of a family of hash functions can be qualified by the probability that an element has the smallest hash value among a group of elements. a family of hash functions, $\mathcal{H} = \{h : [n] \rightarrow [m]\}$, is ϵ -min-wise if for every $A \subset [n]$ and $x \in [n] \setminus A$, we have

¹Assuming exponentiation and module operation can be done in $\mathcal{O}(1)$ time

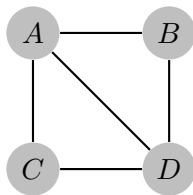


FIGURE 2.1: Example of graph streams arrival order

$$\Pr_{h \in \mathcal{H}} [\forall a \in A, h(x) < h(a)] = \frac{1 \pm \epsilon}{|A| + 1} \quad (2.18)$$

Indyk [71] showed that the ϵ -min-wise property can be achieved via a $\mathcal{O}(\log 1/\epsilon)$ -wise hash family, under the constraints that $|A| \in \mathcal{O}(\epsilon m)$. Its space usage is $\mathcal{O}(\log 1/\epsilon \log m)$ and its computation time is $\mathcal{O}(\log 1/\epsilon)$.

2.4 Streaming Models

In the data stream model, instead of being stored in the memory and supporting random access, each element arrives one by one in the form of a data stream. The goal is using one or several passes to solve a problem using much less space than the storing the whole stream.

In the case of graph streams, there are two kinds of data streams available: if each element represents an edge in the graph, then it is a **edge-arrival** stream. If each element denotes a vertex together with all of its neighbours which arrived earlier, it is a **vertex-arrival** stream. For example, if the graph in Figure 2.1 is fit into a edge-arrival stream, then its arrival sequence might be: $(A, B); (C, D); (A, C); (A, D); (B, D)$, where each element is an edge and elements are separated by semicolon. If the graph in Figure 2.1 is fit into a vertex-arrival stream, then the arrival sequence might look like: $(B, \{\}); (A, \{B\}); (D, \{A, B\}); (C, \{A, D\})$, where each element is a tuple containing a vertex and its neighbours that have already arrived before, and elements are separated by semicolon.

Moreover, we could categorize the stream into **insertion-only** model, **turnstile** model, and **dynamic** model. In the insertion-only model, only insertions of stream elements are allowed. While in turnstile and dynamic model, both insertions and deletions are

allowed. The key difference between turnstile and dynamic model is that, in turnstile model, element can be deleted if and only if it is still present (i.e., it has been inserted and not been deleted from the stream). However, in dynamic model, element can be deleted regardless whether it is presented or not. For instance, consider a stream of items identified by their unique item numbers (e.g., 1, 2, 3), and let $+$ and $-$ denote whether it is an insertion or deletion. The sequence of $+1; +2; +3$ is an insertion-only stream, the sequence of $+1; -1; +2; +3; +1; -2$ is a turnstile stream, and the sequence of $+1; +2; -2; -2; +3; +2$ is a dynamic stream.

Lastly, the order of arrival can be either **arbitrary** or **random**. In arbitrary ordering, the streaming algorithm needs to consider the worst-case ordering, but in random ordering, each element is arriving next with some probability, so it is often easier than the arbitrary ordering.

Halldórsson et al. [57] have introduced the **online streaming** model, which preserves properties from both data streams model and online model. In online streaming model, data elements arrive one by one in a streaming fashion, and the server (or the processing unit) typically does not have enough memory to store the whole stream. In addition, at any given time of the stream, the algorithm must be able to report a solution that satisfies the theoretical guarantees (e.g., approximation ratio) of the algorithm. Usually, the algorithm will start with a initial solution, and modify it based on the newly arrived element. Similar to the online model, each decision about the solution is irrevocable, meaning that once an element is added to (or removed from) the solution, it cannot be removed from (or added to) the solution later. The solution can be either stored locally or in a remote server, where in the later case, the algorithm “report” to the remote server about its decisions. We use solution space and working space to differentiate the space required to store the solution, and the additional space require to perform computations.

The stream studied in this work has several common assumptions. Firstly, like all other streamed graph research, it is assumed that the vertex universe is known in advance. Secondly, for algorithms that are developed on special graphs (e.g., tree and forest), we assume that the graph at the end of the stream is guaranteed to be that special graph, but there is no such requirement during stream processing. Moreover, similar to relevant research [39], we assume that the number of deletions in a turnstile stream is bounded

by $\mathcal{O}(n)$ (or $\mathcal{O}(\bar{d}n)$ if the average degree is known). This is because one of our algorithms (Algorithm 1) keeps track of the neighbours of sampled vertices and our analysis relies on the fact that there are at most $\mathcal{O}(n)$ insertions. If an arbitrary number of deletions is allowed, the number of insertions might be arbitrarily large and thus break our space analysis. Therefore, in order to limit the space usage, we assume that the number of deletions is $\mathcal{O}(n)$ (or $\mathcal{O}(\bar{d}n)$), so that the number of insertions is also bounded by $\mathcal{O}(n)$ (or $\mathcal{O}(\bar{d}n)$).

2.5 Elementary Streaming Algorithms

Here we introduce several elementary streaming tools that are used by our algorithms. Consider a vector v with n coordinates, that is, $v \in \mathbb{R}^n$. The coordinates of v are updated through a turnstile vector stream. Let v_i be the final value of coordinate i in v , and m be the length of the stream. Assume the value of each coordinate update is in the range of $[-M, M]$.

For $p > 0$, the L_p norms of a vector is defined as follows:

Definition 2.4 (L_p norms). Let $p > 0$ be a real number, the L_p norm of vector v is defined as $\|v\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$.

For $p \in (0, 2]$ and a constant error term $\epsilon > 0$, Kane et al. [76] have developed an algorithm to $(1 \pm \epsilon)$ estimate the L_p norm.

Theorem 2.5. [76] For all $p \in (0, 2)$, given error rate $1 > \epsilon > 0$ and fail probability $1 > \delta > 0$, there exists an algorithm that uses $\mathcal{O}(\epsilon^{-2} \log(mM) \log \delta^{-1})$ bits of space and reports $(1 \pm \epsilon) \|v\|_p$ with probability at least $1 - \delta$. Both the update time and the report time are $\tilde{\mathcal{O}}(\epsilon^{-2} \log \delta^{-1})$.

Moreover, defining $0^0 = 0$, the L_0 “norm” of a vector can be defined as

Definition 2.6 (L_0 norm). $L_0 = \|v\|_0 = (\sum_{i=1}^n |x_i|^0)$,

which also can be interpreted as the count of non-zero coordinates of the vector. And it has quotation around the word norm as it is not a valid norm in mathematics². In streaming algorithms, Kane et al. [77] also give an efficient algorithm for estimating L_0 .

²It violates the homogeneity property of a norm, which requires $\|av\| = |a|\|v\|$ all every scalar a

Theorem 2.7. [77] *Given error rate $1 > \epsilon > 0$ and fail probability $1 > \delta > 0$, there is an algorithm that uses $\mathcal{O}(\epsilon^{-2} \log(\delta^{-1}) \log(n)(\log(1/\epsilon) + \log \log(mM)))$ bits of space and $(1 \pm \epsilon)$ -approximates L_0 with at least $1 - \delta$ success probability. Its update and reporting time is both $\mathcal{O}(\log \delta^{-1})$.*

Given $k \leq n$, a vector of length n is **k -sparse** if it has at most k non-zero coordinates. For example, $v = \{0, 1, 2, 0, 0\}$ is k -sparse for any $k \leq 2$. The k -sparse recovery problem has been studied under various models, which is about recovering a k -sparse vector under different constraints. In the streaming context, when the vector is k -sparse, we want to recover it using the least space and time as possible. And when it is not k -sparse, we are fine with either a fail output, or a subset of its coordinates. Cormode and Firmani [37] showed that

Theorem 2.8. [37] *Given k and error probability $0 < \delta < 1$, there exists an algorithm that can recover a k -sparse vector exactly with probability $1 - \delta$. The algorithm uses $\mathcal{O}(k \log n \log(k/\delta))$ bits of space.*

Sparse recovery can recover the vector if it is sparse, but what if the vector is not k -sparse and we still want find out k coordinates that are representative enough? Sometimes, coordinates with value greater than a certain threshold are considered as representative coordinates, and we want to output a list of coordinates that include all of them. This problem is also known as the ϵ -heavy hitter problem. An ϵ -heavy hitter is defined as

Definition 2.9 (ϵ -heavy hitter). Given $p \geq 1$ and $0 < \epsilon < 1$, a coordinate i of the vector v is an ϵ -heavy hitter of v in its L_p norm if $|x_i|^p \geq \epsilon \|v\|_p^p$.

In this work, we focus on one variant of the heavy hitter problem defined as follows.

Definition 2.10 (The ϵ -Heavy Hitter Problem). Given $p \geq 1$ and $0 < \epsilon < 1$, output a list of size $\mathcal{O}(\epsilon^{-1})$ that contains all ϵ -heavy hitters of v .

More specifically, we only consider the case where $p = 1$. Many heavy-hitter algorithms for $p = 1$ are developed based on the Count-Min sketch [38]. And in the same paper, Cormode and Muthukrishnan obtained the following theorem for heavy hitters.

Theorem 2.11. [38] *Given $\epsilon, \phi \in (0, 1)$, and a turnstile stream of coordinate updates on vector v , there is an algorithm that uses $\mathcal{O}(\epsilon^{-1} \log(n) \log(\frac{2 \log n}{\delta \phi}))$ space, such that*

all items with frequency at least $(\epsilon + \phi)\|v\|_1$ will be outputted, and with probability $1 - \delta$, no items with frequency less than $\phi\|v\|_1$ will be outputted. It has update time $\mathcal{O}(\log(n) \log(\frac{2\log n}{\delta\phi}))$ and query time $\mathcal{O}(\epsilon^{-1})$.

Chapter 3

Further Related Works

3.1 Dominating Set

In this section, we firstly introduce other research in computing the minimum dominating set or the domination number. Then we cover some additional *MDS* studies under the data streams model.

3.1.1 Approximating the Dominating Set

In certain graphs, for the dominating set problem, its greedy algorithm (or variants of its greedy algorithm) has better approximation ratio guarantee than it is in general graphs. For example, since the approximation rate of the greedy algorithm above depends on the maximum degree, Δ , it gives a constant-factor approximation in constant-bounded-degree graphs. In addition, in graphs with minimum vertex cover of size r , an $\mathcal{O}(r \ln \gamma)$ approximation can be achieved, where γ is the domination number [22, 50] (remark: [50] used LP based algorithm). Siebertz [112] has obtained similar upper-bound results on $K_{t,t}$ -free graphs (graphs that do not have complete bipartite graph with t vertices on each side as subgraphs), since every $K_{t,t}$ -free graph has a vertex cover of size at most t . His algorithm outputs a $\mathcal{O}(t \ln \gamma)$ approximation and is a variant of the classic greedy algorithm. In each round, instead of selecting the vertex with the maximum contribution, his algorithm selects a set of vertices so that it includes at least one vertex from the minimum dominating set when the number of undominated vertices is large. And when the number of remaining undominated vertices is small, his algorithm only

performs slightly worse than the greedy algorithm. Moreover, on graphs with degeneracy at most d , Jones et al. [74] slightly changed the greedy algorithm and obtained a d^2 -approximation algorithm.

Meanwhile, other techniques have used the linear program (LP) to compute the *MDS*. In graphs with arboricity α , Lenzen and Wattenhofer [87] have provided a linear-time $(\alpha^2 + 3\alpha + 1)$ -approximation algorithm for the *MDS* problem under distributed model, where each node is viewed as a server that can exchange information with servers that have an edge between them. And their algorithm can be generalized to a central algorithm with appropriate forest decomposition. Bansal and Umboh [10] then showed a 3α -approximation algorithm and proved that it is NP-hard to achieve approximation ratio better than $(\alpha - 1 - \epsilon)$ for any $\epsilon > 0$. Their upper-bound algorithm solves the relaxed linear program (LP) of the dominating set problem first. And then it includes every vertex that has weight greater than $\frac{1}{3\alpha}$ into a candidate dominating set S . Lastly, in the remaining vertices, it selects all vertices that are not dominated by S into S . Clearly, by the construction rule, S is a valid dominating set. And Bansal and Umboh [10] showed that the size of S is at most $3\alpha\gamma$ via charging arguments. Moreover, since the degeneracy d of graphs with arboricity α must be between α and $2\alpha - 1$ (inclusively), hence this algorithm can be generalized to a $3d$ approximation algorithm for graphs with degeneracy d . Furthermore, under a more careful analysis, Dvořák [46] has improved the approximation of this algorithm to $2d - 1$. See [88] for experimental evaluations of these two LP-based algorithms, the greedy algorithm, and their combinations.

Currently, graphs with constant-bounded-arboricity(or constant-bounded-degeneracy) are the most general graphs with constant *MDS* approximation in polynomial time. Siebertz [112] has shown that even in $K_{3,3}$ -free graphs, unless $\text{NP} \subseteq \text{DTIME}(2^{n^{1-\epsilon}})$ for some $\epsilon < \frac{1}{2}$, we can not achieve an approximation rate better than $c \frac{\log n}{\log \log n}$ in polynomial time for some constant c .

For even sparser graphs, Cockayne et al. [36] gave a linear-time algorithm for the more general mixed dominating set problem in trees, where nodes can have different property such as must in the chosen in the set or must be dominated. Similarly, Takamizawa et al. [114] and Kikuno et al. [81] independently demonstrated a linear-time algorithm for series-parallel graphs, which can be generalized to graphs with bounded treewidth with additional polynomial time.

3.1.2 Computing the Domination Number

For some special graph classes, $\gamma(G)$ is known to be a fixed number. For example, $\gamma(G)$ is 1 when G is a complete graph, a star graph or a wheel graph, and $\gamma(G) = 2$ when G is a complete bipartite graph or a crown graph. In general, the domination number highly depends on the structure of the graph, and it can vary from $\mathcal{O}(1)$ (e.g., a star graph) to $\Omega(n)$ (e.g., a path). Hence, in other graph classes, instead of determining $\gamma(G)$ exactly, researchers have been trying to find its upper/lower bound in terms of other easily estimable graph parameters, such as the maximum/minimum vertex degree.

3.1.2.1 Upper Bounds of Domination Number

Trivially, in any graph, $\gamma(G)$ is upper bounded by the number of G 's vertices, n . And it is not hard to see that $\gamma(G) = n$ when all vertices are isolated. This upper bound can be further reduced with the knowledge of G 's minimum vertex degree, δ . Arnautov [4] and Payan [105] showed that

$$\gamma(G) \leq \frac{n}{\delta + 1} \sum_{i=1}^{\delta+1} \frac{1}{i}, \quad (3.1)$$

which implies that

$$\gamma(G) \leq \left(\frac{\ln(\delta + 1) + 1}{\delta + 1} \right) n, \quad (3.2)$$

when $\delta \geq 1$ [70], and this bound is proven to be asymptotically optimal when $\delta \rightarrow \infty$ [2]. Imrich et al. [70] proved Bound (3.2) by calculating the expected size of a randomly constructed dominating set. They choose a random subset X of G 's vertices by uniformly and independently sampling each vertex with probability p . Let Y be the vertices in G that are not dominated by X , clearly X and Y form a dominating set of G . It is known that the size of the minimum dominating set, $\gamma(G)$, is at most the expected size of the constructed dominating set. Hence, by choosing the probability p that minimizes the expected size, this upper bound is obtained. Clark et al. [35] have proved a slightly better bound when $\delta \geq 1$.

$$\gamma(G) \leq \left(1 - \prod_{i=1}^{\delta+1} \frac{i\delta}{i\delta+1}\right)n, \quad (3.3)$$

which then is improved by Biró et al. [13] in 2012. They showed that

$$\gamma(G) \leq \left(1 - \frac{\delta^2 - \delta + 1}{1 + \delta \prod_{i=1}^{\delta-1} \left(1 + \frac{\delta+1}{i\delta}\right)}\right)n \quad (3.4)$$

Recently, Bujtás and Klavžar [24] further improved this upper bound for $\delta \leq 50$. They set up a set of inequalities that involve δ and $(\delta + 2)$ other positive variables. And they proved that $\gamma(G)$ is no more than of a specific proportion of n , where the proportion is determined by the ratio between two of the variables. Hence, different upper bounds can be calculated for different values of δ by finding the solutions to the inequalities. For example, $\gamma(G) \leq \frac{2671n}{7766}$ when $\delta = 5$, and $\gamma(G) \leq \frac{1702n}{5389}$ when $\delta = 6$. Currently, Bound (3.4) is the best upper bound for $\delta \geq 51$, while [24] provides the best upper bound for $6 \leq \delta \leq 50$. The best upper bound for $\delta = 5$ is $\frac{n}{3}$ [23]. And the best upper bound for $1 \leq \delta \leq 4$ is

$$\gamma(G) \leq \frac{\delta n}{3\delta - 1}, \quad (3.5)$$

which is a generalized equation of several works. Ore [102] proved it for graphs with $\delta = 1$, Blank [14], McCuaig and Shepherd [92] have independently proved that it holds for all graphs with $\delta = 2$ and $n \geq 8$. For graphs with $n < 8$, there are seven exceptions (e.g. C_4 , a cycle graph with 4 vertices). Moreover, the results for $\delta = 3, 4, 5, 6$ are separately proved in [72, 108, 113, 123] by carefully choosing a dominating set based on the vertex disjoint paths cover (a set of disjoint paths that covers all G 's vertices) with specific properties. Note that [63] conjectured that Bound (3.5) holds for all $\delta \geq 1$, but for any $\delta \geq 5$, Bound (3.4) and the results from [24] are already better than it.

The upper bounds can be further reduced for some special graph classes. Some researchers have considered graphs without certain types of cycle. For connected C_4 -free graph with $\delta \geq 1$, Brigham and Dutton [21] proved an upper bound of $\frac{1}{2}(n - \frac{\delta(\delta-1)}{2})$. Küpper and Volkman [85] have improved it to $\frac{3}{7}(n - \frac{(3\delta+1)(\delta-2)}{6})$ for the same graph class with $\delta \geq 2$. In 2009, Harant and Rautenbach showed that if the graph G does

not contain cycles of length 4, 5, 7, 10, or 13, then a $\frac{3n}{8}$ upper bound can be obtained for any $\delta \geq 2$ (as for general graph, this upper bound is obtained for $\delta \geq 3$). Similarly, improved upper bounds can be established for regular graphs. For cubic graphs, Kostochka and Stodolsky [83] have proved a $\frac{4n}{11}$ upper bound for any connected cubic (i.e. $\Delta = \delta = 3$) graph with $n > 8$. The same upper bound has been proven for quartic graph (i.e. $\Delta = \delta = 4$) in [89].

3.1.2.2 Lower Bounds of Domination Number

Only a few results have been obtained on the lower bound side. It is known that in any graph,

$$\frac{n}{\Delta + 1} \leq \gamma(G), \quad (3.6)$$

where Δ is the maximum vertex degree of G [20].

Similarly, the lower bound can be improved in certain graph classes. For example, Meierling and Volkman [95] and Lemańska [86] have independently proved that $\frac{n-|L|+2}{3} \leq \gamma(T)$ for any tree T with $n \geq 3$, where $|L|$ is the number of T 's leaves. This bound is sharp as the equality can be found in any paths with multiple-of-3 number of vertices.

3.1.2.3 Approximating the Domination Number

Instead of obtaining an upper bound and a lower bound for the domination number, another line of research has been focused on approximating the actual domination number.

Give a graph G , the distance between two vertices, u and v , is the number of edges on the shortest path from u to v (excluding u and v). For example, if u and v are adjacent (i.e., share a common edge), the distance between them is 1. The k -distance independent set of G is a set of G 's vertices such that no two of them are at distance less or equal than k . The maximum cardinality of G 's k -distance independent set is referred as the k -distance independence number. It is not hard to see that the 2-distance independence number is a lower bound for the domination number in any graphs, as in 2-distance

independent set, every vertex must be adjacent to at most one vertex from the set. Böhme and Mohar [16] proved that domination number is also constantly upper bounded by the size maximum 2-distance independent set in graphs excluded proper complete bipartite graph minors. For example, in planar graphs, the size maximum 2-distance independent set is a 20 approximation of the size of the minimum dominating set. Dvořák [45] extended this result to a linear-time constant-approximation algorithm for graphs with bounded expansion. However, note that in graphs with arboricity α , the ratio between domination number and 2-distance independence number could be as large as $\mathcal{O}(n)$. Consider the incidence graph of a complete graph K_n , which is a bipartite graph with $V(K_n)$ and $E(K_n)$ as two parts of the graph. And there is an edge between $u \in V(K_n)$ and $e \in E(K_n)$ if $u \in e$ in K_n . Moreover, there is an additional vertex w , such that w is adjacent to every vertex in $E(K_n)$. According to Dvořák (Lemma 5, [46]), this graph has domination number at least $\frac{n}{2}$, 2-distance independence number at most 2, and arboricity at most 3.

When giving query access to the adjacency list of each vertex, constant-time approximation algorithms for the domination number has been explored in many studies. Parnas and Ron [104] and Nguyen and Onak [100] provided algorithms that achieve $\mathcal{O}(\log \Delta)$ approximation for graphs with bounded degree Δ and $\mathcal{O}(\log \bar{d}/\epsilon)$ approximation for graphs with average degree \bar{d} , both approximations have additive error of ϵn . Moreover, Hassidim et al. [61] have improved it to $(1 \pm \epsilon n)$ approximation for graphs with maximum degree of Δ .

3.1.2.4 Bounds of Connected Domination Number

Since only connected graphs (graphs without isolated vertices) can have valid connected and total dominating set, the results in this section are all obtained for connected graphs.

Determining the connected domination number for arbitrary graphs is also NP-complete [54], which holds for even planar graph and quartic graph. Sampathkumar and Walikar [110] state that $\frac{n}{\Delta+1} \leq \gamma_C(G) \leq 2m - n$, and $\gamma_C(G) \leq 2n - 2$ for $n \geq 3$. In 1984, an upper bound of $n - \Delta$ has been established [64]. Moreover, the same paper has shown that for graphs with $n > 2$, $\gamma_C(G) + l(G) = n$ [64], where $l(G)$ is the maximum leaf number of G (the largest number of leaves in any of G 's spanning trees). So that for any tree T , $\gamma_C(T) = n - |L|$, where $|L|$ is the number of T 's leaves. Furthermore, by

combining with $l(G)$'s lower bound $(n - 3\lfloor \frac{n}{\delta+1} \rfloor + 2)$ [82], an upper bound of $(3\lfloor \frac{n}{\delta+1} \rfloor - 2)$ can be obtained [63].

Moreover, the relationship between $\gamma_C(G)$ and $\gamma(G)$ has been studied in much research. It is known that $\gamma_C(G) \leq 3\gamma(G) - 2$ [44].

3.2 Streaming Dominating Set

Fafianie and Kratsch [51] studied the streaming kernelization of a variant of dominating set, k -edge dominating set, where we are seeking a set of edges of size at most k that dominate the rest of other edges. In streaming kernelization, instead of computing the solution, the algorithm gave a equivalent and space-reduced instance called kernel that has the exact same answer. They showed a two-pass deterministic algorithm that uses $\mathcal{O}(k^3 \log n)$ bit of local memory and outputs a $\mathcal{O}(k^3 \log k)$ -space kernel in insertion-only arbitrary-ordering edge-arrival model. In short, their algorithm computes a kernel for the $2k$ -vertex cover problem in the first pass, and in the second pass, it includes edges between vertices that are kept during the first pass.

The dominating set problem is closely related to the set cover problem because of the interchangeability between them. For instance, given a graph G , we can construct a set of sets by letting each set represent a vertex and its neighbours in G . Clearly, a subset of G 's vertices is a dominating set if and only if their corresponding sets is a set cover. Alternatively, give a set of sets, we can build a graph by creating a vertex for each set and a vertex for each element ¹. Two vertices share an edge if they are both indicating some sets, or one of them is indicating a set and the other one is indicating an element in that set. One can show that a subset of sets is a set cover if and only if their corresponding vertices (note: here only the vertices indicating sets are considered) form a dominating set.

Therefore, as indicated by the results from the minimum set cover problem [5], any randomized streaming algorithms that α approximates the minimum dominating set problem on general graphs must use $\tilde{\Omega}(\frac{n^2}{\alpha^2})$ bits, where $\alpha = o(\sqrt{\frac{n}{\log n}})$ and $\tilde{\Omega}$ suppress the log factors. Similarly, Banerjee and Bhore [9] have studied the generalization of DS , k -tuple dominating set, and a variant of DS , the Liar's dominating set. They showed

¹Here we assume that the set number universe and set element universe are disjoint

a $\Omega(n^2)$ lower bound for both problems in graph streams. Recently, by reduction from the INDEX problem, Chitnis and Cormode [31] showed that $\Omega(n^2)$ space is required to answer the one-pass streaming k -dominating set problem for even $k = 3$. In addition, for edge dominating set, perhaps surprisingly, Fafianie and Kratsch [51] also proved that for graphs with m edges, $\Omega(m)$ space is required for any single-pass deterministic kernelization algorithm in the same model. To illustrate, consider $k = 1$ and an graph G that has a star of $m - 1$ edges and one extra edge incident on one of the star leaves. Trivially there is an edge dominating set of size 1 and we expected the kernel to have the same size. The adversarial stream shows all edges of star first and then the extra edge. So that when the last edge arrives, the algorithm must be able to decide whether one of its vertex has appeared in the previously shown star or not, hence according to the famous *set reconciliation problem*, $\Omega(m)$ space is required. This lower bound is generalized to $\Omega(n)$ for any randomized algorithm in [33] with similar proof technique.

3.3 Independent Set

In this section, similar to Section 3.1, we firstly introduce other research in computing the maximum independent set or the independence number. Then, we cover additional relevant studies under the streaming model.

3.3.1 Approximating the Independent Set

In sparse graphs, one can also obtain a approximation ratio of $\mathcal{O}(\bar{d})$ without the help of semi-definite programming. Hochbaum [66] gave a LP-based algorithm that outputs an independent set with approximation ratio $\frac{\bar{d}+1}{2}$ using $\mathcal{O}(\bar{d}n^{3/2})$ time. The algorithm relies on the LP program of the fractional vertex cover problem introduced in [99], where each node could be assigned with value 0, 1/2, or 1. Based on the LP solution, the graph nodes can be partitioned into three subsets R (vertices with solution value 0), Q (vertices with solution value 1/2), and P (vertices with solution value 1). Note that at least one of the maximum independent set in the original graph contain all vertices in R and none of the vertices in P . And for the subgraph induced by Q , $\beta(Q)/n_Q$ is at most 1/2. Halldórsson and Radhakrishnan [58] further combined this technique with the greedy algorithm above, and improved the approximation ratio to $\frac{2\bar{d}+3}{5}$.

Furthermore, in even more restricted graphs or graphs that exclude certain minors, one can compute the *MIS* exactly. For example, there is a polynomial-time algorithm for finding the *MIS* in claw-free graphs [101], P_5 -free graphs [90], interval graphs ², and bipartite graphs. For interval graphs, the independent set problem is also known as the Interval Scheduling problem, and it can be solved using the earliest deadline first scheduling algorithm, which iteratively picks the interval that has the earliest end time. For bipartite graphs, as implied by König's theorem, we could use the well-known bipartite matching algorithm to compute a maximum independent set. And for chordal graphs ³, one can find the *MIS* in just linear time [53].

3.3.2 Computing the Independent Number

3.3.2.1 Upper bounds of Independence Number

For a general graph, its independence number can be upper bounded by its maximum and minimum degree. Given the maximum degree Δ , it is not hard to show that $\beta \leq n - \frac{m}{\Delta}$ [119]. For every independent set I , there is no edges in the subgraph induced by I , hence for every edge, at least one of its end must be in $V(G) \setminus I$. Since the degree of each vertex is at most Δ and there are at most $n - \beta$ vertices in $V(G) \setminus I$, we have $m \leq \Delta(n - \beta)$. Rearranging the terms, we have the inequality.

Alternatively, there is a trivial bound of β using the minimum degree, $\beta \leq n - \delta$. This can be shown by considering any vertices in the independent set, it must have at least δ neighbours that can not be included into the set, hence the inequality holds.

3.3.2.2 Lower bounds of Independence Number

In terms of the lower bound on the independence number, Harant and Schiermeyer have shown that for any graphs, the size of every solution outputted by the simple greedy algorithm is at least $\frac{(2m+n+1) - \sqrt{(2m+n+1)^2 - 4n^2}}{2}$. Also, it is not hard to see that the inverse of the chromatic number (χ) is no greater than the independence number. The chromatic number is the smallest number of colors required to color each vertex in the

²But in the two-dimensional geometric settings, it is NP-hard to find a *MIS*

³A graph is a chordal graph if every induced cycle contains exactly 3 vertices. A cycle is a induced cycle if it is a cycle and no non-adjacent vertices in this cycle shares a common edge

graph, such that no adjacent vertices have the same color. Clearly, all vertices that have the same color form an independent set, as none of them shares an edge. By the Pigeonhole Principle, at least one of the colors contains at least $1/\chi$ vertices.

The lower bound can be further improved with the knowledge of vertex degree, Turán [115] showed that for any graph

$$\beta \geq \frac{n}{1 + \bar{d}}, \quad (3.7)$$

which is also known as the Turán bound. As a result of the Turán Bound, we could bound the independence number by the maximum degree as $\beta \geq \frac{n}{1 + \Delta}$, since $\Delta \geq \bar{d}$. However, this bound is always no better than the Turán Bound. The solution of the greedy algorithm that iteratively adds the minimum-degree vertex is shown to have size at least of the Turán Bound. And indeed this greedy algorithm is an elegant proof of the Turán Bound.

Caro [29] and Wei [118] have independently demonstrated that

$$\beta \geq \sum_{v \in V(G)} \frac{1}{1 + d(v)} \quad (3.8)$$

which is also known as the Caro-Wei Bound. Clearly, the Caro-Wei Bound is better than the Turán Bound as it is always no less than the Turán Bound (see [17] for a simple proof). We could prove the Caro-Wei bound as follows. Consider a random permutation of G 's vertices, and add a vertex v in S if there is no edges between it and the vertices after it (i.e. it appears at the last in $N[v]$). By construction, S is a valid independent set with expected size $\sum_{v \in V(G)} \frac{1}{1 + d(v)}$, as each vertex is included with probability $\frac{d(v)!}{(d(v)+1)!} = \frac{1}{1 + d(v)}$. Hence, the maximum independent set is at least of this expected size. Indeed one can translate the proof into simple approximation algorithms for the independent set. For example, we could randomly permute the vertices, and output the set of vertices that precede all of their neighbours as the solution. Alternatively, we could randomly assign weights to each vertex, and include a vertex into the solution if it has the smallest weight among it and all of its neighbours. By design, both result sets are independent, and their expected size is exactly the value of Caro-Wei Bound. In fact, Caro and Tuza have demonstrated that even for large classes of hyper-graphs, the Caro-Wei bound is

the strongest lower bound that can be obtained from the degree information alone [30]. And currently, we do not know if there is an asymptotically better lower bound on general graphs.

Meanwhile, Boppana et al. [17] have demonstrated that in sparse graph with average degree \bar{d} , the performance ratio of Caro-Wei bound is at most $\frac{\bar{d}+2}{1.657}$, which is approximately $1.207(\alpha+1)$ in graphs with arboricity α , and 4.828 in planar graphs. And for graphs with maximum degree Δ , the Caro-Wei Bound can approximate the independence number with a ratio of $\frac{\Delta+1}{2}$.

For sparse graph, finer lower bounds can be obtained. For example, it is well known that in trees with $n \geq 2$, $\beta \geq \frac{n}{2}$. This is because every tree is also bipartite, and by the property of bipartite graph, each side of the bipartite graph itself is a valid independent set. Also, By the Pigeonhole Principle, one of its sides must have size at least $\frac{n}{2}$, thus $\beta \geq \frac{n}{2}$. Moreover, for planar graphs, we have $\beta \geq \frac{n}{4}$. This is implied by the Four-Color Theorem as every planar graph is four-colorable. So that the set of vertices with the same color forms a independent set, and again by the Pigeonhole Principle, one of the color must have size at least $\frac{n}{4}$.

3.4 Streaming Independent Set

Halldórsson et al. [59] designed a simple sampling-based algorithm that can c -approximate the maximum independent set using $\tilde{\mathcal{O}}(\frac{n^2}{c^2})$ space, where $\tilde{\mathcal{O}}$ suppresses the logarithm factors. Their algorithm samples $\frac{n}{c}$ vertices and maintains the subgraph induced by them. By running the offline maximum independent set algorithm (in exponential time) on the induced subgraph, we could achieve an approximation ratio of c . Cormode et al. [40] also showed that in vertex-arrival stream with very large average degree, there exist an algorithm that $(\log n)$ -approximates the Caro-Wei Bound using $\mathcal{O}(\log^3 n)$ space.

We also remark there is another line of works have studied the independent set problem in other special graph classes, such as the interval graphs and geometric graphs [8, 27, 48].

In terms of other lower bounds, Cormode et al. [41] have studied the hardness of the streaming independent set problem on general graphs in different stream models. By

reduction from the classic INDEX problem, they showed that for every one-pass algorithm, $\Omega(n^2)$ space is required for finding even a maximal independent set. Moreover, this problem does not become easier we relaxed to approximated solution or even partial solution. Halldórsson et al. [59] showed a $\Omega(\frac{n^2}{c^2 \log^2 n})$ space lower bound for every algorithm that c -approximates the maximum independent set problem, which matches (up to logarithmic factors) the $\tilde{O}(\frac{n^2}{c^2})$ -space upper bound algorithm they provided in the same work (see above). Also if $c \in o(\log n)$, the lower bound can be improved to $\Omega(\frac{n^2}{c^4})$. For small values of m , Braverman et al. [19] showed that $\Omega(\frac{m}{c^2})$ space is required to c -approximate the *MIS*. And by reduction from a graph-based variant of the INDEX problem, Cormode et al. [41] showed a $n^{2-o(1)}$ space lower bound for finding an independent set that covers only $n - n^{1-\epsilon}$ vertices for every $\epsilon > 0$. And if we relaxed the covering condition to a constant fraction of n , the space lower bound is $n^{1+\Omega(\frac{1}{\log \log n})}$. As for the vertex-arrival stream, Braverman et al. [19] has proved a $\Omega(\frac{n}{c^2})$ space lower bound for c approximations, and their result is built on a even more relaxed streaming model, where each vertex is arrived with all of its neighbours rather than only the neighbours who have appeared before. This lower bound is further improved by Cormode et al. [41]. They showed by reduction from a multiparty generalization of the INDEX problem, that every one-pass algorithm that c -approximates the maximum independent set would require $\Omega(\frac{n^2}{c^T})$ space.

Chapter 4

Streaming Dominating Set

In this chapter, we introduce our results on domination number in streamed trees and forests. A graph is a tree if it is connected and contains no cycles, and is a forest if it consists of one or more disjoint trees. Previous results showed that the tree domination number can be 3-approximated using just the number of non-leaf vertices [47, 84, 86, 95].

To begin with, in Section 4.1, we prove that this approximation ratio can be further improved to 2 and applied to forests by knowing the number of support vertices. Recall that a vertex is a support vertex if it is adjacent to at least one leaves. Moreover, we show that the tree connected domination number, is exactly the number of non-leaf vertices.

Later in Section 4.2, we present our streaming algorithm that $(1 \pm \epsilon)$ -approximate the number of non-leaf vertices using one pass and $\text{polylog}(n)$ space, as well as our streaming algorithm that $(1 \pm \epsilon)$ -approximate the number of support vertices in two passes and $\tilde{O}(\sqrt{n})$ space. Hence, the tree connected domination number can be $(1 \pm \epsilon)$ -approximated in one pass and $\text{polylog}(n)$ space. And the forest domination number can be $2(1 \pm \epsilon)$ -approximated in two passes and $\tilde{O}(\sqrt{n})$ space.

4.1 Tree Approximation

Let $\gamma(G)$ be the domination number of graph G , H_2 be the set of non-leaf vertices, and S be the set of support vertices. In this section, we prove that $\frac{|H_2|+|S|}{2}$ gives a

2-approximation of the domination number, $\gamma(G)$. To begin with, in Section 4.1.1, we briefly review previous result that gives a 3-approximation of $\gamma(G)$ in trees with order $n \geq 3$. And then in Section 4.1.1, we prove the upper bound and the lower bound of our estimate, respectively. Our estimate not only have a better approximation ratio, but also can be applied to trees with $n \geq 2$, hence it can be generalized to forest without isolated vertices.

4.1.1 3-approximation of Domination Number

Previous works showed that

Theorem 4.1. [47, 84, 86, 95] *For every tree with $n \geq 3$, $\frac{|H_2|}{3} \leq \gamma(G) \leq |H_2|$.*

Therefore, for every tree with $n \geq 3$, $|H_2|$ gives a 3-approximation of the domination number. Here we cannot remove the constraints on order, i.e., $n \geq 3$. This is because trees with 2 vertices is a essentially path of length 2, which contains no non-leaf vertices but has a domination number of 1. The upper bound of Theorem 4.1, $\gamma(G) \leq |H_2|$, can be easily proved via showing H_2 itself is always a dominating set. And the lower bound of Theorem 4.1, $\frac{|H_2|}{3} \leq \gamma(G)$, can be proved via induction on the order.

We also remark that Meierling and Volkmann [95] as well as Lemańska [86] have established a higher lower bound, $\frac{|H_2|+2}{3} \leq \gamma(T)$. But for the sake of our approximation algorithm, the $\frac{|H_2|}{3}$ lower bound is sufficient.

4.1.2 2-approximation of Domination Number

In this section, we show that the tree approximation ratio above can be improved to 2 by introducing the number of support vertices. Moreover, our estimate holds for every tree with order $n \geq 2$, thus it can be generalized to every forest without isolated vertices. Before showing the bounds, we first prove the following lemma on every connected graph with $n > 2$.

Lemma 4.2. *For every connected graph with $n > 2$, there exists at least one minimum dominating set that contains all support vertices and no leaves.*

Proof. We prove this by showing that given any minimum dominating set, we could always adjust it to make it contain no leaves and all support vertices without increasing its size.

Firstly, it is clear that in every connected graph with $n > 2$, all support vertices have degree greater than 1 (i.e., are not themselves leaves). This is because if a vertex v is a support vertex, by the definition of support vertex, it must be adjacent to at least one leaf. If v is also a leaf, then it must have at most one neighbour. Hence, combining these two claims, v must be adjacent to one and only one leaf. Then, graph G must contain only two vertices, v and its leaf neighbour, contradicting our assumption that $n > 2$.

Now assume we are given a minimum dominating set. If the dominating set contains no leaves and all support vertices, then nothing needs to be done. If not, then we could always remove the leaves from the dominating set and add their neighbours into the dominating set. Note that by our definition of support vertices, the neighbours of leaves must be also support vertices. First of all, the resulting set is still a dominating set as newly added support vertices dominate the removed leaves and themselves. Moreover, the dominating set must contain all support vertices, otherwise at least one leaf is undominated. Lastly, the size of the new dominating set is no larger than the original dominating set, because each removed leaf can introduce at most one neighbour (i.e., support vertex) into the new set. ■

Now we give an upper bound and a lower bound on the domination number of a tree or a forest (the upper bound actually can be applied for any connected graphs of size at least 2). Both bounds are (nearly) sharp and are expressed in terms of the number of non-leaf vertices ($|H_2|$) and the number of support vertices ($|S|$).

Lemma 4.3. *For every connected graph G ,*

$$\frac{|H_2| + |S|}{2} \geq \gamma(G) \tag{4.1}$$

Proof. Firstly, consider a graph with order exactly 2: since there is no isolated vertex, it must be a path of length 2, P_2 . Each of the two vertices in P_2 is both a support

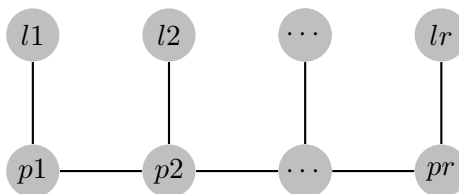


FIGURE 4.1: Sharp example of our upper bound on tree domination number

vertex and a leaf and we could pick either of them as the dominating set. Hence, $\frac{|H_2|+|S|}{2} = 1 = \gamma(G)$, the inequality holds.

Next we prove the inequality for graphs with $n > 2$. It is known that if a graph G is split into k disjoint subgraphs, G_1, G_2, \dots, G_k , we have $\gamma(G) \leq \gamma(G_1) + \gamma(G_2) + \dots + \gamma(G_k)$. This is because connecting two or more disjoint graphs only introduces new edges, thus vertices do not become undominated. Rewrite the inequality as $|S| + \frac{|H_2|-|S|}{2}$, the first term could be viewed as adding all vertices in $|S|$ to the dominating set. By doing so, all of the vertices in $N[S]$ (including all leaves) are dominated. The second term could be viewed as an upper bound on the domination number after removing all S and leaves. Clearly, the remaining graph is a forest with $(|H_2| - |S|)$ vertices. Thus it remains to prove that in such a forest, there exists a dominating set of size at most $\frac{|H_2|-|S|}{2}$.

Note that all isolated vertices in the induced graph are already dominated by some vertices in S , because a vertex becomes isolated only if all of its neighbors are in S . Thus, we only need to show that for each tree components (of size $k > 1$) in the remaining forest, there exists a dominating set of size at most $\frac{k}{2}$. Ore [102] proved that for every graph G with order n and no isolated vertices, $\gamma(G) \geq \frac{n}{2}$. Hence, this lemma follows. ■

Moreover, this upper bound is sharp. An example would be P_2 as shown in the proof. An example of arbitrary size is illustrated by Figure 4.1. Given a path of arbitrary length, r , the graph in Figure 4.1 is constructed by adding a leaf to every vertex on the path. Clearly, we have $|S| = |H_2| = \gamma = r$.

Lemma 4.3 gives an upper bound of the domination number in every connected graph, thus it also holds for every tree with $n \geq 2$. In order to show that $\frac{|H_2|+|S|}{2}$ is a 2-approximation of the tree domination number, it remains to show that the domination

number is also lower bounded by $\frac{|H_2|+|S|}{4}$. Hence, we present our lower bound result as follows.

Lemma 4.4. *For every tree T with $n \geq 2$,*

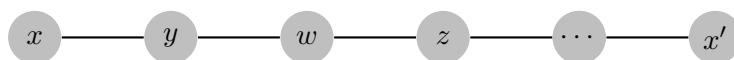
$$\gamma(T) \geq \frac{|H_2| + |S|}{4} \quad (4.2)$$

Proof. We prove it by induction on the number of nodes. We consider the base case as a tree with $n \leq 4$, which can be easily verified by hand.

Therefore it remains to prove the induction. To begin with, denote two or more leaves as twins if they share a common parent node (i.e., a support vertex). We claim that, if the tree contains twins, then we are done by induction. To illustrate, consider removing any one of the twins from the tree. By Lemma 4.2, there exists one minimum dominating set that does not contain any leaves, and removing one of the twins certainly does not remove any node in this minimum dominating set. Hence, after the removal, the domination number γ , the number of non-leaf vertices $|H_2|$, and the number of support vertices $|S|$ stay the same. Therefore, by induction hypothesis, Lemma 4.4 holds.

Hence, we assume there are no twins and consider the longest path, P , in the tree. As shown in the graph below, denote the two endpoints as x and x' respectively, let y be the “parent” of x , w be the “parent” of y , and z be the “parent” of w .

We argue that the length of P must be greater than 4, thus z and x' must be two different vertices. This can be seen via contradiction. Assuming P has length 4, then z and x' are denoting the same vertex. Since P is the longest path, x and z (or equivalently, x') must be leaves, and the neighbours of y and w must be leaves. Otherwise, if x or z is not leaf, or there is a path of length two incident on y or w , P is not longest path, violating our choice of the longest path. If y or w have leaf neighbours other than x and z , then there exist twins in the graph, violating our assumption of no twins. If x is the only leaf adjacent to y , and z is the only leaf adjacent to w , then this graph has order 4 and should be considered as the base case. Similar arguments can be obtained for P with length smaller than 4. Hence, P 's length must be greater than 4.



We claim that if w has a leaf incident on it (i.e., w is a support vertex), then we are done by induction. Note that by Lemma 4.2, assuming the optimal dominating set contains both w and y as they are support vertices. Note that y must have degree exactly 2, otherwise the graph has either a twin, or a path longer than the path from x to x' (violating our choice of the longest path). Hence, deleting x, y becomes a leaf which is already dominated by w . Therefore, after removing x , the optimal dominating set size goes down by one. Since y becomes a leaf, both H_2 and S decrease by one, the inequality holds by our induction hypothesis.

Hence it remains to show that $\gamma(T) \geq \frac{|H_2(T)|+|S|}{4}$ when there are no twins and w is not a support vertex. Let T' and T'' be the two sub-trees formed by deleting the edge (w, z) , such that T' contains w and T'' contains z . Note that by our previous analysis, only paths of length exactly 2 can be incident on w , otherwise the tree falls into the previous cases. Denote the number of such paths as r ; clearly, $r \geq 1$ because there is path from x to w . In tree T' , it is clear that by picking all r child nodes of w , we can obtain a optimal dominating set. Since w is not included in the optimal dominating set, $\gamma(T) = \gamma(T') + \gamma(T'') = r + \gamma(T'')$. Let $\psi(T) = |H_2(T)| + |S(T)|$. We have the following three inequalities:

$$1a. H_2(T) \leq H_2(T'') + (r + 1) \quad (\text{if } z \text{ has degree greater than 1 in } T'')$$

$$1b. H_2(T) \leq H_2(T'') + (r + 1) + 1 \quad (\text{if } z \text{ has degree 1 in } T'')$$

$$2. S(T) \leq S(T'') + r$$

Inequality 1a and 1b hold because tree T' contains r vertices that are in $H_2(T')$, and by adding the edge between w and z , w must have degree at least 2. Also, if z has degree 1 in T'' , z is not in $H_2(T'')$, adding an edge between z and w increases z 's degree by one, hence it is in $H_2(T)$, otherwise, z is already in $H_2(T'')$. Similarly, inequality 2 holds because there are r vertices in $S(T')$, adding an edge between w and z does not introduce new support vertices in $S(T)$. Hence, $S(T) \leq S(T'') + r$. This is an inequality because adding the edge (w, z) makes z non-leaf, thus a support vertex might be removed. Therefore, we have $\psi(T) \leq \psi(T'') + 2r + 2$. Therefore:

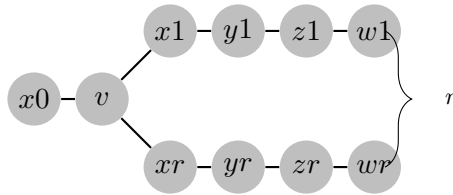


FIGURE 4.2: Tight example of our lower bound on tree domination number

$$\begin{aligned}
\gamma(T) &= \gamma(T'') + r \\
&\geq \frac{|H_2(T'')| + |S(T'')|}{4} + r \text{ [by induction hypothesis]} \\
&= \frac{|H_2(T'')| + |S(T'')| + 4r}{4} \\
&\geq \frac{|H_2(T)| + |S(T)|}{4}
\end{aligned} \tag{4.3}$$

where the last inequality holds because $4r \geq 2r + 2$ when $r \geq 1$, which is always true since there is a path from w to x .

■

The lower bound introduced by Lemma 4.4 is asymptotically tight. Consider the graph shown in Figure 4.2. This graph contains a vertex v with one leaf and r paths of length 4 (P_4) incident on it, which can be also viewed as a star graph with $r + 1$ leaves except r of them are replaced with P_4 . Clearly, by Lemma 4.2, the domination number of this graph is $r + 1$, as there are $r + 1$ support vertices and the set of them dominates all other vertices. Meanwhile, by construction, we have $3r + 1$ vertices of degree two or more, and $|S| = r + 1$. Hence,

$$\frac{|H_2| + |S|}{4} = \frac{4r + 2}{4} \approx r + \frac{1}{2}. \tag{4.4}$$

When $r \rightarrow \infty$, $\frac{r+1}{r+\frac{1}{2}} \rightarrow 1$. Hence, this lower bound is asymptotically tight.

In addition, Lemma 4.4 can be generalized to forest as follows.

Lemma 4.5. *For every forest F without isolated vertices,*

$$\gamma(F) \geq \frac{|H_2| + |S|}{4} \tag{4.5}$$

Proof. The proof is fairly straight forward, as each component of the forest is itself a tree (of size at least 2) and there are no shared edges between them. Hence we know that $|H_2(F)|$ and $|S(F)|$ are equal to the sum of $|H_2(T)|$ and $|S(T)|$ for all $T \in F$. Moreover, $\gamma(F)$ is equal to sum of $\gamma(T)$ for all $T \in F$. Therefore the lemma follows. ■

Combining Lemma 4.3 and 4.5, we have the following theorem:

Theorem 4.6. *For every forest F without isolated vertices, we have:*

$$\frac{|H_2| + |S|}{4} \leq \gamma(F) \leq \frac{|H_2| + |S|}{2}. \quad (4.6)$$

Therefore, $\frac{|H_2| + |S|}{2}$ gives a 2 approximation of the forest domination number.

Moreover, combining the result of Theorem 4.6 and Theorem 4.1, we know that when $3|S| < |H_2|$, the following inequality holds for forests without isolated vertices.

$$\frac{|H_2| + |S|}{4} \leq \frac{|H_2|}{3} \leq \gamma(F) \leq \frac{|H_2| + |S|}{2} \leq \frac{2|H_2|}{3}. \quad (4.7)$$

Although Theorem 4.1 does not generalize to a forest, it is okay here because we only used its lower bound side (i.e., $\frac{|H_2|}{3} \leq \gamma(F)$). Note that this lower bound also holds for trees with order $n \geq 2$. And it suffices to just prove it for trees with order 2, as trees with order greater than 2 is proved in Theorem 4.1. A tree with order 2 is just a path of length two, which has no H_2 vertices but a domination number of 1. Thus, the lower bound holds. Since having multiple disjoint trees increases neither the domination number nor the number of non-leaf vertices, the lower bound can be generalized to forests without isolated vertices. Therefore, we have the following corollary.

Corollary 4.7. *For every forest F without isolated vertices, when $3|S| < |H_2|$, we have*

$$\frac{|H_2|}{3} \leq \gamma(F) \leq \frac{2|H_2|}{3} \quad (4.8)$$

4.1.3 Exact Estimate of Connected Domination Number

Recall that a dominating set is a connected dominating set if the subgraph induced by it is connected, and we use γ_C to denote the connected domination number, the cardinality of the minimum connected dominating set (*MCDS*). Let T be a tree containing $n \geq 2$ vertices, we show that

Theorem 4.8. *For every tree T with $n \geq 2$, $\gamma_C(T) = \max\{1, |H_2|\}$.*

Proof. When $n = 2$, there is a minimum connected dominating set of size one and $|H_2| = 0$, the equality holds.

For $n > 2$, since T is a tree, then $|H_2| \geq 1$, hence it suffices to prove that $MCDS = H_2$. By Lemma 4.2, we can assume there are no leaves in the *MCDS*, then we can prove this theorem via contradiction. Suppose there exists a vertex $v \in H_2$ and $v \notin MCDS$. Firstly, we claim that v can not be a support vertex, otherwise its leaf is undominated, a contradiction. Moreover, if v is not a support vertex, then it must have at least two neighbours in H_2 . This is because if it has a leaf neighbour, it must be a support vertex. If it only has one neighbour in H_2 and no leaf neighbours, then it must be a leaf. Consider removing v from the tree, the tree is divided into two parts, T' and T'' . By the property of tree, every pair of vertices (x, y) such that $x \in T'$ and $y \in T''$ are disconnected, otherwise there is a cycle in the tree. Note that T' and T'' must have at least one vertex in the dominating set, respectively. The dominating set in T' and the dominating set in T'' are disconnected because v is not in the dominating set. Thus, a contradiction. ■

Note this can not be generalized to a forest without isolated vertices, as some components might be P_2 .

4.2 Tree Streaming Algorithms

First note that $|Deg_1|$ and $|H_2|$ could be small (i.e., in $o(n)$), making the simple sampling plus counting technique infeasible here. Take the number of leaves as an example,

assuming its actual count is $\mathcal{O}(k)$, one common technique is to sample $\Omega(\frac{n}{\epsilon^2 k})$ vertices beforehand. Then use one pass to count the number of leaves in the the sampled vertices, and return the estimate as the count number divided by the sampling probability. One can show that the returned result is unbiased and within ϵ factor of the actual number with constant probability. However, k can be as small as a constant, hence the space usage becomes $\Omega(\frac{n}{\epsilon^2})$. We show that by utilizing existing algorithms that calculate L_p norms of a vector, $|H_2|$ and $|Deg_1|$ can be estimated in a one-pass turnstile stream using only $\text{polylog}(n)$ space.

4.2.1 Estimating the Number of Non-leaf Vertices

By estimating the L_0 norm of the node degree vector, we can estimate $|H_2|$. A degree vector of a graph, $\{v_1, \dots, v_n\}$, is a vector with n entries, one for each vertex, and each coordinate value is the degree of that vertex. A graph arriving via the turnstile streaming model can be easily converted to its degree-vector representation. For an edge-arrival stream, we start with $v = \{0\}^n$, and for each edge arrival, (i, j) , we increment the two corresponding entries, v_i and v_j , by one if it is a insertion, otherwise we decrement each by one. Similarly, for vertex-arrival stream, we increment (or decrement) the corresponding coordinate whenever we see a vertex appeared as the neighbour of the newly arrived vertex.

In order to estimate $|H_2|$, we need to consider the number of coordinates that have degree greater than 1. Note that by changing the initialized entries of the vector to -1 , i.e., $v = \{-1\}^n$, at each point in the stream, the value of each entry is equal to its degree in the graph minus 1. Since, by assumption, there are no isolated vertices, hence in the new vector, leaves have coordinate value of 0, and non-leaf vertices have positive coordinate values. As $|L_0|$ represents the number of non-zero coordinates, an algorithm that $(1 \pm \epsilon)$ -approximates $|L_0|$ gives an $(1 \pm \epsilon)$ approximation for $|H_2|$.

However, none of the existing $|L_0|$ sketches support the operation “initialize the vector with $\{-1\}^n$ ”. Hence, we can postpone the degree decrements to a post-processing step. That is, initializing the $|L_0|$ sketch, feeding the stream to the sketch, and decrementing 1 for all vertices. Recall that we can $(1 \pm \epsilon)$ estimate $|L_0|$ in $\mathcal{O}(\epsilon^{-2} \log(n)(\log(1/\epsilon) + \log \log(mM)))$ space and $\mathcal{O}(1)$ update/report time [77]. As the algorithm succeeds with constant probability, we can further boost the success probability to $1 - \delta$ by running $\mathcal{O}(\log(\delta^{-1}))$

independent instances simultaneously and taking the median as the output. To illustrate, assuming the algorithm succeed with probability $\frac{1}{c}$ for some constants c , and the boosting algorithm runs $c \ln(\delta^{-1})$ independent instances concurrently. The boosting algorithm returns the median result, hence it fails if at least half of the instances fail. Since instances are running independently, the fail probability is at most

$$\left(1 - \frac{1}{c}\right)^{c \ln(\delta^{-1})} \leq e^{-\frac{1}{c} c \ln(\delta^{-1})} = e^{\ln \delta} = \delta \quad (4.9)$$

Therefore, we have the following lemma and theorems.

Lemma 4.9. *There is a randomized turnstile streaming algorithm that uses $\mathcal{O}(\text{polylog}(n))$ space and with probability $1 - \delta$, outputs a $(1 \pm \epsilon)$ approximation of $|H_2|$. The update and reporting time is $\mathcal{O}(\log \delta^{-1})$.*

Theorem 4.10. *There is a randomized turnstile streaming algorithm that uses $\mathcal{O}(\text{polylog}(n))$ space and with probability $1 - \delta$, outputs a $(3 \pm \epsilon)$ approximation of the domination number of a tree. The update and report time is $\mathcal{O}(\log \delta^{-1})$.*

Theorem 4.11. *There is a randomized turnstile streaming algorithm that uses $\mathcal{O}(\text{polylog}(n))$ space and with probability $1 - \delta$, outputs a $(1 \pm \epsilon)$ estimate of the connected domination number of a tree. The update and report time is $\mathcal{O}(\log \delta^{-1})$.*

4.2.2 Estimating the Number of Support Vertices

In this section, we present an algorithm that $(1 \pm \epsilon)$ -estimates $|S|$ in a turnstile stream when $|S|$ is no less than a size threshold, K_1 . This algorithm (Algorithm 1) uses $\tilde{\mathcal{O}}\left(\frac{n}{\epsilon^2 K_1}\right)$ bits, where $\tilde{\mathcal{O}}$ suppresses the logarithm factor. And then we present an algorithm that outputs an exact estimate of $|S|$ (as well as $|H_2|$) when $|H_2|$ is less than another size threshold, K_2 . The space usage of the second algorithm is $\tilde{\mathcal{O}}\left(\frac{n}{K_2}\right)$. Both algorithms assume that the set of vertices is known in advance, and the number of edge deletions in the stream is bounded by $\mathcal{O}(n)$.

4.2.2.1 Two-pass $(1 \pm \epsilon)$ Approximation

Lemma 4.12. *Given a size threshold K_1 , a constant error term $\epsilon_1 \in (0, 1)$, and a turnstile stream, when $|S| \geq K_1$, Subroutine 1 outputs an $(1 \pm \epsilon_1)$ approximation of $|S|$.*

This algorithm uses $\tilde{O}(\frac{n}{\epsilon_1^2 K_1})$ space and fails with probability $3e^{-\frac{c_1}{3}}$. Moreover, when $|S| < K_1$, the probability that the returned estimate is larger than $2K_1$ is at most $2e^{-\frac{c_1}{3}}$.

Subroutine 1 Estimating $|S|$

- 1: **Input:** a size threshold K_1 , a large constant c_1 , and an error term ϵ_1
 - 2: **Initialization:** sample $\frac{c_1 n}{\epsilon_1^2 K_1}$ vertices uniformly at random, denote the graph induced by the sampled vertices as I . For each $v \in V(I)$, initialize a empty neighbour list $l(v)$. Set the edge counter m and the size counter t to 0.
 - 3:
 - 4: **First Pass:**
 - 5: **for all** $e = (u, v)$ **do**
 - 6: Increment/Decrement the edge counter m by one
 - 7: **if** $u \in V(I)$ **then**
 - 8: Add/Remove v to/from $l(u)$
 - 9: Increment/Decrement the size counter t by one
 - 10: Perform the same operation on v
 - 11: Abort if $t \geq \frac{2m}{n} \frac{c_1 n}{\epsilon_1^2 K_1} e^{\frac{c_1}{3}}$
 - 12:
 - 13: **Second Pass:**
 - 14: Count the degree of all vertices in $V(I)$ or in $l(w)$ for some $w \in V(I)$
 - 15:
 - 16: **let** $C = \{u \mid u \in V(I), \exists v \in l(u) \text{ s.t. } d(v) = 1\}$
 - 17: **return** $|\hat{S}| = |C| \times \frac{\epsilon_1^2 K_1}{c_1}$
-

Proof. Firstly, we claim that, conditional on the algorithm does not abort at line 11, the candidate set, C , obtained at line 16 contains only vertices from S , and if a vertex in S is sampled, it is kept in C . This is not hard to see, by performing a second pass, we can obtain the exact degree of all remaining vertices and their neighbours. So that a vertex is included in C if there is at least one leaf adjacent to it, which is exactly the definition of S .

It remains to prove the bounds of this algorithm, conditional on the algorithm does not abort. For all $i \in V(G)$, let the binary random variable X_i represent whether a vertex v_i is a support vertex and is sampled in I . Let $X = \sum_{i \in G} X_i$. Clearly, since vertices are sampled uniformly with probability $p = \frac{c_1}{\epsilon_1^2 K_1}$, we have

$$E[X] = \sum_{i \in G} E[X_i] = \sum_{i \in G} \Pr[v_i \in S \cap I] = \sum_{i \in S} \Pr[v_i \in I] = \frac{c_1 |S|}{\epsilon_1^2 K_1} \quad (4.10)$$

Thus by the design of algorithm, $E[|\widehat{S}|] = \frac{\epsilon_1^2 K_1}{c_1} \times E[X] = |S|$. Moreover, when $|S| \geq K_1$, we have

$$\begin{aligned} \Pr [||\widehat{S}| - |S|| \geq \epsilon_1 |S|] &= \Pr [|X - E[X]| \geq \epsilon_1 E[X]] \\ &\leq 2e^{-\frac{\epsilon_1^2 E[X]}{3}} \\ &\leq 2e^{-c_1/3} \end{aligned} \tag{4.11}$$

where we apply the negatively correlated Chernoff bound (Theorem 2.3) in the second-last inequality, since the X_i s are negatively correlated. And the last inequality holds as $|S| \geq K_1$.

Next, we show that when $|S| < K_1$, conditional on the algorithm does not abort, the probability that our returned estimate $|\widehat{S}|$ is larger than $2K_1$ is small. Again, we use the negatively correlated Chernoff bound to prove this

$$\begin{aligned} \Pr [|\widehat{S}| > 2K_1] &= \Pr [X > \frac{2K_1 E[X]}{|S|}] < \Pr [X > (1 + \frac{K_1}{|S|})E[X]] \\ &\leq e^{-\frac{K_1^2 E[X]}{3|S|^2}} \\ &\leq e^{-\frac{c_1 K_1}{3|S|\epsilon^2}} \\ &\leq e^{-\frac{c_1}{3\epsilon^2}} \end{aligned} \tag{4.12}$$

where the first inequality and the last inequality hold because $|S| < K_1$.

Lastly, we bound the space usage of the algorithm and the probability that the algorithm aborts at line 11. Note that by the design of the algorithm, at most $\frac{2m}{n} \frac{c_1 n}{\epsilon_1^2 K_1} e^{\frac{c_1}{3}}$ vertices can be stored from the first pass, otherwise it aborts at line 11. The number of total edge insertions is bounded by $\mathcal{O}(n)$, since we assumed that the number of edge deletions is bounded by $\mathcal{O}(n)$, and it is known that the number of edges of a forest is bounded by $\mathcal{O}(n)$. Thus, we have $\frac{2m}{n} \in \mathcal{O}(1)$. Therefore, since each vertex requires $\mathcal{O}(\log n)$ bits to store its identifier, the space usage of the first pass is $\widetilde{\mathcal{O}}(\frac{n}{\epsilon_1^2 K_1})$. Moreover, in the second pass, each degree counter takes an additional $\mathcal{O}(\log n)$ space to store. Hence, the total space usage is still $\widetilde{\mathcal{O}}(\frac{n}{\epsilon_1^2 K_1})$.

In this algorithm, we sampled $\frac{c_1 n}{\epsilon_1^2 K_1}$ vertices in advance. And the average degree of the graph at any point of the stream can be calculated as $\frac{2m}{n}$. Hence, if we pick a vertex

uniformly at random, the expected number of its neighbours is bounded by $\frac{2m}{n}$. Let Y_i be the number of neighbours of vertex i , $E[Y_i] = \frac{2m}{n}$. Let $Y = \sum_{v \in V(G)} Y_v$. We have

$$E[Y] = \sum_{v \in V(G)} E[Y_v] = \frac{2m}{n} \frac{c_1 n}{\epsilon_1^2 K_1} \quad (4.13)$$

And since we are sampling each vertex uniformly at random, Y_v s are independent of each other. Algorithm aborts if more than $\frac{2m}{n} \frac{c_1 n}{\epsilon_1^2 K_1} e^{\frac{c_1}{3}}$ vertices are retained. By Markov inequality, the probability that the actual size is $e^{\frac{c_1}{3}}$ times larger than the expected size is at most $e^{-\frac{c_1}{3}}$.

Applying a union bound with the abort probability and the concentration probabilities above, we can bound the fail probability of our algorithm as follows. When $|S| \geq K_1$, Algorithm 1 outputs an $(1 \pm \epsilon)$ -estimate of $|S|$ and fails with probability at most $2e^{-\frac{c_1}{3}} + e^{-\frac{c_1}{3}} = 3e^{-\frac{c_1}{3}}$. Similarly, when $|S| < K_1$, the estimate returned by Algorithm 1 is larger than $2K_1$ with probability at most $e^{-\frac{c_1}{3\epsilon_1^2}} + e^{-\frac{c_1}{3}} < 2e^{-\frac{c_1}{3}}$ since $\epsilon_1 < 1$.

■

4.2.2.2 Two-pass Exact Estimate

Consider the degree vector of the graph G , where the value of each coordinate i represents the degree of vertex i in G . Note that similar to the $|H_2|$ approximation, if we decrement the value of each coordinate by 1, each leaf has coordinate value 0. Hence, the resulting degree vector is $|H_2|$ -sparse, i.e., it has at most $|H_2|$ non-zero coordinates. Therefore, when $|H_2|$ is small, we could use this idea and perform a $|H_2|$ -sparse recovery to recover all H_2 vertices. Observe that by doing this, no leaf is sampled, and with probability $1 - \delta$, all H_2 vertices are recovered. Hence, we can assume that vertices that are not sampled are all leaves. Then using a second pass, we could verify whether a sampled vertex is a support vertex or not by telling whether all of its neighbours are in the sampled set or not. However, we have ignored support vertices which are also leaves, which only happens in the case of P_2 (paths of length 2). This is not hard as in P_2 , both endpoints are leaves, hence we can tell whether an edge is indeed a P_2 by verifying whether both endpoints are sampled or not. Lastly, it remains to verify whether $|H_2|$ is indeed small (i.e., whether we have successfully recovered all H_2 vertices). Since the total degree

of a graph G is equal to twice its edge count, we can verify this equation by counting the number of edges, obtaining the degrees of all sampled vertices, and assuming all the non-sampled vertices have degree one. The detailed description of this algorithm is summarized in Algorithm 2.

Subroutine 2 Estimating $|S|$ when $|H_2|$ is small ($|H_2| \leq K_2$)

```

1: Input: a size threshold  $K_2$  and a large constant  $c_2$ 
2: Initialization: The sparse recovery data structure in [37] with  $k = K_2$  and  $\delta = 1/c_2$ 
3:
4: First Pass:
5: for all  $e = (u, v)$ , insertion/deletion do
6:   Update the sparse recovery structure correspondingly
7:
8: for all  $i \in [1 \dots n]$  do
9:   Decrement  $i$  by 1 in the sparse recovery structure
10:
11: Denote the result of sparse recovery as  $H$ 
12: Let  $P_2$  counter  $p = 0$  and a edge counter  $m = 0$ . For  $v \in H$ , initialize a degree
    counter  $d_v = 0$  and leaf counter  $l_v = 0$ 
13:
14: Second Pass:
15: for all  $e = (u, v)$ , insertion/deletion do
16:   increment/decrement  $m$  by one
17:   increment/decrement  $d_i$  by one for both  $i = u$  and  $i = v$ 
18:   if  $u \notin H$  and  $v \notin H$  then
19:     increment/decrement  $p$  by one
20:   if  $u \in H$  and  $v \notin H$  then
21:     increment/decrement  $l_u$  by one
22:   if  $v \in H$  and  $u \notin H$  then
23:     increment/decrement  $l_v$  by one
24:
25: if  $2m \neq n - |H| + \sum_{v \in H} d_v$  then
26:   Return FAIL
27: else
28:   Let  $|\widehat{S}| = |\{u \mid u \in H \text{ and } l(u) = 1\}| + 2p$ 
29:   Let  $|\widehat{H}_2| = |H|$ 
30:   Return  $|\widehat{S}|$  and  $|\widehat{H}_2|$ 

```

Lemma 4.13. *Given a size threshold K_2 , a constant c_2 , and turnstile forest streams, Subroutine 2 uses two passes and outputs the exact estimates for both $|S|$ and $|H_2|$. The fail probability is at most $1/c_2$ when $|H_2| \leq K_2$. The space usage is $\tilde{O}(K_2)$.*

Proof. Note that in the first pass, only vertices in H_2 are sampled. This is because line 8-9 decreases the frequency of all vertices by one, so each leaf has 0 frequency and by

the definition of k -sparse recovery, they are not recovered by the sparse recovery data structure. Moreover, by Theorem 2.8, when $|H_2| \leq K_2$ (i.e., when it is small), the sparse recovery fails with probability no more than $1/c_2$.

Since the sparse recovery data structure might output a false positive result. That is, a k -sparse vector when the original degree vector is not k -sparse. We need to perform a sanity check at line 25 to ensure that the returned result contains all H_2 vertices, rather than part of them. For this sanity check to work, we prove that the equality holds if and only if $H = H_2$.

To begin with, we prove that the equality holds if $H = H_2$. This is trivial as for every graph G , we have the following relationship between its edge count, m , and its total degree count.

$$2m = \sum_{v \in V(G)} d(v) = n - |H_2| + \sum_{v \in H_2} d(v) \quad (4.14)$$

Therefore, if $H = H_2$, the equality in the sanity check must hold.

Next, we prove that the equality does not hold if $H \neq H_2$. Suppose $H \neq H_2$, that is, $\exists x \in H_2$ s.t. $x \notin H$, we have

$$\begin{aligned} 2m - (n - |H| + \sum_{v \in H} d(v)) &= n - |H_2| + \sum_{v \in H_2} d(v) - (n - |H| + \sum_{v \in H} d(v)) \\ &= \sum_{v \in H_2} d(v) - |H_2| - \sum_{v \in H} d(v) + |H| \\ &= \sum_{v \in H_2 \setminus H} d(v) - |H_2| - \sum_{v \in H \setminus H_2} d(v) + |H| \\ &= \sum_{v \in H_2 \setminus H} d(v) - |H_2| - |H \setminus H_2| + |H| \\ &= \sum_{v \in H_2 \setminus H} d(v) - |H_2 \setminus H| \\ &> 0 \end{aligned} \quad (4.15)$$

where the second-last equality holds because by our definition of H_2 , all non-leaf vertices are in H_2 , hence vertices in $H \setminus H_2$ are leaves and have degree of 1. And the last inequality

holds because vertices in H_2 have degree at least 2, and $H_2 \setminus H \neq \emptyset$. Hence, if some H_2 vertices are not sampled, i.e., $H_2 \setminus H \neq \emptyset$, then $2m > n - |H| + \sum_{v \in H} d(v)$. The equality in the sanity check must be evaluated to false, the algorithm returns *FAIL*.

It remains to prove that conditional on all H_2 vertices being sampled, both $|\widehat{S}|$ and $|\widehat{H}_2|$ are exact estimates of $|S|$ and $|H_2|$. If all H_2 vertices are sampled successfully, we can infer whether a vertex is a leaf or not by testing whether it is in H or not. As indicated by line 20-23, by counting the number of leaves, l_v , for each $v \in H$, we could identify all the support vertices in H . But not all support vertices have degree greater than 1, by definition we can have support vertices of degree 1 in path of length 2 (P_2) and each P_2 has two support vertices. Thus, we need to also count the number of P_2 , p . Fortunately this is not hard at all, as if an edge arrives with none of its endpoint in H_2 , this edge must represent a P_2 .

Lastly, the space used by Algorithm 2 is $\widetilde{\mathcal{O}}(K_3)$. In the first pass, the sparse recovery structure takes $\widetilde{\mathcal{O}}(K_3)$ space to store. And in the second pass, for each vertex in S , we introduce two counters that can be both stored in $\mathcal{O}(\log n)$ bits. Therefore the total space usage of Algorithm 2 is $\widetilde{\mathcal{O}}(K_3)$.

■

4.2.3 Estimating Domination Number

Although $|S|$ can not be $(1 \pm \epsilon)$ -approximated when it is small and $|H_2|$ is large, we show that for domination number approximation, as indicated by Corollary 4.7, $|H_2|$ itself is already a good approximation.

Theorem 4.14. *Given an error rate $\epsilon \in (0, 1)$ and a fail rate $\delta \in (0, 1)$, as well as a turnstile stream of a forest, Algorithm 3 uses two passes and outputs a $2(1 \pm \epsilon)$ approximation of the domination number with probability $1 - \delta$. The space usage of the algorithm is $\widetilde{\mathcal{O}}(\sqrt{n})$.*

Proof. To begin with, as shown in Theorem 4.6 and Corollary 4.7, the maximum of our two estimates, $\frac{2|H_2|}{3}$ and $\frac{|H_2|+|S|}{2}$, is guaranteed to be a 2 approximation of the domination number γ . Thus it suffices to show that we could approximate both terms well, or we could approximate the larger term well.

Algorithm 3 Main Algorithm for Estimating γ

1: **Input:** error rate ϵ and fail rate δ
2:
3: **Initialization:** Set $K_1 = \sqrt{n}$, $K_2 = 12\sqrt{n}$, $c_1 = 3 \ln(6\delta^{-1})$, $c_2 = \delta^{-1}$, $\epsilon_1 = \epsilon$
4:
5: Run the following algorithms concurrently:
6: 1. Algorithm for estimating $|H_2|$ with ϵ and $\delta/2$, denote the returned result as $|\widehat{H}_2|$
7: 2. Subroutine 1 with K_1 , c_1 , and ϵ_1 , denote the returned result as $|\widehat{S}|$
8: 3. Subroutine 2 with K_2 and c_2 , denote the returned result as $|\widehat{S}'|$ and $|\widehat{H}_2'|$
9:
10: **if** Subroutine 2 returns *FAIL* **then**
11: **Return** $\widehat{\gamma} = \max\{\frac{2|H_2|}{3}, \frac{|H_2|+|S|}{2}\}$
12: **else**
13: **Return** $\widehat{\gamma} = \max\{\frac{2|\widehat{H}_2'|}{3}, \frac{|\widehat{H}_2'|+|\widehat{S}'|}{2}\}$

When $|H_2| \leq K_2 = 12\sqrt{n}$, by Lemma 4.13, Subroutine 2 succeed with probability $1 - 1/c_2 = 1 - \delta$. Thus, by the design of the algorithm, we return our estimate at line 13. By Lemma 4.13, conditional on successful returning, Subroutine 2 gives exact estimates of $|H_2|$ and $|S|$. Hence we have a 2 approximation of the domination number.

When $|H_2| > 12\sqrt{n}$, we have two cases to consider. If $|S| \geq \sqrt{n}$, then the algorithm might return the estimate of $\frac{2|H_2|}{3}$ or the estimate of $\frac{|H_2|+|S|}{2}$. On the one hand, if the estimate of $\frac{2|H_2|}{3}$ is returned, by Theorem 4.9, we know that the algorithm on line 6 outputs a $(1 \pm \epsilon)$ estimate of $|H_2|$ with probability $1 - \delta/2$. Hence we can obtain a $2(1 \pm \epsilon)$ approximation of the domination number.

On the other hand, if the estimate of $\frac{|H_2|+|S|}{2}$ is returned, by Lemma 4.12, Subroutine 1 gives a $(1 \pm \epsilon)$ estimate of $|S|$ with probability

$$1 - 3e^{-c_1/3} = 1 - 3e^{-\ln(6\delta^{-1})} = 1 - \frac{\delta}{2} \quad (4.16)$$

Apply a union bound over the algorithms for estimating $|S|$ and $|H_2|$, the probability of failure is at most δ . And the error rate is still $1 \pm \epsilon$ as we are summing up the two terms. Hence, the returned estimate is still a $2(1 \pm \epsilon)$ approximation of γ .

Lastly, if $|S| < \sqrt{n}$, then by Lemma 4.12, $|\widehat{S}| \leq 2\sqrt{n}$ with probability $1 - 2e^{-\frac{c_1}{3}}$. By Theorem 4.9, if $\epsilon < 1/2$, then $|\widehat{H}_2| \geq (1 - \epsilon)|H_2| > |H_2|/2$ with probability $1 - \delta/2$. Since $|H_2| > 12\sqrt{n}$, we can bound the probability of $3|\widehat{S}| \leq |\widehat{H}_2|$ as

$$\begin{aligned}
\Pr [3|\widehat{S}| \leq |\widehat{H}_2|] &= \Pr [|\widehat{S}| \leq 2\sqrt{n} \wedge |\widehat{H}_2| > 6\sqrt{n}] \\
&\geq 1 - \Pr [|\widehat{S}| > 2\sqrt{n}] - \Pr [|\widehat{H}_2| \leq 6\sqrt{n}] \\
&\geq 1 - 2e^{-\frac{\epsilon_1}{3}} - \delta/2 \\
&\geq 1 - \delta
\end{aligned} \tag{4.17}$$

where we apply the union bound in the first inequality. Therefore, with probability at least $1 - \delta$, the maximum between the two estimates, $\frac{2|\widehat{H}_2|}{3}$ and $\frac{|H_2|+|S|}{2}$, is $\frac{2|\widehat{H}_2|}{3}$. As shown before, $|\widehat{H}_2|$ is a $1 \pm \epsilon$ estimate of $|H_2|$, hence the returned estimate is a $2(1 \pm \epsilon)$ approximation of γ with probability $1 - \delta$.

The space usage of Algorithm 3 is the maximum space usage of its three sub-algorithms, which is $\tilde{O}(\sqrt{n})$. ■

4.3 Hardness Results

For graphs with bounded arboricity, Chitnis and Cormode [31] provide a lower bound on approximating the domination number.

Theorem 4.15. [31] *For any $\beta \geq 1$, any randomized streaming algorithm that $\frac{\beta}{32}$ -approximates the domination number on graphs of bounded arboricity $\beta+2$ would require $\Omega(n)$ space, i.e., essentially storing the whole graph.*

This lower bound holds for even under the vertex-arrival model.

As for forests without isolated vertices, we can show the space lower bound for streaming algorithms by reducing from the Boolean Hidden Matching Problem [116] using the same reduction technique from Hossein et al. [49]. For simplicity, please refer to [116] and [49] for the definition of Boolean Hidden Matching Problem and the graph construction in reduction. Elad and Wei [116] showed that

Theorem 4.16. [116] *For every randomized protocol for the Boolean Hidden Matching Problem, if the message length is in $o(\sqrt{n})$, then it fails with probability at least $\frac{1}{4}$ on some input.*

Hence, by the same graph construction from [49], we can show that no randomized streaming algorithm can approximate the domination number within a factor better than $3/2$ using only $o(\sqrt{n})$ space and have fail probability less than $\frac{1}{4}$.

Theorem 4.17. *For every forest without isolated vertices, approximating the domination number to a factor better than $3/2$ with probability at least $\frac{3}{4}$ would require $\Omega(\sqrt{n})$ space.*

Chapter 5

Streaming Independent Set

In this chapter, we present our results on the streaming independent set problem. Recall that the independence number of a graph G is lower-bounded by $\sum_{v \in V(G)} \frac{1}{d(v)+1}$, which is also known as the Caro-Wei Bound. In Section 1.2, we have described a permutation-based greedy algorithm that outputs an independent set of size the Caro-Wei Bound in expectation. In Section 5.1, we show that this permutation-based greedy algorithm can be simulated in edge-arrival insertion-only data streams using a small amount of space. More specifically, we show that when the maximum degree is no more than $\Delta \leq \frac{\epsilon^2 n}{3(d+1)^3}$, the Caro-Wei Bound can be $1 \pm \epsilon$ -approximated with probability at least $1 - \delta$ using $\mathcal{O}(\bar{d}\epsilon^{-2} \log n \log \delta^{-1})$ bits. Also, we show that, with little modification, this algorithm can be applied in the *online streaming* model with $\mathcal{O}(\log \epsilon^{-1})$ update time and $\mathcal{O}(\log \epsilon^{-1} \log n \log \delta^{-1})$ working space. Moreover, the maximum degree constraints can be removed in the edge-arrival algorithm by integrating the heavy hitter algorithm into the algorithm with an extra post-processing step. Lastly, we show if the stream is vertex-arrival and random-order, the space can be further reduced.

Next in Section 5.2 and Section 5.3, we present our results on streaming forest independent number approximation. Recall that Chitnis and Krauthgamer [84] showed that the independence number of a tree can be $3/2$ -approximated using the number of leaves, but no streaming algorithm is shown to estimate the number of leaves. We show that by knowing the number of support vertices (vertices adjacent to leaves), we can improve the approximation ratio from $3/2$ to $4/3$. We further show the streaming algorithms to achieve both approximations. The algorithm for the $3/2$ approximation uses one pass

and only logarithm space, while the algorithm for the $4/3$ approximation uses two passes and $\tilde{O}(\sqrt{n})$ space. Both algorithms can be used in edge-arrival turnstile streams with arbitrary ordering.

Lastly, two existing lower bounds on approximating the independence number are shown. And we argue that our first result does not violate the known $\Omega(\bar{d}\epsilon^{-2})$ lower bound on approximating the Caro-Wei Bound [40].

5.1 Sparse Graph Streaming Algorithms

In this section, we show our algorithm and its variants on approximating the Caro-Wei Bound. Let λ be the value of the Caro-Wei Bound. Note that all algorithms in this section are applied to insertion-only model.

5.1.1 An Edge-Arrival Algorithm for Independence Number

With a ϵ -min-wise hash family \mathcal{H} that maps $[n]$ to $[n^3]$ to avoid collisions, our algorithm and our main theorem are presented as Algorithm 4 and Theorem 5.1. We choose the output universe of our hash function to be $[n^3]$ so that the collision (i.e., vertices with the same hash value) happens with low probability (i.e., n^{-c} for some constant c). Since the ϵ -min-wise property is achieved via a $(\log \epsilon^{-1})$ -wise hash family, by the property of k -wise independence with $k \geq 2$, two vertices have the same hash value with probability at most $\frac{1}{n^3}$. Applying an union bound over all pairs of vertices, the probability of having at least one collision is at most $1/n$.

Algorithm 4 Approximating the Caro-Wei Bound

- 1: **Input:** the average degree \bar{d} , an error term ϵ
 - 2: **Initialization:** Randomly select a hash function $h \in \mathcal{H}$. Let $p = \frac{4(\bar{d}+1)}{\epsilon^2 n}$. Sample a set S of vertices of size pn uniformly at random
 - 3:
 - 4: **for all** $e = (u, v)$ **do**
 - 5: **if** $(u \in S) \wedge (h(u) \geq h(v))$ **then**
 - 6: Remove u from S
 - 7: Perform the same operation on v
 - 8:
 - 9: **return** $\hat{\lambda} = |S|/p$
-

Theorem 5.1. *Given the average degree \bar{d} and an error term $0 < \epsilon < 1$, Algorithm 4 is an insertion-only algorithm that outputs an $(1 \pm \epsilon)$ -approximation of the Caro-Wei Bound (λ) in graphs with maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d} + 1)^3}$. It succeeds with probability at least $2/3$ and uses space $\mathcal{O}(\bar{d}\epsilon^{-2} \log n)$.*

To prove Theorem 5.1, we need to show the expectation of $\widehat{\lambda}$ is not far away from λ and the probability that $\widehat{\lambda}$ is too far away from its expectation can be bounded.

Lemma 5.2. $E[\widehat{\lambda}] = (1 \pm \epsilon) \lambda$

Proof. Let binary random variables X_v denote whether vertex v is sampled in S and not removed from S during the stream. That is, $X_v = 1$ if $v \in S$ at the end of the stream, otherwise $X_v = 0$. Let $X = \sum_{v \in V(G)} X_v$, clearly, $X = |S|$ at the end. It is not hard to see that $X_v = 1$ if and only if v is sampled and v has the smallest hash value across $N[v]$. According to the property of ϵ -min-wise hash families, v has the smallest hash value in $N[v]$ with probability $\frac{1 \pm \epsilon}{|N[v]|}$. And by the construction of our algorithm, a vertex v is sampled with probability p . Since these two events are independent, the probability that $X_v = 1$ is $(1 \pm \epsilon) \frac{p}{d(v)+1}$. Hence, by the linearity of expectation, we have

$$E[|S|] = E[X] = E\left[\sum_{v \in V(G)} X_v\right] = \sum_{v \in V(G)} E[X_v] = p \sum_{v \in V(G)} \frac{1 \pm \epsilon}{d(v) + 1} = (1 \pm \epsilon) p \lambda \quad (5.1)$$

Hence, $E[\widehat{\lambda}] = E\left[\frac{|S|}{p}\right] = (1 \pm \epsilon) \lambda$. ■

Next, we need to bound the probability that $\widehat{\lambda}$ is ϵ factor away from its expectation. We persist with the binary random variables from the proof of Lemma 5.2, that is, $X_v = 1$ if $v \in S$ at the end, otherwise $X_v = 0$. Note that not all pairs of X_v and X_u are independent. For all pairs of vertices (e.g., u and v), we have three cases. Firstly, if u and v are adjacent, then X_v and X_u are strongly negatively correlated. That is, if one of X_u and X_v is 1, then the other one must be 0, they cannot be both 1. This is because, since $u \in N[v]$ and $v \in N[u]$, if u has the smallest hash value in $N[u]$, we must have $h(u) < h(v)$, thus it is impossible for v to have the smallest hash value in

$N[v]$. Secondly, if u and v are not adjacent but both adjacent to at least one common vertex (i.e., there are two edges between some paths from u to v), X_v and X_u are positively correlated. Denote one of the common vertices as w , i.e., w is in $N[u] \cap N[v]$. Intuitively, if u has the smallest hash value in $N[u]$, the hash value of w is lower bounded by $h(u)$, hence $h(v)$ is less than $h(w)$ with higher probability. This positive correlation is formalized in the upcoming Lemma 5.3. Lastly, if u and v does not fall into the previous two cases, X_v and X_u are independent. Therefore, we cannot directly apply standard probabilistic concentration bounds (e.g., Chernoff Bound) from Section 2.2 as they all require independence or negative correlation. However, if the variance of $\hat{\lambda}$ is bounded, we can still bound the error rate via Chebyshev's inequality (2.11).

Before bounding the variance of $\hat{\lambda}$, we need to firstly formalize the positive correlation between the X_i s. To begin with, we prove the following purely algebraic Lemma 5.3, which is used in the establish the later positive-correlation proof.

Lemma 5.3. For $k \geq 1$, $\sum_{i=1}^n \frac{(n-i+k)!}{(n-i)!} = \frac{n(n+1)\dots(n+k)}{(k+1)}$

Proof. We prove this by induction on n . Consider the base case where $n = 1$, we have

$$\sum_{i=1}^n \frac{(n-i+k)!}{(n-i)!} = k! = \frac{k!(k+1)}{k+1} = \frac{n(n+1)\dots(n+k)}{(k+1)} \quad (5.2)$$

Assume the equality holds for $n - 1$, and we prove it also holds for n .

$$\begin{aligned} \sum_{i=1}^n \frac{(n-i+k)!}{(n-i)!} &= \sum_{i=2}^n \frac{(n-i+k)!}{(n-i)!} + \frac{(n-1+k)!}{(n-1)!} \\ &= \sum_{i=1}^{n-1} \frac{(n-1-i+k)!}{(n-1-i)!} + \frac{(n-1+k)!}{(n-1)!} \\ &= \frac{(n-1)n \cdots (n+k-1)}{(k+1)} + n(n+1)\dots(n+k-1) \\ &= \frac{(n-1)n \cdots (n+k-1) + n(n+1)\dots(n+k-1)(k+1)}{(k+1)} \\ &= \frac{(n-1+k+1)n \cdots (n+k-1)}{(k+1)} \\ &= \frac{n \cdots (n+k)}{(k+1)} \end{aligned} \quad (5.3)$$

■

And now we formalize the positive correlations. For a vertex v , we use the notation $v < N(v)$ to denote the event that v has smaller hash value (or equivalently, smaller order) among $N[v]$. We have

Lemma 5.4. *Let X and Y be two non-adjacent vertices, where $|N(X) \cap N(Y)| = k$, $|N(X) \setminus N(Y)| = l$, and $|N(Y) \setminus N(X)| = r$. If the vertices are presented in an uniform random order, we have*

$$\Pr[Y < N(Y) \mid X < N(X)] = \frac{(l + r + 2k + 2)}{(r + k + 1)(l + k + r + 2)} \quad (5.4)$$

Proof. Consider $n = l + k + r + 2$ vertices $X, Y, W_1, \dots, W_l, U_1, \dots, U_k$, and V_1, \dots, V_r . Assuming X comes before W_1, \dots, W_l and U_1, \dots, U_k , what is the conditional probability that Y is less than U_1, \dots, U_k and V_1, \dots, V_r . For simplicity, we use W, U , and V to represent $W_1, \dots, W_l, U_1, \dots, U_k$, and V_1, \dots, V_r . Note that X has degree $l + k$, Y has degree $k + r$, X and Y are not adjacent and they share k neighbours (i.e., $N(X) = \{W \cup U\}$ and $N(Y) = \{U \cup V\}$).

Since vertices arrive in a random order, it is equivalent to consider it as a random permutation of the vertex set, and $X < N(X)$ if X comes the first among $N[X]$ in the permutation. Hence, we could prove the condition probability via counting the number of permutations that satisfy $X < N(X)$, as well as the number of permutations that satisfy both the $X < N(X)$ and $Y < N(Y)$. Clearly, there are $n! = (l + k + r + 2)!$ permutations in total. Firstly, we calculate the number of permutations S that have X before W and U . Ignore Y and V , there are $(l + k + 1)!$ different ordering for X, W , and U , and $(l + k)!$ of them start with X . This can be shown by fixing X at front and consider the permutations inside of W and U . Therefore, $\frac{(l+k)!}{(l+k+1)!} = \frac{1}{l+k+1}$ proportion of permutations satisfy the condition that $X < N(X)$, and there are $S = \frac{(l+k+r+2)!}{l+k+1}$ such permutations.

Next, we count the number of permutations which satisfy both conditions, $X < N(X)$ and $Y < N(Y)$. To simplify the counting process, we categorize all valid permutations into the following four cases.

1. X, Y , followed by a mixture of W, V , and U .

2. Y, X , followed by a mixture of W, V , and U .
3. X , followed by some W, Y , followed by a mixture of the rest of W, V , and U .
4. Y , followed by some V, X , followed by a mixture of W , the rest of V , and U .

For the first two cases, since X and Y are fixed, and there is no constraints inside of W, V , and U , thus there are $(l+k+r)!$ different orderings. For the *third* case, one can generate such sequence by fixing X at beginning, then selecting 1 to k elements from W , adding Y , and lastly, adding the mixture of the rest of W (if applicable), U , and V . Assuming the number of W before Y is i , there are $\binom{l}{i}$ choices of W , and for each chosen subset, there are $i!$ orderings of it, and $(l-i+r+k)!$ orderings of the rest of W, U , and V . Hence, for each i , the number of different permutations is $\sum_{i=1}^l \binom{l}{i} i! (l-i+r+k)!$. Simplify it, we have

$$\begin{aligned}
\sum_{i=1}^l \binom{l}{i} i! (l-i+r+k)! &= \sum_{i=1}^l \frac{l!}{i! (l-i)!} i! (l-i+r+k)! \\
&= l! \sum_{i=1}^l \frac{(l-i+r+k)!}{(l-i)!} \\
&= l! \frac{l(l+1) \dots (l+r+k)}{(r+k+1)} \\
&= \frac{l(l+r+k)!}{(r+k+1)}
\end{aligned} \tag{5.5}$$

where the third equality comes from Lemma 5.3.

By similar arguments, one can compute the number of permutations satisfying case 4 as $\sum_{i=1}^r \binom{r}{i} i! (r-i+l+k)! = \frac{r(l+r+k)!}{(l+k+1)}$. Let T be the number of permutations that satisfy both conditions, summing the four cases up, we have

$$\begin{aligned}
T &= 2(l+k+r)! + \frac{l(l+r+k)!}{(r+k+1)} + \frac{r(l+r+k)!}{(l+k+1)} \\
&= \left(2 + \frac{l}{(r+k+1)} + \frac{r}{(l+k+1)}\right) (l+k+r)!
\end{aligned} \tag{5.6}$$

Finally, the conditional probability can be calculated as T/S .

$$\begin{aligned}
\Pr[Y < N(Y) \mid X < N(X)] = T/S &= \frac{(2 + \frac{l}{(r+k+1)} + \frac{r}{(l+k+1)})(l+k+r)!(l+k+1)}{(l+k+r+2)!} \\
&= \frac{(2 + \frac{l}{(r+k+1)} + \frac{r}{(l+k+1)})(l+k+1)}{(l+k+r+1)(l+k+r+2)} \\
&= \frac{2(r+k+1)(l+k+1) + l(l+k+1) + r(r+k+1)}{(r+k+1)(l+k+r+1)(l+k+r+2)} \\
&= \frac{(l+r+k+1)(l+r+2k+2)}{(r+k+1)(l+k+r+1)(l+k+r+2)} \\
&= \frac{(l+r+2k+2)}{(r+k+1)(l+k+r+2)}
\end{aligned} \tag{5.7}$$

■

By Lemma 5.4, it is not surprising that when $k = 0$ (i.e., X and Y do not share any neighbours), $\Pr[Y < N(Y) \mid X < N(X)] = \frac{1}{r+1} = \Pr[Y < N(Y)]$, meaning that $X < N(X)$ and $Y < N(Y)$ are independent. Moreover, if $k > 0$, then we always have $\frac{(l+r+2k+2)}{(r+k+1)(l+k+r+2)} > \frac{1}{r+k+1}$. Thus, we can conclude that $X < N(X)$ and $Y < N(Y)$ are positively correlated if and only if X and Y share at least one common neighbour.

It remains to bound the variance of our output $\hat{\lambda}$. Recall that X_v is a binary random variable indicating whether $v \in S$ at the end of Algorithm 4, p is the proportion of vertices that are sampled in the algorithm, and $X = \sum_{v \in V(G)} X_v$. Note that $\text{Var}(\hat{\lambda}) = \text{Var}(\frac{|S|}{p}) = \text{Var}(X)$. Hence it suffices to bound the variance of X .

Lemma 5.5. *If $\Delta \leq \frac{\epsilon^2 n}{3(d+1)^3}$ and $p \geq \frac{4(\bar{d}+1)}{\epsilon^2 n}$, then*

$$\text{Var}(X) \leq \frac{\epsilon^2 E^2[X]}{3} \tag{5.8}$$

Proof. By our analysis for Lemma 5.2, $\Pr[X_v = 1] = \frac{(1 \pm \epsilon)p}{d(v)+1}$. For simplicity, we ignore the $(1 \pm \epsilon)$ factor in the following variance analysis, that is, $\Pr[X_v = 1] = \frac{p}{d(v)+1}$; the same analysis still holds without removing $(1 \pm \epsilon)$. For each binary random variable X_v , we have

$$\text{Var}(X_v) = E[X_v^2] - E^2[X_v] = \frac{p}{d(v)+1} \left(1 - \frac{p}{d(v)+1}\right) \tag{5.9}$$

Assume non-adjacent vertices i and j share $k \geq 0$ common neighbours. The covariance between two variables X_i and X_j is

$$\begin{aligned}
\text{Cov}(X_i, X_j \mid (i, j) \notin E) &= E[X_i X_j] - E[X_i]E[X_j] \\
&= \Pr[X_i = 1 \wedge X_j = 1] - \Pr[X_i = 1] \Pr[X_j = 1] \\
&= \Pr[X_i = 1 \mid X_j = 1] \Pr[X_j = 1] - \Pr[X_i = 1] \Pr[X_j = 1] \\
&= \frac{p(d(i) + d(j) + 2)}{(d(i) + 1)(d(i) + d(j) - l + 2)} \frac{p}{d(j) + 1} - \frac{p}{d(i) + 1} \frac{p}{d(j) + 1} \\
&= \frac{p^2 l}{(d(i) + d(j) - l + 2)(d(i) + 1)(d(j) + 1)},
\end{aligned} \tag{5.10}$$

where the second-last equality arises from Equation (5.7) with $d(i) = l + k$ and $d(j) = k + r$. It is not hard to see that equation (5.10) is a constraint function subject to $d(i) \geq 1$, $d(j) \geq 1$, and $k \leq \min\{d(i), d(j)\}$. This function has global maximum at $\frac{p^2 k}{(k+1)^2(k+2)}$ when $k = d(i) = d(j)$ for different values of $d(i)$ and $d(j)$, which is maximized at $\frac{p^2}{12}$ when $k = 1$.

And if vertex i and j are adjacent (i.e., $(i, j) \in E(G)$), X_i and X_j are strongly negatively correlated, as $X_i = 1$ implies $X_j = 0$. Hence,

$$\begin{aligned}
\text{Cov}(X_i, X_j \mid (i, j) \in E) &= E[X_i X_j] - E[X_i]E[X_j] = 0 - \frac{p}{d(i) + 1} \frac{p}{d(j) + 1} \\
&= -\frac{p^2}{(d(i) + 1)(d(j) + 1)}
\end{aligned} \tag{5.11}$$

Moreover, let P be the number of vertex pairs that share at least one common neighbour. Assume the vertices are labeled from 1 to n , by variance equation for the sum of correlated variables, we have (for simplicity, we sometimes abbreviate $1 \leq i < j \leq n$ as $i < j$),

$$\begin{aligned}
\text{Var}(X) &= \sum_{i=1}^n \text{Var}(X_i) + 2 \sum_{1 \leq i < j \leq n} \text{Cov}(X_i, X_j) \\
&= \sum_{i=1}^n \frac{p}{d(i)+1} \left(1 - \frac{p}{d(i)+1}\right) + 2 \sum_{1 \leq i < j \leq n} \text{Cov}(X_i, X_j) \\
&\leq \sum_{i=1}^n \frac{p}{d(i)+1} + 2 \sum_{i < j \wedge (i,j) \notin E} \frac{p^2 k}{(k+1)^2 (k+2)} - 2 \sum_{i < j \wedge (i,j) \in E} \frac{p^2}{(d(i)+1)(d(j)+1)} \\
&\leq p\lambda + 2p^2 \frac{P}{12}
\end{aligned} \tag{5.12}$$

Note that for each vertex v with degree d_v , it introduces at most $\frac{d_v(d_v-1)}{2} \leq \frac{d_v^2}{2}$ new pairs of vertices that share a common neighbour (i.e. v). Thus, $P \leq \frac{1}{2} \sum_{v \in V(G)} d_v^2$. Since we assume that $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d}+1)^3}$, we have

$$\begin{aligned}
\sum_{v \in V(G)} d_v^2 &\leq \Delta^2 \frac{\bar{d}n}{\Delta} + (n - \frac{\bar{d}n}{\Delta}) < (\Delta \bar{d} + 1)n \leq \left(\frac{\epsilon^2 n \bar{d}}{3(\bar{d}+1)^3} + 1\right)n < \frac{\epsilon^2 n^2}{3(\bar{d}+1)^2} + n \\
&\leq \frac{\epsilon^2 n^2}{2(\bar{d}+1)^2},
\end{aligned} \tag{5.13}$$

where the last inequality holds when $n \geq \frac{\sqrt{6}(\bar{d}+1)}{\epsilon}$. And the first inequality is because given the maximum degree Δ and a sufficiently large n , the term $\sum_{v \in V(G)} d_v^2$ is maximized when the degree distribution is highly biased. That is, when there are around $\frac{\bar{d}n}{\Delta}$ vertices hitting the maximum degree Δ , while the rest of vertices are having degree 1 (as the graph is assumed to be connected).

Finally, combining Equation (5.12) and (5.13), we have

$$\begin{aligned}
\text{Var}(X) &\leq p\lambda + 2p^2 \frac{P}{12} \\
&\leq p\lambda + p^2 \frac{\epsilon^2 n^2}{24(\bar{d}+1)^2} \\
&\leq p\lambda + p^2 \frac{\epsilon^2 \lambda^2}{24} \\
&= E[X] + \frac{\epsilon^2 E^2[X]}{24} \\
&\leq \frac{\epsilon^2 E^2[X]}{3}
\end{aligned} \tag{5.14}$$

where the third inequality holds because the value of Turán Bound is at most the value of Caro-Wei Bound, i.e., $\frac{n}{d+1} \leq \sum_{v \in V(G)} \frac{1}{d(v)+1} = \lambda$ (see [17] for a proof). The inequality in the second-last line arises from $E[X] = p\lambda$. And the last inequality holds if $E[X] \geq \frac{24}{7\epsilon^2}$, since we have assumed that $p \geq \frac{4(\bar{d}+1)}{\epsilon^2 n}$ and $\lambda \geq \frac{n}{\bar{d}+1}$, $E[X] = p\lambda \geq \frac{4}{\epsilon^2}$, thus the inequality follows. ■

Finally, it is time to prove our main theorem, Theorem 5.1.

Proof of Theorem 5.1. As proved in Lemma 5.2 and Lemma 5.5, the expectation of the returned result $\hat{\lambda}$ is $(1 \pm \epsilon)\lambda$, and $\text{Var}(X) \leq \frac{\epsilon^2 E^2[X]}{3}$. Hence, by Chebyshev's inequality, we have

$$\begin{aligned} \Pr [|\hat{\lambda} - \lambda| \geq 3\epsilon\lambda] &\leq \Pr [|\hat{\lambda} - E[\hat{\lambda}]| \geq \epsilon E[\hat{\lambda}]] \\ &= \Pr [|X - E[X]| \geq \epsilon E[X]] \\ &\leq \frac{\text{Var}(X)}{\epsilon^2 E^2[X]} \\ &\leq \frac{1}{3}, \end{aligned} \tag{5.15}$$

where the first inequality holds because $\epsilon < 1$ so that $(1 + \epsilon)^2 < (1 + 3\epsilon)$ and $(1 - \epsilon)^2 > (1 - 3\epsilon)$. Therefore, the algorithm succeeds with probability at least $2/3$.

The algorithm only keeps a ϵ -min-wise hash function and $\mathcal{O}(\bar{d}\epsilon^{-2})$ vertices. The hash function takes $\mathcal{O}(\log \epsilon^{-1} \log n)$ bits to store, and each vertex v takes $\mathcal{O}(\log n)$ bits to store. Therefore, the total space usage is $\mathcal{O}((\bar{d}\epsilon^{-2} + \log \epsilon^{-1}) \log n) = \mathcal{O}(\bar{d}\epsilon^{-2} \log n)$. ■

Moreover, the success probability of Algorithm 4 can be boosted to $1 - \delta$ by running $2 \log \delta^{-1}$ independent instances concurrently, and return the median of their results. Since every instance is independent of each other, the probability that half of instances fail (i.e., obtain a result that is 3ϵ away from λ) is at most $3^{\log \delta^{-1}} = \delta$. And the space usage of the boosting algorithm is $\mathcal{O}(\bar{d}\epsilon^{-2} \log \delta^{-1} \log n)$.

5.1.2 Modifications to *Online Streaming* Model

Algorithm 4 can be modified to output an actual independent set in the *online streaming* model. As required by the *online streaming* model, our algorithm must maintain a valid independent set in every given point of the stream. Initially, the solution contains all vertices, as no edges have arrived yet, hence the graph is full of isolated vertices. Each time an edge arrives, the algorithm needs to decide which vertex should be removed from the solution to maintain the validity of the solution. And the removed vertex cannot be added back to the solution later (decisions in the *online streaming* model are irrevocable). The solution set can be stored as a n -bit array, such that each index is corresponding with one vertex, and the bit is set to 1 if the solution contains this vertex. We can store the solution set either locally or in a remote server, and our algorithm “reports” its decisions on the server. Either way, we are interested in the external memory usage, that is, the memory used other than to store the solution. Also, for simplicity, we assume that removing an already-removed vertex is valid, and this operation has no effect on the solution. We use working space to refer to the space usage of the algorithm (i.e., space used other than storing the solution set), and update time to refer to the decision time of the algorithm when an edge arrives. Recall that Halldórsson et al. [57] gave an *online streaming* algorithm with $\mathcal{O}(\log n)$ update time and $\mathcal{O}(n)$ working space usage, which reports an independent set of size at least the Caro-Wei Bound in expectation.

Our algorithm is shown in Algorithm 5. Similar to Algorithm 4, this algorithm uses a randomly drawn ϵ -min-wise hash function to obtain a random permutation of vertices, and uses this permutation to indicate whether a vertex should be reserved in the solution. Upon each edge arrival, we compute the hash value (i.e., the order) of the two vertices, $h(u)$ and $h(v)$, if $h(u) \geq h(v)$, then u cannot have the smallest order in $N[u]$, hence it is safe to remove it from the solution. And since there is a solution set and the algorithm reports its decision directly to the solution set, there is no need to maintain a sample set of vertices to help estimate the size of the final solution. The properties of Algorithm 5 is summarized in Theorem 5.6.

Theorem 5.6. *Algorithm 5 is an online streaming algorithm that reports an independent set of size $1 \pm \epsilon$ times the Caro-Wei Bound (λ) in graphs with maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d} + 1)^3}$. It succeeds with probability at least $2/3$, it uses working space $\mathcal{O}(\log \epsilon^{-1} \log n)$ and has update time $\mathcal{O}(\log \epsilon^{-1})$.*

Algorithm 5 Reporting Independent Set with Caro-Wei Bound

-
- 1: **Input:** an error term ϵ , a solution set containing all vertices initially
 - 2: **Initialization:** Randomly select a hash function $h \in \mathcal{H}$
 - 3:
 - 4: **for all** $e = (u, v)$ **do**
 - 5: **if** $h(u) \geq h(v)$ **then**
 - 6: Remove u from the solution
 - 7: **else**
 - 8: Remove v from the solution
-

Proof. Let a binary random variable X_v denote whether v is not removed by Algorithm 5. Since a vertex v is removed if and only if it does not have the smallest hash value across $N[v]$, $\Pr[X_v = 1] = \frac{1+\epsilon}{d(v)+1}$. Let X be the sum of all X_v 's. By the construction of the algorithm, X is exactly the size of the final solution. And the expectation of X is

$$E[X] = \sum_{v \in V(G)} E[X_v] = \sum_{v \in V(G)} \Pr[X_v = 1] = (1 \pm \epsilon) \sum_{v \in V(G)} \frac{1}{d(v) + 1} = (1 \pm \epsilon) \lambda \quad (5.16)$$

Moreover, following the same proof as Lemma 5.5 (note: since we are not sampling vertices, the sampling probability p in Lemma 5.5's analysis can be treated as 1), we can bound the variance of X as $\text{Var}(X) \leq \frac{\epsilon^2 E^2[X]}{3}$. Applying Chebyshev's inequality, we have

$$\Pr[|\hat{\lambda} - \lambda| \geq 3\epsilon\lambda] \leq \frac{1}{3}, \quad (5.17)$$

Hence, the algorithm fails with probability at most $1/3$.

The algorithm only uses a randomly drawn ϵ -min-wise hash function, which can be stored using $\mathcal{O}(\log \epsilon^{-1} \log n)$ bits and has calculation time $\mathcal{O}(\log \epsilon^{-1})$. Hence, the space usage and the update time of Algorithm 5 is $\mathcal{O}(\log \epsilon^{-1} \log n)$ and $\mathcal{O}(\log \epsilon^{-1})$. ■

The success probability can be boosted to $1 - \delta$ by running $\log \delta^{-1}$ independent instances simultaneously and return the maximum result. By the union bound, the probability

that all instances fail is at most $(\frac{1}{3})^{\log \delta^{-1}} = \delta$. After boosting, the space usage becomes $\mathcal{O}(\log \epsilon^{-1} \log \delta^{-1} \log n)$ and the update time becomes $\mathcal{O}(\log \epsilon^{-1} \log \delta^{-1})$.

5.1.3 Removing the Maximum Degree Constraint

Theorem 5.1 holds only when the maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d} + 1)^3}$. We can remove this constraints by finding the heavy hitters (i.e., vertices with very high degree) and removing them from the graph. To illustrate, firstly note that by the Turán Bound, the independence number is at least $\frac{n}{\bar{d}+1}$. Also, given a graph, adding a new vertex to this graph increases its independence number by at most 1, which can be easily proved via contradiction. Hence, if we remove at most $\frac{\epsilon n}{\bar{d}+1}$ vertices from the graph, and the independence number of the remaining graph is at most ϵ -factor less than the independence number in the original graph. That is, let G be a graph and G' be the graph after removing at most $\frac{\epsilon n}{\bar{d}+1}$ vertices from G , we have $(1 - \epsilon)\beta(G) \leq \beta(G') \leq \beta(G)$. If all the high-degree vertices are removed, then our results above can be applied to the remaining graph. The cost of doing this is a slightly worse and biased approximation ratio, plus slightly more space. And the algorithm cannot be adapted to the *online streaming* model as there is a post processing step involved.

To begin with, we formalize our argument about removing high-degree vertices without influencing the independence number too much in Lemma 5.7. Let H_μ be the set of vertices with degree at least μ , and G_{-H_μ} be the graph after removing all vertices in H_μ .

Lemma 5.7. *For every $\mu \geq \frac{\bar{d}+1}{\sqrt{\epsilon}}$, $\lambda(G) \geq \lambda(G_{-H_\mu}) \geq (1 - \epsilon)\lambda(G)$.*

Proof. The first inequality, $\lambda(G) \geq \lambda(G_{-H_\mu})$, clearly holds as removing vertices does not increase the independence number.

Given the average degree \bar{d} , the total degree of all vertices is $\bar{d}n$. By the Pigeonhole Principle, there are at most $\frac{\bar{d}n}{\mu}$ vertices with degree at least μ . Hence, we have

$$\begin{aligned}
\lambda(G_{-H_\mu}) &> \lambda(G) - \sum_{v \in H_\mu} \frac{1}{d(v)+1} \geq \lambda(G) - \frac{1}{\mu+1} \frac{\bar{d}n}{\mu} > \lambda(G) - \frac{\bar{d}n}{\mu^2} \\
&\geq \lambda(G) - \epsilon \frac{\bar{d}n}{(\bar{d}+1)^2} \quad (5.18) \\
&\geq \lambda(G) - \epsilon \frac{n}{(\bar{d}+1)} \\
&\geq \lambda(G) - \epsilon \lambda(G)
\end{aligned}$$

where the first inequality is not a equality because vertices adjacent to H_μ have smaller degree in G_{-H_μ} than in G , hence $\frac{1}{d(v)+1}$ is larger in calculating $\lambda(G_{-H_\mu})$. The last inequality holds because the value of Turán Bound is at most the value of Caro-Wei Bound, i.e., $\frac{n}{\bar{d}+1} \leq \sum_{v \in V(G)} \frac{1}{d(v)+1} = \lambda$ (see [17] for a proof). ■

Our algorithm is presented in Algorithm 6. In this algorithm, we input a set of vertices, R , which contains the set of vertices to be removed. Also, we sample a set of vertices beforehand and use them to estimate the Caro-Wei Bound. Similar to Algorithm 4, a vertex is removed from the sample set if they have hash value greater than some of its neighbours. However, we cannot do that during the stream, as we need to remove R first. Hence, we will be obtain all the neighbours of the sampled vertices, and use a post-processing step to remove R and remove sampled vertices that do not have smallest hash value across its neighbours.

Lemma 5.8. *Given the average degree \bar{d} and an error term $0 < \epsilon < 1$, condition on R containing all vertices with degree at least $\frac{\epsilon^2 n}{3(\bar{d}+1)^3}$ and no vertices with degree less than $\frac{\bar{d}(\bar{d}+1)}{\epsilon}$, Algorithm 6 is an insertion-only algorithm that outputs an $(1-\epsilon)(1 \pm \epsilon)$ -approximation of the Caro-Wei Bound (λ). It succeeds with probability at least $19/30$ and uses space $\mathcal{O}(\bar{d}^2 \epsilon^{-2} \log n)$.*

Proof. By the construction of Algorithm 6, if vertices in R are adjacent to some sampled vertices, they are removed from the neighbourhood list of sampled vertices (line 13 – 14 in the algorithm). Hence, by the condition on line 16 – 17, each sampled vertex is retained in S if and only if it has the smallest hash value in the subgraph after removing R . Let a binary random variable X_v be the indicator of v being sampled and retained in S . That

Subroutine 6 Approximating the Caro-Wei Bound - No Degree Constraint

```

1: Input 1: the average degree  $\bar{d}$ , an error term  $\epsilon$ 
2: Input 2:  $R$ , a set of removed vertices
3:
4: Initialization: Randomly select a hash function  $h \in \mathcal{H}$ . Let  $p = \frac{4(\bar{d}+1)}{\epsilon^2 n}$ . Randomly
   sample a set  $S$  of vertices of size  $pn$ . And for each vertex  $u$  in  $S$ , initialize an empty
   neighbour list  $L_u$ 
5:
6: for all  $e = (u, v)$  do
7:   if  $u \in S$  then
8:     Add  $v$  to  $L_u$ .
9:   Perform the same operation on  $v$ 
10:  Abort if the total size  $\sum_{u \in S} (1 + |L_u|)$  is greater than  $10\bar{d}pn$ 
11:
12: Post-processing:
13: if  $\exists u$  s.t.  $|\{v \mid u \in L_v\}| \geq 1 \wedge u \in R$  then
14:   Remove  $u$  from  $L_v$  for all  $v$ 
15:
16: if  $\exists u$  s.t.  $u \in S \wedge h(u) > h(v)$  for some  $v \in L_u$  then
17:   Remove  $u$  from  $S$ 
18:
19: return  $\hat{\lambda} = |S|/p$ 

```

is, $X_v = 1$ if v is sampled initially, and is not removed by line 16 – 17. Since the sampling procedure and the hash function are independent, $\Pr[X_v = 1] = (1 \pm \epsilon) \frac{p}{d'(v)+1}$, where $d'(v)$ is the degree of v after removing R . Let X be the sum of all such random variables, $\sum_{v \in V(G)} X_v$. Clearly, $X = |S|$ at the end of stream. By the linearity expectation, the expectation of $\hat{\lambda}$ can be calculated as follows.

$$E[\hat{\lambda}] = E[|S|]/p = E[X]/p = \frac{1}{p} \sum_{v \in V(G)} E[X_v] = \sum_{v \in V(G)} \frac{1 \pm \epsilon}{d'(v) + 1} = (1 \pm \epsilon)\lambda', \quad (5.19)$$

where λ' is the Caro-Wei Bound of the subgraph after removing R .

The variance of $\hat{\lambda}$ is the same as the variance of X , as p is fixed. By Lemma 5.5, condition on R contains all vertices with degree at least $\frac{\epsilon^2 n}{3(\bar{d}+1)^3}$, we have $\text{Var}(X) \leq \frac{\epsilon^2 E^2[X]}{3}$. Hence,

$$\Pr[|\hat{\lambda} - \lambda'| \geq 3\epsilon\lambda'] \leq \Pr[|X - E[X]| \geq \epsilon E[X]] \leq \frac{\text{Var}(X)}{\epsilon^2 E^2[X]} \leq \frac{1}{3} \quad (5.20)$$

By Lemma 5.7, condition on R contains no vertices with degree smaller than $\frac{\bar{d}(\bar{d}+1)}{\epsilon}$, $\lambda' \geq (1 - \epsilon)\lambda$. Also, the algorithm aborts if the size of sampled vertices and their neighbourhood lists are too large. Since the average degree is \bar{d} , in expectation, $\bar{d}pn$ vertices are stored by our algorithm. By Markov's inequality, the probability that the actual number of stored vertices is 10 times greater than its expectation, $\bar{d}pn$, is at most $1/10$.

Applying the union bound, the probability that this algorithm fails or aborts is at most $11/30$. Hence, with probability at least $19/30$, Algorithms 6 outputs an $(1 - \epsilon)(1 \pm \epsilon)$ -approximation of the Caro-Wei Bound. The space usage of this algorithm is space to store the hash function, plus the space to store sampled vertices and their neighbours. Because of our constraints on line 10, there are at most $\mathcal{O}(\bar{d}pn) = \mathcal{O}(\bar{d}^2 \epsilon^{-2})$ vertices being stored, where each vertex takes $\mathcal{O}(\log n)$ bits to store. The hash function takes $\mathcal{O}(\log \epsilon^{-1} \log n)$ bits to store. Hence, the total space usage is $\mathcal{O}(\bar{d}^2 \epsilon^{-2} \log n)$. ■

According to Lemma 5.8, the success of Algorithm 6 depends on the quality of provided list of removed vertices, R . By Theorem 2.11 and running the algorithm for heavy hitters with $\epsilon = \phi = \frac{\epsilon^2}{6(\bar{d}+1)^4}$ and $\delta = n^{-c}$ for some constant c , we can obtain the desired removal lists with high probability. The space usage of this heavy hitter instance is $\mathcal{O}(\epsilon^{-1} \log^2 n)$. By running $\log n^c$ (for some constant c) instances of Algorithm 6 concurrently and return the maximum result, the failure probability can be reduce to n^{-c} . Note that there is only one instance of the heavy hitter algorithm, and its result is applied to all instances of Algorithm 6. Applying a union bound over the fail rate of heavy hitter and the fail rate of our concurrent instances, the whole algorithm fails with probability at most n^{-c} . Hence, we have the following Theorem 5.9.

Theorem 5.9. *Given the average degree \bar{d} and an error term $0 < \epsilon < 1$, by running heavy hitters from Theorem 2.11 and $\mathcal{O}(\log \delta^{-1})$ instances of Algorithm 6, we can $(1 - \epsilon)(1 \pm \epsilon)$ -approximate the Caro-Wei Bound (λ) with high probability. The space usage is $\mathcal{O}(\bar{d}^2 \epsilon^{-2} \log^2 n)$.*

5.1.4 Vertex-Arrival Random-Order Algorithm

In previous sections, we use a family of ϵ -min-wise hash functions to generate a nearly random permutation. However, note that if the stream vertex-arrival and random-order, then the arrival order of the vertices is itself a random permutation of vertices. Moreover, by the definition of vertex-arrival stream, each stream element is a new vertex and its neighbourhood list, containing its neighbours that have already arrived before. If we use the arrival order as our random permutation (i.e., vertices which come earlier have smaller order), we can tell if a vertex has the smallest order in its neighbourhood by detecting whether its neighbourhood list is empty. If so, then none of its neighbours have arrived before it, thus they all have a larger order than the current vertex. Hence, it suffices to just keep a counter to count the number of vertices with empty neighbourhood list. Furthermore, we could further reduce the space usage by incrementing the counter with certain probability. Our results are summarized in Algorithm 7 and Theorem 5.10.

Algorithm 7 Approximating the Caro-Wei Bound

```

1: Input: average degree  $\bar{d}$ , an error term  $\epsilon$ 
2:
3: Initialization: A counter  $c = 0$ . Let  $p = \frac{4(\bar{d}+1)}{\epsilon^2 n}$ 
4:
5: for all  $v, N(v)$  in the stream do
6:   if  $N(v)$  is empty then
7:     increase the counter  $c$  by one with probability  $p$ 
8:   Abort the algorithm if  $c > 10np$ 
9:
10: return  $\hat{\lambda} = c/p$ 

```

Theorem 5.10. *Given the average degree \bar{d} and an error term $0 < \epsilon < 1$, Algorithm 7 is an insertion-only random-order vertex-arrival algorithm that outputs an $(1 \pm \epsilon)$ -approximation of the Caro-Wei Bound (λ) in graphs with maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d}+1)^3}$. It succeeds with probability at least $19/30$ and uses space $\mathcal{O}(\log(\bar{d}\epsilon^{-2}))$.*

Proof. Firstly, note that $\hat{\lambda}$ is an unbiased estimator of λ , as each vertex arrives uniformly at random. Let binary random variable X_v denote whether c is incremented when v arrives. $X_v = 1$ if the neighbourhood list of v is empty, and the counter is increased. These two events are independent, the former event happens with probability $\frac{1}{d(v)+1}$, and the latter event happens with probability p . Let $X = \sum_{v \in V(G)} X_v$. By the linearity of expectation, we have

$$E[\widehat{\lambda}] = E[c]/p = E[X]/p = \frac{1}{p} \sum_{v \in V(G)} E[X_v] = \sum_{v \in V(G)} \frac{1}{d(v) + 1} = \lambda \quad (5.21)$$

By Lemma 5.5, the variance of X , $\text{Var}(X) \leq \frac{\epsilon^2 E^2[X]}{3}$. Applying Chebyshev's inequality, we have

$$\Pr[|\widehat{\lambda} - \lambda| \geq \epsilon\lambda] = \Pr[|X - E[X]| \geq \epsilon E[X]] \leq \frac{\text{Var}(X)}{\epsilon^2 E^2[X]} \leq \frac{1}{3}, \quad (5.22)$$

The algorithm might fail if the counter, c , becomes too large so that line 8 is executed. By Markov's inequality,

$$\Pr[X \geq 10np] \leq \frac{E[X]}{10np} = \frac{\lambda}{10n} \leq \frac{1}{10} \quad (5.23)$$

Therefore, applying the union bound, the algorithm succeeds with probability at least $1 - 1/3 - 1/10 = 19/30$.

The space usage of the algorithm is the size of the counter. By the construction of the algorithm, the counter is incremented to at most $10np$, otherwise line 8 is executed. Hence, it can be stored in $\mathcal{O}(\log(np)) = \mathcal{O}(\log(\bar{d}\epsilon^{-2}))$ bits. ■

Similarly, by running $\log \delta^{-1}$ independent instances concurrently and return the maximum result, the success probability can be boosted to $1 - \delta$. This is because the probability that all instances fail is at most $(\frac{11}{30})^{\log \delta^{-1}} = \delta$. The space usage of the new algorithm after boosting is $\mathcal{O}(\log(\bar{d}\epsilon^{-2}) \log \delta^{-1})$.

5.2 Tree Approximation

5.2.1 3/2 Approximation of Independence Number

Theorem 5.11. [84] *For every tree with $n \geq 3$, $\max\{\frac{n}{2}, |Deg_1|\} \leq \beta \leq \frac{n+|Deg_1|}{2}$*

which can be generalized to trees with $n \geq 2$ as:

Theorem 5.12. *For every tree with $n \geq 2$, $\max\{\frac{n}{2}, |Deg_1| - 1\} \leq \beta \leq \frac{n+|Deg_1|}{2}$*

Moreover, given a forest without isolated vertices, if it has $c(F)$ tree components, then we have

Theorem 5.13. *For every forest without isolated vertices,*

$$\max\{\frac{n}{2}, |Deg_1| - c(F)\} \leq \beta \leq \frac{n + |Deg_1|}{2} \quad (5.24)$$

Therefore, any estimation that is between $\max\{\frac{n}{2}, |Deg_1| - c(F)\}$ and $\frac{n+|Deg_1|}{2}$ gives a $3/2$ approximation of the forest independence number.

5.2.2 4/3 Approximation of Independence Number

Before establishing our upper bound and lower bound of β , first we proved the following lemma.

Lemma 5.14. *In any connected graph with $n \geq 3$, there exists at least one maximum independent set containing all leaves and no support vertices.*

Proof. This can be proved easily via deleting and adding vertices in any given maximum independent set. Assume we are given a maximum independent set. If it contains all leaves, then we are done. If some leaves are not included, then their support vertices must be in the independent set. This can be seen via contradiction, assume that both the support vertex and its leaves are not in the independent set, then clearly we could add all the leaves in the set, which violates the assumption that the set is maximum. Note that each support vertex must be adjacent to at least one leaf, and if it is in the independent set, none of its leaves is in the set. Hence, we could remove all support vertices from the given independent set, and add all their leaves into the set, the size of the resulting set does not decrease. Given that the original set is already maximum, the new set is also maximum. ■

Hence in the following section, we assume that the maximum independent set contains every leaf and none of the support vertices (unless $n = 2$).

Lemma 5.15. *For every tree T with $n \geq 2$,*

$$\beta \geq \frac{1}{2}(n + |Deg_1| - |S|) \quad (5.25)$$

Proof. We prove it by showing that there always exist a valid independent set of size $\frac{1}{2}(n + |Deg_1| - |S|)$. To begin with, consider the case where $n = 2$, we can easily verify that $\beta = 1$ and $n = |Deg_1| = |S| = 2$, the inequality holds. Hence, in the following arguments, we assume that $n > 2$.

Rearranging the terms, we have $\frac{1}{2}(n - |Deg_1| - |S|) + |Deg_1|$. This term is equivalent as adding all leaves to the independent set, as well as half of the H_2 vertices that are not support vertices. First of all, by the definition of leaf, all of them are not adjacent to each other when $n > 2$, hence it is valid to add them into the independent set. By doing this, no support vertex can be added into the set as it must be adjacent to at least one leaf. After removing all leaves and support vertices, the remaining graph is a forest with $n - |Deg_1| - |S|$ nodes. Note that none of the vertices in the forest are adjacent to the vertices that are already in the independent set, otherwise they become support vertices. Hence, any independent set of the forest is independent from what we have previously chosen (i.e., the set of all leaves), and their union is still a valid independent set of the original graph. Therefore, it suffices to show that in a forest of size n , there is a independent set of size at least $\frac{n}{2}$. By Theorem 5.12, this holds for any trees with $n \geq 2$.

■

This lower bound is indeed sharp, as any paths of even length, the independence number is equal to $\frac{n}{2}$ and $|Deg_1| = |S|$.

Next we show a similar upper bound in terms of the same quantities.

Lemma 5.16. *For every tree T with $n \geq 2$,*

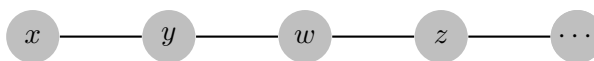
$$\beta \leq \frac{2}{3}(n + |Deg_1| - |S|) \quad (5.26)$$

Proof. We prove this lemma via induction on the number of nodes. We consider all trees with $2 \leq n \leq 4$ as the base cases, which can be easily verified by hand.

Now we assume that the inequality holds for any tree with number of vertices less than n . There are two cases to consider.

Case 1: There exist twins. Recall a pair of vertices is called twins if they are both leaves and share a common parent (support vertex). Let T' be the tree after removing either one of the twins. According to Lemma 5.14, $\beta(T) = \beta(T') + 1$. And since we removed one of the twin, $n_T = n_{T'} + 1$, $|Deg_1(T)| = |Deg_1(T')| + 1$, and $|S(T)| = |S(T')|$. Hence, by the induction hypothesis, this inequality holds.

Case 2: There is no twin. In this case, consider the longest path in T as shown below, denote one of its end point as x , the parent of x as y , the parent of y as w , and the parent of w as z . By the induction hypothesis, T has more than four nodes and no twins, hence z is guaranteed to be non-leaf. This can be seen by contradiction. Assuming z is a leaf and there is more than four nodes, then the path from x to z is the longest, thus the neighbours of w and y can only be leaves. Since we assume there are no twins, w and y both have degree two, there is exactly four nodes, a contradiction. By similar argument, it is not hard to see that y is guaranteed to have degree two.



Case 2.1: $d_w > 2$. When the degree of w is greater than 2, we remove both x and y , denote the resulting tree as T' . Note that this removes a leaf and a support vertex. Moreover, because w has degree more than 2, the removal of x and y does not make w a leaf, and w is a support vertex in T' if and only if it is a support vertex in T . Therefore, $n_T = n_{T'} + 2$, $|Deg_1(T)| = |Deg_1(T')| + 1$, and $|S(T)| = |S(T')| + 1$. Moreover, we claim that $\beta(T) = \beta(T') + 1$. This is because by Lemma 5.14, x is in $\beta(T)$ and y is not in $\beta(T)$, thus the existence of x and y has no impact on whether w and the rest of vertices are in $\beta(T)$ or not (i.e., they are in $\beta(T)$ if and only if they are also in $\beta(T')$). By the induction hypothesis, the inequality holds.

Case 2.2: $d_w = 2$. It remains to show that the inequality holds when $d_w = 2$. Similarly, we have two cases to consider:

Case 2.2.1: z is not a support vertex. In this case, we remove x and y . Similar to case 2.1, we have $n_T = n_{T'} + 2$ and $\beta(T) = \beta(T') + 1$. However, since $d_w = 2$ and z is not a support vertex in T , the removal of x and y makes w a leaf and z a support vertex.

Hence $|Deg_1(T)| = |Deg_1(T')|$, and $|S(T)| = |S(T')|$. By the induction hypothesis, the inequality holds.

Case 2.2.2: z is a support vertex. On the other hand, if z is already a support vertex in T , then consider T' as the tree after removing x , y , and w from T . Since z is a support vertex, then according to Lemma 5.14, z must not be in $\beta(T)$, thus w must be in $\beta(T)$. Therefore, $n_T = n_{T'} + 3$ and $\beta(T) = \beta(T') + 2$. Moreover, removing these three vertices does not make z a leaf or make it not a support vertex in T' , unless z itself has degree 2. In the former case, we have $|Deg_1(T)| = |Deg_1(T')| + 1$, and $|S(T)| = |S(T')| + 1$. And by the induction hypothesis, the inequality holds. In the latter case, we have a path of length 5 (P_5), since x is a leaf, y , w , and z have degree 2, and z is also a support vertex. It can be easily verified by hand that the inequality holds for P_5 , which also could be viewed as a special base case. ■

This upper bound is asymptotically tight. Consider the following graph, which is a star of degree r and replace each leaf with a path of length 3. Clearly this graph is a tree with $3r + 1$ vertices, and $|S| = |Deg_1| = r$. The independence number of this graph is $2r$ as all the r leaves and their r grandparents form the maximum independent set.

Combining Lemma 5.15 and Lemma 5.16, the quantities n , $|Deg_1|$, $|S|$ together give a $4/3$ approximation of the independence number in trees.

Theorem 5.17. *For every tree T with $n \geq 2$,*

$$\frac{1}{2}(n + |Deg_1| - |S|) \leq \beta \leq \frac{2}{3}(n + |Deg_1| - |S|) \quad (5.27)$$

Moreover, these two lemmas can be generalized to forests without isolated vertices, as the independence number of such forest is equal to the sum of the independence number of each its components.

Theorem 5.18. *For every forest F without isolated vertices,*

$$\frac{1}{2}(n + |Deg_1| - |S|) \leq \beta \leq \frac{2}{3}(n + |Deg_1| - |S|) \quad (5.28)$$

In addition, we know that by definition, $n = |H_2| + |Deg_1|$ and $|S| \leq |Deg_1|$. Hence, if we assume $|S| \leq \frac{|H_2|}{2}$, we have the following inequality,

$$|S| = \frac{|S|}{2} + \frac{|S|}{2} \leq \frac{|H_2|}{4} + \frac{|Deg_1|}{2} = \frac{|H_2| + 2|Deg_1|}{4} = \frac{n + |Deg_1|}{4} \quad (5.29)$$

Utilizing Equation (5.29) and combining Theorem 5.12 and Theorem 5.18, the following inequality holds if $|S| \leq \frac{|H_2|}{2}$,

$$\frac{3}{8}(n + |Deg_1|) \leq \frac{n + |Deg_1| - |S|}{2} \leq \beta \leq \frac{n + |Deg_1|}{2} \leq \frac{2}{3}(n + |Deg_1| - |S|) \quad (5.30)$$

Note that the upper bound of Theorem 5.12 also holds for forests without isolated vertices. Therefore, we have the following corollary that is also a $4/3$ approximation of β when $|S| \leq \frac{|H_2|}{2}$.

Corollary 5.19. *In every forest F without isolated vertices, if $|S| \leq \frac{|H_2|}{2}$, then*

$$\frac{3}{8}(n + |Deg_1|) \leq \beta \leq \frac{n + |Deg_1|}{2} \quad (5.31)$$

5.3 Tree Streaming Algorithms

The algorithms for estimating $|S|$ have been presented and proved in Section 4.2. Hence, here we only demonstrate our approach for estimating $|Deg_1|$ and the number of tree components.

5.3.1 Estimating the Number of Leaves

Note that $|Deg_1|$ (i.e., the number of leaves) cannot be estimated by simply subtracting the approximation of $|H_2|$ from n , as $|Deg_1|$ could be in $o(|H_2|)$. Also the vector representation and L_0 norm does not quite work here. Therefore, we start by considering the following lemma.

Lemma 5.20. *In a tree T with $n \geq 2$ and maximum degree Δ ,*

$$|Deg_1(T)| = \Delta + \sum_{i=3}^{\Delta} (|H_i| - 1) = 2 + \sum_{i=3}^{\Delta} |H_i| \quad (5.32)$$

Proof. We prove the lemma by induction. Without loss of generality, we assume that the graph is always tree and the number of vertices n is at least 2.

To begin with, let us consider the base case where $n = 2$: it is essentially a pair of vertices with one edge between them. By definition, $|Deg_1| = 2$, all vertices have degree smaller than 2, the equality holds.

Assume the equation holds for the tree with n , and we are adding a new node, u_{n+1} , to it. The newly added node must be leaf, otherwise it creates a cycle in the tree. There are two cases to consider.

Firstly, when the new leaf is added to an existing leaf node u_l , the equation holds. This is because by adding a node to u_l , u_l has degree 2 and is not a leaf anymore. As u_{n+1} is a newly introduced leaf, $|Deg_1|$ stays the same. Meanwhile, for $3 \geq i \geq \Delta$, $|H_i|$ stays the same.

Another case is that the new leaf is introduced at a non-leaf node u_o . In this case, $|Deg_1|$ increments by one, since no leaf is removed. Assume previously u_o has degree d , after adding a new leaf, $d' = d + 1$, thus making $|H_{d'}|$ increase by one. Note that $|H_i|$ stays the same for all $i \geq 2, i \neq d'$. Therefore, both sides of the equation increase by one, and the equality holds. ■

Lemma 5.20 can be generalized to a forest without isolated vertices as follows.

Corollary 5.21. *If forest F without isolated vertices has maximum degree of Δ and $c(F)$ tree components, then*

$$|Deg_1(F)| = 2c(F) + \sum_{i=3}^{\Delta} |H_i| \quad (5.33)$$

Proof. This equality holds for each individual tree component, and the combination of them does not change the degree of any vertices. Hence

$$|Deg_1(F)| = \sum_{T \in F} |Deg_1(T)| = \sum_{T \in F} 2 + \sum_{i=3}^{\Delta} |H_i(T)| = 2c(F) + \sum_{i=3}^{\Delta} |H_i(F)| \quad (5.34)$$

■

By Corollary 5.21, we have

$$|Deg_1| = 2c(F) + \sum_{i=3}^{\Delta} |H_i| = 2c(F) + \sum_{i=3}^{\Delta} ((i-2)|Deg_i|) \quad (5.35)$$

Recall that if we map the degree of each vertex onto a vector v of length n (i.e., $v = \{0\}^n$), where a coordinate value $|v_i|$ represents a vertex's degree, then the L_1 norm of this degree vector is

$$L_1 = \|v\| = \sum_{i=1}^n |v_i| = \sum_{i=1}^{\Delta} i |Deg_i| \quad (5.36)$$

Thus, if we subtract every coordinate on the degree vector by 2 (which is essentially the same as initializing the vector as $\{-2\}^n$ rather than $\{0\}^n$), we have

$$L_1 = \|v\| = \sum_{i=1}^n |v_i - 2| = |Deg_1| + \sum_{i=3}^{\Delta} (i-2) |Deg_i| \quad (5.37)$$

Combining Equation (5.35) and (5.37), $|Deg_1|$ can be expressed as

$$|Deg_1| = \frac{L_1}{2} + c(F) \quad (5.38)$$

Hence, in order to estimate $|Deg_1|$, we can estimate the number of tree components and the L_1 norm of the degree vector “starting with $\{-2\}^n$ ”. Similarly, since the L_1 sketches do not support the operation “start with $\{-2\}^n$ ”, we can postpone the degree decrements to a post-processing step. The number of tree components can be obtained exactly by keeping a counter m counting the number edges, and returning $c(F) = n - m$. By the

property of a forest, it is an exact estimate of $c(F)$. Also, by Theorem 2.5, $|L_1|$ can be $(1 \pm \epsilon)$ -approximated using $\mathcal{O}(\epsilon^{-2} \log(mM))$ bits of space. The success probability of the L_1 approximation algorithm can be increased to $1 - \delta$ by running $\mathcal{O}(\log \delta^{-1})$ independent instances simultaneously and returning the median of them, which introduces an extra factor of $\mathcal{O}(\log \delta^{-1})$ in both space and time.

Lemma 5.22. *Given a forest stream and $\epsilon, \delta \in (0, 1)$, there is a randomized turnstile streaming algorithm that uses $\mathcal{O}(\text{polylog}(n))$ space and, with probability $1 - \delta$, outputs a $1 \pm \epsilon$ approximation of $|Deg_1|$.*

Combining Theorem 5.13 and Lemma 5.22, we have

Theorem 5.23. *There is a randomized turnstile streaming algorithm that uses $\mathcal{O}(\text{polylog}(n))$ space and with probability $1 - \delta$, outputs a $3/2$ approximation of the optimal independent set size of a forest.*

5.3.2 Estimating Independence Number

The main algorithm for approximating the forest independence number is similar to the main algorithm for forest domination number approximation (Algorithm 3). We run several subroutines concurrently and return the minimum between $\frac{3(n+|Deg_1|)}{8}$ and $\frac{n+|Deg_1|-|S|}{2}$ as our estimate.

Algorithm 8 Main Algorithm for Estimating β

- 1: **Input:** error rate ϵ and fail rate δ
 - 2:
 - 3: **Initialization:** Set $K_1 = \sqrt{n}$, $K_2 = 8\sqrt{n}$, $c_1 = 3 \ln(6\delta^{-1})$, $c_2 = \delta^{-1}$, $\epsilon_1 = \epsilon/2$
 - 4:
 - 5: Run the following algorithms concurrently:
 - 6: 1. Algorithm for estimating $|Deg_1|$ with $\epsilon/2$ and $\delta/2$, denote the returned result as $|\widehat{Deg}_1|$
 - 7: 2. Subroutine 1 with K_1 , c_1 , and ϵ_1 , denote the returned result as $|\widehat{S}|$
 - 8: 3. Subroutine 2 with K_2 and c_2 , denote the returned result as $|\widehat{S}'|$ and $|\widehat{H}_2'|$
 - 9:
 - 10: **if** Subroutine 2 returns *FAIL* **then**
 - 11: **Return** $\widehat{\beta} = \min\left\{\frac{3(n+|\widehat{Deg}_1|)}{8}, \frac{n+|\widehat{Deg}_1|-|\widehat{S}|}{2}\right\}$
 - 12: **else**
 - 13: Let $|\widehat{Deg}_1'| = n - |\widehat{H}_2'|$
 - 14: **Return** $\widehat{\beta} = \min\left\{\frac{3(n+|\widehat{Deg}_1'|)}{8}, \frac{n+|\widehat{Deg}_1'|-|\widehat{S}'|}{2}\right\}$
-

Theorem 5.24. *Given error rate and fail rate $\epsilon, \delta \in (0, 1)$, as well as a turnstile forest stream, Algorithm 3 uses two passes and outputs a $4/3(1 \pm \epsilon)$ approximation of the independence number with probability $1 - \delta$. The space usage of the algorithm is $\tilde{O}(\sqrt{n})$.*

Proof. As proved in Theorem 5.18 and Corollary 6.7, the minimum between our two estimates, $\frac{3(n+|Deg_1|)}{8}$ and $\frac{n+|Deg_1|-|S|}{2}$, is guaranteed to be a good $4/3$ approximation of the independence number β . Hence, it suffices to show that we could either approximate both terms well, or approximate the larger term well.

When $|H_2| \leq K_2 = 8\sqrt{n}$, the algorithm returns at line 14. By Lemma 4.13, Subroutine 2 gives exact estimates of $|H_2|$ and $|S|$ with probability $1 - 1/c_2 = 1 - \delta$. Also, since $n = |H_2| + |Deg_1|$ and n is known, we can obtain an exact estimate of $|Deg_1|$. Hence, no matter which estimate is returned, it is a $4/3$ approximation of the independence number.

When $|H_2| > 8\sqrt{n}$, we have two cases to consider. If $|S| \geq \sqrt{n}$, our algorithm may return the estimate of $\frac{3(n+|Deg_1|)}{8}$ or the estimate of $\frac{n+|Deg_1|-|S|}{2}$ (line 11). On the one hand, If the estimate of $\frac{3(n+|Deg_1|)}{8}$ is returned, by Lemma 5.22 we know that the algorithm on line 6 outputs a $(1 \pm \frac{\epsilon}{2})$ estimate of $|Deg_1|$ with probability $1 - \delta/2$. Thus our result is a $4/3(1 \pm \epsilon/2)$ approximation of β .

On the other hand, if the estimate of $\frac{n+|Deg_1|-|S|}{2}$ is returned, by Lemma 4.12, Subroutine 1 gives a $(1 \pm \frac{\epsilon}{2})$ estimate of $|S|$ with probability

$$1 - 3e^{-c_1/3} = 1 - 3e^{-\ln(6\delta^{-1})} = 1 - \frac{\delta}{2}. \quad (5.39)$$

Applying a union bound over the algorithm for estimating $|S|$ and the algorithm for estimating $|Deg_1|$, the probability of failure is at most δ . And on the upper-bound side, the approximation ratio is at most

$$\begin{aligned} \frac{1}{2}(n + (1 \pm \frac{\epsilon}{2})|Deg_1| - (1 \pm \frac{\epsilon}{2})|S|) &\leq \frac{1}{2}(n + |Deg_1| - |S| + \frac{\epsilon}{2}(|Deg_1| + |S|)) \\ &\leq (1 + \epsilon) \frac{1}{2}(n + |Deg_1| - |S|) \end{aligned} \quad (5.40)$$

where the last inequality holds because $|S| \leq |Deg_1| \leq n$, thus $|Deg_1| + |S|$ is no more than $2(n + |Deg_1| - |S|)$. Similarly, we prove the approximation ratio on the lower bound as

$$\frac{1}{2} (n + (1 \pm \frac{\epsilon}{2})|Deg_1| - (1 \pm \frac{\epsilon}{2})|S|) \geq (1 - \epsilon) \frac{1}{2} (n + |Deg_1| - |S|) \quad (5.41)$$

Hence, our algorithm returns a $4/3(1 \pm \epsilon)$ approximation of the independence number.

Lastly, if $|S| < \sqrt{n}$, then by Lemma 4.12, $|\widehat{S}| \leq 2\sqrt{n}$ with probability $1 - 2e^{-\frac{c_1}{3\epsilon^2}}$. Also by Theorem 4.9, we know that $|\widehat{H}_2| \geq (1 - \epsilon)|H_2| > |H_2|/2$ holds with probability $1 - \delta/2$ (if $\epsilon < 1/2$). Since $|H_2| \geq 8\sqrt{n}$, by applying a union bound, we can claim that the probability of $2|\widehat{S}| > |\widehat{H}_2|$ is at most $2e^{-\frac{c_1}{3\epsilon^2}} + \delta/2 \leq \delta$. Therefore, with probability $1 - \delta$, the minimum between the two estimates is $\frac{3(n + |\widehat{Deg}_1|)}{8}$. As shown before, $|\widehat{Deg}_1|$ is a $1 \pm \epsilon$ estimate of $|Deg_1|$, hence the return estimate is a $4/3(1 \pm \epsilon)$ approximation of β with probability $1 - \delta$.

The space usage of Algorithm 8 is the maximum space usage of its three sub-algorithms, which is $\tilde{O}(\sqrt{n})$. ■

5.4 Hardness Results

Cormode et al. [40] have proved a lower bound for approximating the Caro-Wei Bound. They showed that every randomized algorithm with constant error probability requires $\Omega(\frac{n}{\lambda_L c^2 p})$ space to c -approximate the Caro-Wei Bound, where λ_L is a known lower bound of the Caro-Wei Bound and p is the number of passes allowed. By using the Turán Bound as the lower bound, we have

Theorem 5.25. [40] *Every one-pass streaming algorithm that approximate the independence number with the $\bar{d}(1 \pm \epsilon)$ approximation ratio would require $\Omega(\frac{\bar{d}}{\epsilon^2})$ space.*

But we also remark that the above lower bound only applies to graphs with average degree at least $\frac{16n-28}{49}$. Recall that in Cormode et al. [40], they prove the lower bound via reduction from the set disjointness problem on the universe of $[k]$, such that

they construct a graph based on the sets from Alice and Bob, and an algorithm that c -approximates the Caro-Wei Bound on the constructed graph solves the set disjointness problem. In their construction, $k + 3$ groups of vertices are created, and they denote them as A, B, C, U_1, \dots, U_k . Moreover, for an arbitrary integer z , they let $q = 2zc^2$ and $a = kq$. And they have $|A| = |B| = a$, $|C| = z$, and $|U_i| = q$ for all $i \in [k]$, thus $n = 3a + z = z(6kc^2 + 1)$. The construction can be summarized as follows. Firstly, they add edges between every pairs of vertices from $A \cup B$ to make $A \cup B$ a large clique. And then they add edges between every pairs of $u \in U_i$ and $v \in A$ if Alice's set contains i . Similarly, when the algorithm state is passed to Bob, edges are added between every pairs of $u \in U_i$ and $v \in B$ if Bob has i . Since $A \cup B$ is a clique in all situations, we have

$$\bar{d} \geq \frac{2a(2a - 1)}{n} = \frac{2(n - z)(2(n - z) - 3)}{9n} \geq \frac{\frac{12n}{7}(\frac{12n}{7} - 3)}{9n} = \frac{16n - 28}{49}, \quad (5.42)$$

where the last inequality holds because $z \leq \frac{n}{7}$ when both c and k are at least 1.

For forest without isolated vertices, Esfandiari et al. [49] established a lower bound for the maximum matching problem by reduction from the Boolean Hidden Matching problem. Their reduction technique also indicates that no randomized streaming algorithm can approximate the independence number within a factor better than $4/3$ in only $o(\sqrt{n})$ space.

Theorem 5.26. *For any forests without isolated vertices, any randomized algorithms that approximate the independence number to a factor better than $4/3$ require $\Omega(\sqrt{n})$ space.*

Chapter 6

Other Streaming Graph Problems

Support vertices can be useful in understanding and approximating other graph problems. In this section, we show another application of the it in matching and vertex cover. A set of edges is a matching if none of them share a common vertex, and a set of vertices is a vertex cover if for each edge, at least one of its endpoint is in the set. Both problems have wide applications and have been extensively studied in the data streams model [25, 26, 32, 33, 39, 49]. It is known that by the König-Egeváry theorem, in trees and forests, the size of the maximum matching (i.e., matching number) is equal to the size of the minimum vertex cover (i.e., covering number). Hence, it suffices to just demonstrate our result on matching number.

For the matching number in tree (or forest) streams, previous results showed various 2-approximation algorithms [25, 39, 49]. We show that, with additional knowledge on the number of support vertices, the approximation ratio can be improved to 3/2. To begin with, we prove that $\frac{|H_2|+|S|}{2}$ is a 3/2-approximation of the matching number. And then we show how our streaming algorithms for the domination number can be migrated here to help approximate the matching number.

6.1 Trees Approximation

6.1.1 2 Approximation of Matching and Vertex Cover

Previous studies [25, 39, 49] have shown that the matching number can be $1 \pm \epsilon$ approximated in tree or forest streams using various graph quantities. For example, in an edge-arrival stream, we say an edge, e , is an E_1 edge if for every edge arrives after e , it does not share a common vertex with e . Cormode et al. have used $|E_1|$ as their 2 approximation estimate for trees. Other works [25] [49] have used the number of the non-leaf vertices and the number of tree components to estimate the matching number in forests. Let ϕ be the matching number, the size of the maximum matching, $c(F)$ be the number of tree components in a forest, and H_2 be the set of non-leaf vertices. They showed that

Theorem 6.1. [25] [49] *In forests without isolated vertices, we have*

$$\max\{c(F), \frac{|H_2| + c(F)}{2}\} \leq \phi \leq |H_2| + c(F) \quad (6.1)$$

6.1.2 3/2 Approximation of Matching and Vertex Cover

We show that by incorporating the support vertices into the estimate, we can further improve the approximation ratio to $3/2$. Before establishing our upper bound and lower bound on ϕ , first we prove the following lemma.

Lemma 6.2. *In any connected graph with $n \geq 3$, there exists a maximum matching such that every support vertex is matched, and its matching edge is between it and one of its leaf.*

Proof. The first half of the lemma (every support vertex is matched) can be proved via contradiction. Assume we have a maximum matching and a support vertex s is not matched, then we could include an edge between s and one of its leaves into the matching, a contradiction.

The later half can be proved via deleting and adding edges in any existing maximum matching. Assume we are given a maximum matching M . If M satisfies the condition,

then we are done. If there is a support vertex s such that it is matched through an edge between it and a non-leaf vertex, then none of its leaves are matched. This is because the only way to make a leaf matched is by adding its incident edge into the matching. And by the definition of matching, edges in the matching cannot share a common vertex. Hence, we can remove the matching edge of s and include one of the edges between s and its leaves into the matching. This is valid as the leaf is not matched before, and s is matched only by the newly added edge. The size of maximum matching does not decrease. ■

We may now prove an upper bound on ϕ ,

Lemma 6.3. *For every tree T with $n \geq 2$,*

$$\phi \leq \frac{|H_2| + |S|}{2} \tag{6.2}$$

Proof. We show that the size of the maximum matching is at most $\frac{|H_2| + |S|}{2}$. When $n = 2$, we have $\phi = 1$ and $|H_2| = |S| = 2$, the inequality holds. Hence, in the following arguments, we assume that $n > 2$.

Rearranging the terms, we have $|S| + \frac{|H_2| - |S|}{2}$. By Lemma 6.2, all support vertices are matched by the edges between them and one of their leaves. Hence, removing all support vertices and their leaves removes $|S|$ edges in the matching. The remaining graph is a forest F with $n(F) = |H_2| - |S|$, and all edges in F can be added into the matching as they do not share a common endpoint with edges between support vertices and leaves (i.e., the matching edges that are removed). Hence it remains to prove that for every forest, there is a matching of size at most $\frac{n(F)}{2}$. Since every forest is bipartite, and in a bipartite graph with total order of n , the matching number is at most $\frac{n}{2}$. This is because one side has at most $\frac{n}{2}$ nodes. By the Pigeonhole Principle, if there is a matching of size greater than $\frac{n}{2}$, at least one node has more than one matching edges incident on it, a contradiction. ■

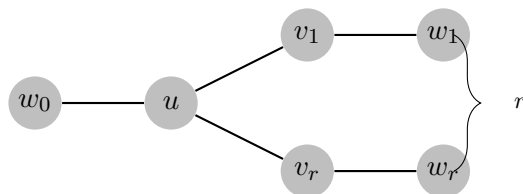


FIGURE 6.1: Sharp example of our upper bound on tree matching number

This upper bound is sharp. For example, Figure ?? has the matching number equals exactly to $\frac{|H_2|+|S|}{2}$. It is constructed by taking a star graph H with center, u , and r leaves, v_1, \dots, v_r , then for each vertex in H , attaching a new leaf w_i on it. Denote the constructed graph as G . Clearly, G has $|H_2| = |S| = r + 1$, as all vertices in H become a support vertex and have more than 1 neighbour. And it is not hard to see that $\phi = r + 1$, as the maximum matching is the edge set $E(G) \setminus E(H)$.

As for the lower bound, we have

Lemma 6.4. *For every tree T with $n \geq 2$,*

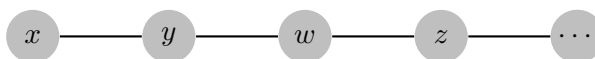
$$\phi \geq \frac{|H_2| + |S|}{3} \quad (6.3)$$

Proof. We prove Lemma 6.4 via induction on the number of nodes n . We consider all trees with $2 \leq n \leq 4$ as the base cases, which can be easily verified by hand.

Now we assume that the inequality holds for every tree fewer than n vertices. There are two cases to consider.

Case 1: There exists a twin. Recall a pair of vertices is twin if and only if they are both leaves and they share a common support vertex, s . Remove either one of the twin and let T' be the tree after the removal. Note that after doing this, s becomes a leaf if and only if s has degree two, in this case $n = 3$, which falls into our base case. Hence, both $|H_2|$ and $|S|$ stays the same after the removal. Moreover, we claim that the matching number also stays the same. This is because only one of s 's edges can be included into matching, if the removed leaf is on this edge, then we could add another edge between s and any one of its remaining leaves. Therefore, by the induction hypothesis, this inequality holds.

Case 2: There are no twins. In this case, consider the longest path in T as shown below, denote one of its endpoint as x , the parent of x as y , the parent of y as w , and the parent of w as z . By the induction hypothesis, T has more than 4 nodes and no twins, hence z is guaranteed to be a non-leaf. This can be seen by contradiction. Assuming z is leaf and there are more than 4 nodes, since the path from x to z is the longest, then the neighbours of w and y can only be leaves. This is because if w has another path of length 2, the path from x to z is not the longest path in T , violating our choice at the beginning. Also, since we assume there is no twins, w and y are both have degree 2, thus there is exactly 4 nodes, a contradiction. By similar argument, it is not hard to see that y is guaranteed to have degree 2.



Case 2.1: $d_w > 2$. When w has degree greater than 2, we remove both x and y , denote the resulting tree as T' . Since w has degree greater than 2, removing x and y does not make it a leaf, thus $|H_2(T)| = |H_2(T')| + 1$. Also, w is a support vertex if and only if it is a support vertex in T , we have $|S(T)| = |S(T')| + 1$. By Lemma 6.2, we know that $e(x, y)$ is in the maximum matching of T , but not $e(y, w)$. Hence, the removal of x and y does not change the matching status of vertices in T' , $\phi(T) = \phi(T') + 1$. By the induction hypothesis, the inequality holds.

Case 2.2: $d_w = 2$. It remains to show that the inequality holds when $d_w = 2$. Similarly, we have two cases to consider:

Case 2.2.1: z is not a support vertex. In this case, we remove x and y . Similar to case 2.1, we have $\phi(T) = \phi(T') + 1$. However, since $d_w = 2$ and z is not a support vertex in T , the removal of x and y makes w a leaf and z a support vertex. Hence $|H_2(T)| = |H_2(T')| + 2$, and $|S(T)| = |S(T')|$. By the induction hypothesis, the inequality holds.

Case 2.2.2: z is a support vertex. On the other hand, if z is already a support vertex in T , then consider T' as the tree after removing x , y , and w from T . Since z is a support vertex, then according to Lemma 6.2, z must be matched and $e(x, y)$ is in the maximum matching. Hence, we know that $e(y, w)$ and $e(w, z)$ are not in the matching, and the removal of x , y , and w does not change the matching status of vertices in T' .

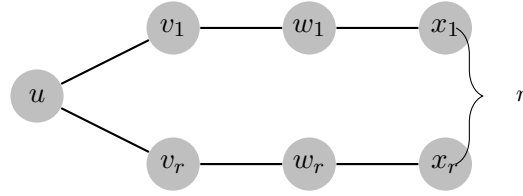


FIGURE 6.2: Sharp example of our upper bound on tree matching number

Therefore, $\phi(T) = \phi(T') + 1$. Moreover, removing these three vertices does not make z a leaf or make it not a support vertex in T' , unless z itself has degree 2. In the former case, we have $|H_2(T)| = |H_2(T')| + 2$, and $|S(T)| = |S(T')| + 1$. And by the induction hypothesis, the inequality holds. In the later case, we have a path of length 5 (P_5), since x is a leaf, y , w , and z have degree 2, and z is also a support vertex. It can be easily verified by hand that the inequality holds for P_5 , which also could be viewed as a special base case.

■

The lower bound is asymptotically tight. Consider the graph G shown in Figure 6.2. G is constructed by starting with a star graph, H , with center u and r leaves, v_1, \dots, v_r , and we replace each leaf with a path of length 3. G has $2r + 1$ non-leaf vertices, including the center u and two of the vertices on each of the paths (i.e., v_1, \dots, v_r and w_1, \dots, w_r). Also $|S(G)| = r$ as only the middle vertices (i.e., w_1, \dots, w_r) on the paths are support vertices. Lastly, a maximum matching can be found by adding all edges between support vertices (i.e., w_1, \dots, w_r) and leaves (i.e., x_1, \dots, x_r), and one edge from the edges incident on center u , thus $\phi = r + 1$.

Combining to Lemma 6.3 and Lemma 6.4, we have the following theorem.

Theorem 6.5. *For every tree T with $n \geq 2$,*

$$\frac{|H_2| + |S|}{3} \leq \phi \leq \frac{|H_2| + |S|}{2} \quad (6.4)$$

Also, Theorem 6.5 can be extended to forests without isolated vertices. This is because in a forest without isolated vertices, each tree component has at least two vertices. Also, for a forest F and its tree components, T_1, \dots, T_k , we have $|H_2(F)| = \sum_{i=1}^k |H_2(T_i)|$, and $|S(F)| = \sum_{i=1}^k |S(T_i)|$, as well as $\phi(F) = \sum_{i=1}^k \phi(T_i)$. That is, introducing an extra

tree component does not change the $|H_2|$, $|S|$, and ϕ value of existing tree components. Hence, if the inequality holds for each of tree component, it also holds for the forest.

Theorem 6.6. *For every forest F without isolated vertices,*

$$\frac{|H_2| + |S|}{3} \leq \phi \leq \frac{|H_2| + |S|}{2} \quad (6.5)$$

Moreover, when taking Theorem 6.1 into account, we have the following corollary that is also a $3/2$ approximation of ϕ when $|S| \leq \frac{|H_2|}{2}$. This corollary is interesting as it indicates that when the number of support vertices is relatively small (when compared to the number of non-leaf vertices), the estimate $\max\{c(F), \frac{|H_2| + c(F)}{2}\}$ itself is already a $3/2$ approximation of ϕ . Thus there is no need to estimate $|S|$ at all.

Corollary 6.7. *In every forest F without isolated vertices, if $|S| \leq \frac{|H_2|}{2}$, then*

$$\max\{c(F), \frac{|H_2| + c(F)}{2}\} \leq \phi \leq \frac{|H_2| + |S|}{2} \leq \frac{3(|H_2| + c(F))}{4} \quad (6.6)$$

6.2 Tree Streaming Algorithms

6.2.1 Estimating the Matching Number and the Covering Number

The algorithms for estimating $|H_2|$ and $|S|$ have been presented and proved in Section 4.2. And the number of tree component, $c(F)$, can be estimated directly by keeping an edge counter to count the total number of edges, as $c(F) = m - n$. Moreover, since the number of total insertion is bounded by $\mathcal{O}(n)$, the space usage of the edge counter is at most $\mathcal{O}(\log n)$, and the algorithm success with probability 1.

The main streaming algorithm for estimating ϕ is similar to the streaming algorithms for domination number γ and independence number β . For simplicity, please refer to Section 4.2 and Section 5.3 for detailed streaming algorithms.

Theorem 6.8. *Given error rate and fail rate $\epsilon, \delta \in (0, 1)$, as well as a turnstile forest stream, there exists a randomized algorithm that uses two passes and outputs a $3/2(1 \pm \epsilon)$ approximation of the matching number with probability $1 - \delta$. The space usage of the algorithm is $\tilde{\mathcal{O}}(\sqrt{n})$.*

It is well-known that by the König-Egeváry theorem, the matching number (ϕ) is equal to the vertex cover number (τ) in bipartite graphs. Clearly every tree/forest is bipartite.

Theorem 6.9. *Given error rate and fail rate $\epsilon, \delta \in (0, 1)$, as well as a turnstile forest stream, there exist a randomized algorithm that uses two passes and outputs a $3/2(1 \pm \epsilon)$ approximation of the vertex cover number with probability $1 - \delta$. The space usage of the algorithm is $\tilde{O}(\sqrt{n})$.*

6.3 Hardness Results

For forest without isolated vertices, Esfandiari et al. [49] have proved the following lower bound for approximating the matching number in the streaming model.

Theorem 6.10. *For any forests without isolated vertices, any randomized algorithms that approximate the matching number to a factor better than $3/2$ require $\Omega(\sqrt{n})$ space.*

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this work, we study two classic combinatorial optimization problems, the dominating set problem and the independent set problem, under the data streams model. In general streamed graphs, computing a c -approximation for the minimum dominating set problem requires $\Omega(\frac{n^2}{c^2})$ bits [5], while c -approximating the maximum independent set requires $\Omega(\frac{n^2}{c^7})$ [41] bits. That is, we need to store almost the entire graph to achieve a good approximation for either problem. The problem does not become easier if we are only interested in the size of the optimal solution, i.e., the domination number and the independence number. However, when the graph is sparse (i.e., has average degree in $o(\log n)$), a good approximation on the domination number and the independence number can be obtained using space much less than storing the whole graph.

The independence number is lower-bound by $\sum_{v \in V(G)} \frac{1}{d(v)+1}$, which is also known as the Caro-Wei Bound [29, 118]. Halldórsson et al. [57] introduced the *online streaming* model and gave an *online streaming* algorithm that outputs an independent set of size at least the Caro-Wei Bound in expectation. Their algorithm uses $\mathcal{O}(n)$ working space and has update time $\mathcal{O}(\log n)$. Cormode et al. [40] gave a turnstile streaming algorithm that $(1 \pm \epsilon)$ -approximates the Caro-Wei Bound with probability at least $1 - \delta$ using $\mathcal{O}(\bar{d}\epsilon^{-2} \log n \log \delta^{-1})$ space. While algorithm by Cormode et al. [40] uses much less space, it cannot be modified to output an actual independent set. Using a different technique, we give an algorithm that combines the advantages of both algorithms. Firstly,

our algorithm can $(1 \pm \epsilon)$ -approximate the Caro-Wei Bound with the same probability and space usage as [40]. More importantly, it can be modified to report an actual independent set in the *online streaming* model with much cheaper working space usage, $\mathcal{O}(\log \epsilon^{-1} \log n \log \delta^{-1})$, and update time, $\mathcal{O}(\log 1/\epsilon)$. Also, the error rate of the modified algorithm is bounded by our analysis: the returned independent set has size within $(1 \pm \epsilon)$ -factor of the Caro-Wei Bound with probability at least $1 - \delta$. Our algorithm requires the graphs have maximum degree $\Delta \leq \frac{\epsilon^2 n}{3(\bar{d} + 1)^3}$, which is still $\mathcal{O}(n)$ if ϵ and \bar{d} are constants. However, if we only need to estimate the Caro-Wei Bound (i.e., no need to output an actual solution), this restriction can be removed with some extra space ($\mathcal{O}(\bar{d} \log n)$ factor) and an extra post-processing time. Lastly, if the stream is vertex-arrival and random-order, the space usage of our algorithm can be further reduced to just $\mathcal{O}(\log(\bar{d}\epsilon^{-2}) \log \delta^{-1})$.

For streamed trees or forests, previous studies showed that the domination number can be 3-approximated using the number of non-leaf vertices, and the independence number can be 3/2-approximated using the number of leaves [84]. The number of non-leaf vertices can be $(1 \pm \epsilon)$ -approximated in data streams using $\text{poly}(\log n)$ space [25]. However, no streaming algorithm has been shown to for estimating the number of leaves. We demonstrate a streaming algorithm that $(1 \pm \epsilon)$ -approximates the number of leaves using $\text{poly}(\log n)$ space. Also, we show that by considering the number of support vertices (i.e., vertices adjacent to leaves), the domination number can be 2-approximated and the independence number can be 4/3-approximated. Using this technique, we design streaming algorithms that $2(1 \pm \epsilon)$ -approximate the domination number and $4/3(1 \pm \epsilon)$ -approximate the independence number using two passes and $\tilde{\mathcal{O}}(\sqrt{n})$ space. We believe the idea of support vertices can be adapted to give improved bounds on other streamed graph problems. For example, we show that the number of support vertices can be used to improve the approximation ratio for the size of the maximum matching from 2 [25, 39, 49] to 3/2 in the data stream model. Moreover, the number of support vertices can be easily estimated in other big-data models, such as the distributed model. Hence, our algorithms can be adapted to give improved bounds in those models.

7.2 Future Work

Our work leaves several open problems for future research. Firstly, as shown in [40], the streaming space lower bound for $(1 \pm \epsilon)$ -approximating the Caro-Wei bound is $\Omega(\bar{d}\epsilon^{-2})$. But the space usage of our algorithm and the algorithm in [40] is $\mathcal{O}(\bar{d}\epsilon^{-2} \log n)$. Can we remove this $\log n$ factor in the space usage or tighten the lower bound?

In this work, we demonstrate three applications of the support vertices, i.e., domination number, independence number, and matching number. Can this idea be applied to other graph problems? Our algorithms used 2 passes and $\tilde{\mathcal{O}}(\sqrt{n})$ space, but the lower bound results from [49] do not rule out the possibility of having $\text{poly}(\log n)$ -space algorithms with the same approximation ratios. Can we reduce to number of passes to one? Can we reduce the space usage to $\text{poly}(\log n)$? If the streaming algorithm uses a pass to estimate certain quantities, a common trick to reduce the number passes is by running $\log n$ guesses concurrently. But this technique does not immediately apply in our case, as in the first pass, we must retain the actual neighbours of sampled vertices.

Finally, can we generalize the definition of support vertices to more general graphs? For example, planar graphs and graphs with bounded arboricity. One possible choice is letting vertices whose degree is greater or equal than at least one of its neighbours be support vertices, which recently has been successfully applied on matching number in graphs with bounded arboricity [75]. Another possible choice is setting a degree threshold, and letting vertices with degree lower than this threshold be low-degree vertices. Vertices adjacent to these low-degree vertices are support vertices. We believe that better bounds can be achieved on a range of graph problems by careful definition of support vertices.

To conclude, in this work, we design better approximated algorithms for several graph problems in streamed sparse graphs, especially graphs with bounded average degree and forests. We mainly focus on the independent set problem and the dominating set problem under the data stream model, but we believe that our techniques are generic, such that they can be easily applied to other graphs problems and other computational models.

Bibliography

- [1] Datasketches. URL <https://datasketches.apache.org/>.
- [2] Noga Alon. Transversal numbers of uniform hypergraphs. *Graphs and Combinatorics*, 6(1):1–4, 1990.
- [3] Filipe Araujo, Jorge Farinha, Patricio Domingues, Gheorghe Cosmin Silaghi, and Derrick Kondo. A maximum independent set approach for collusion detection in voting pools. *Journal of Parallel and Distributed Computing*, 71(10):1356–1366, 2011.
- [4] V. I. Arnautov. Estimation of the exterior stability number of a graph by means of the minimal degree of the vertices (russian). *Prikl. Mat. i Programirovanie*, 11:3–8, 1974.
- [5] Sepehr Assadi, Sanjeev Khanna, and Yang Li. Tight bounds for single-pass streaming complexity of the set cover problem. *SIAM Journal on Computing*, (0):STOC16–341, 2019.
- [6] Giorgio Ausiello, Nicolas Boria, Aristotelis Giannakos, Giorgio Lucarelli, and V Th Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14): 1901–1913, 2012.
- [7] Brenda S Baker. Approximation algorithms for np-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.
- [8] Ainesh Bakshi, Nadiia Chepurko, and David P Woodruff. Weighted maximum independent set of geometric objects in turnstile streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

- [9] Sandip Banerjee and Sujoy Bhore. Algorithm and hardness results on liar’s dominating set and k -tuple dominating set. In *International Workshop on Combinatorial Algorithms*, pages 48–60. Springer, 2019.
- [10] Nikhil Bansal and Seeun William Umboh. Tight approximation bounds for dominating set on graphs of bounded arboricity. *Information Processing Letters*, 122: 21–24, 2017.
- [11] Nikhil Bansal, Anupam Gupta, and Guru Guruganesh. On the lovász theta function for independent sets in sparse graphs. *SIAM Journal on Computing*, 47(3): 1039–1055, 2018.
- [12] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences of the United States of America*, 43(9):842, 1957.
- [13] Csaba Biró, Éva Czabarka, Peter Dankelmann, and László Székely. Remarks on the domination number of graphs. *Inst. Combin. Appl.*, 64:73–83, 2012.
- [14] M. M. Blank. An estimate of the external stability number of a graph without suspended vertices (in russian). *Prikl. Mat. i Programirovanie*, 10:3–11, 1973.
- [15] Joseph K Blitzstein and Jessica Hwang. *Introduction to probability*. Crc Press, 2019.
- [16] Thomas Böhme and Bojan Mohar. Domination, packing and excluded minors. *the electronic journal of combinatorics*, pages N9–N9, 2003.
- [17] Ravi B Boppana, Magnús M Halldórsson, and Dror Rawitz. Simple and local independent set approximation. In *International Colloquium on Structural Information and Communication Complexity*, pages 88–101. Springer, 2018.
- [18] Vladimir Braverman and Stephen R Chestnut. Universal sketches for the frequency negative moments and other decreasing streaming sums. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, page 591, 2015.
- [19] Vladimir Braverman, Zaoxing Liu, Tejasvam Singh, NV Vinodchandran, and Lin F Yang. New bounds for the clique-gap problem using graph decomposition theory. *Algorithmica*, 80(2):652–667, 2018.

- [20] Nicolas Bray and Eric W. Weisstein. Domination number. URL <https://mathworld.wolfram.com/DominationNumber.html>. Last visited on 11/14/2020.
- [21] Robert C. Brigham and Ronald D. Dutton. Bounds on the Domination Number of a Graph. *The Quarterly Journal of Mathematics*, 41(3):269–275, 09 1990. ISSN 0033-5606. doi: 10.1093/qmath/41.3.269. URL <https://doi.org/10.1093/qmath/41.3.269>.
- [22] Hervé Brönnimann and Michael T Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
- [23] Csilla Bujtás. Domination number of graphs with minimum degree five. *Discussions Mathematicae Graph Theory*, 1(ahead-of-print), 2020.
- [24] Csilla Bujtás and Sandi Klavžar. Improved upper bounds on the domination number of graphs with minimum degree at least five. *Graphs and Combinatorics*, 32(2):511–519, 2016.
- [25] Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms-ESA 2015*, pages 263–274. Springer, 2015.
- [26] Marc Bury, Elena Grigorescu, Andrew McGregor, Morteza Monemizadeh, Chris Schwiegelshohn, Sofya Vorotnikova, and Samson Zhou. Structural results on matching estimation with applications to streaming. *Algorithmica*, 81(1):367–392, 2019.
- [27] Sergio Cabello and Pablo Pérez-Lantero. Interval selection in the streaming model. *Theoretical Computer Science*, 702:77–96, 2017.
- [28] Mihaela Cardei, Maggie Xiaoyan Cheng, Xiuzhen Cheng, and Ding-Zhu Du. Connected domination in multihop ad hoc wireless networks. In *JCIS*, pages 251–255. Citeseer, 2002.
- [29] Yair Caro. New results on the independence number. Technical report, Technical Report, Tel-Aviv University, 1979.
- [30] Yair Caro and Zsolt Tuza. Improved lower bounds on k-independence. *Journal of Graph Theory*, 15(1):99–107, 1991.

- [31] Rajesh Chitnis and Graham Cormode. Towards a theory of parameterized streaming algorithms. In *14th International Symposium on Parameterized and Exact Computation (IPEC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- [32] Rajesh Chitnis, Graham Cormode, Mohammad Taghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1234–1251. SIAM, 2014.
- [33] Rajesh Chitnis, Graham Cormode, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Andrew McGregor, Morteza Monemizadeh, and Sofya Vorotnikova. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 1326–1344. SIAM, 2016.
- [34] Brent N Clark, Charles J Colbourn, and David S Johnson. Unit disk graphs. *Discrete mathematics*, 86(1-3):165–177, 1990.
- [35] W Edwin Clark, Boris Shekhtman, Stephen Suen, and David C Fisher. Upper bounds for the domination number of a graph. *Congr. Numer*, 132:99–123, 1998.
- [36] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4(2):41 – 44, 1975. ISSN 0020-0190. doi: [https://doi.org/10.1016/0020-0190\(75\)90011-3](https://doi.org/10.1016/0020-0190(75)90011-3). URL <http://www.sciencedirect.com/science/article/pii/0020019075900113>.
- [37] Graham Cormode and Donatella Firmani. A unifying framework for ℓ_0 -sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.
- [38] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- [39] Graham Cormode, Hossein Jowhari, Morteza Monemizadeh, and S Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In *25th European Symposium on Algorithms, ESA 2017*, page 29. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2017.

- [40] Graham Cormode, Jacques Dark, and Christian Konrad. Approximating the caro-wei bound for independent sets in graph streams. In *International Symposium on Combinatorial Optimization*, pages 101–114. Springer, 2018.
- [41] Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
- [42] Ermelinda DeLaVina, Craig E Larson, Ryan Pepper, Bill Waller, and Odile Favaron. On total domination and support vertices of a tree. *AKCE International Journal of Graphs and Combinatorics*, 7(1):85–95, 2010.
- [43] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 624–633, 2014.
- [44] Pierre Duchet and Henri Meyniel. On hadwiger’s number and the stability number. In *North-Holland Mathematics Studies*, volume 62, pages 71–73. Elsevier, 1982.
- [45] Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013.
- [46] Zdeněk Dvořák. On distance r -dominating and $2r$ -independent sets in sparse graphs. *Journal of Graph Theory*, 91(2):162–173, 2019.
- [47] Stephan J Eidenbenz. Online dominating set and variations on restricted graph classes. *Technical Report/ETH Zurich, Department of Computer Science*, 380, 2002.
- [48] Yuval Emek, Magnús M Halldórsson, and Adi Rosén. Space-constrained interval selection. In *International Colloquium on Automata, Languages, and Programming*, pages 302–313. Springer, 2012.
- [49] Hossein Esfandiari, Mohammadtaghi Hajiaghayi, Vahid Liaghat, Morteza Monezadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. *ACM Transactions on Algorithms (TALG)*, 14(4):1–23, 2018.

- [50] Guy Even, Dror Rawitz, and Shimon Moni Shahar. Hitting sets when the vc-dimension is small. *Information Processing Letters*, 95(2):358–362, 2005.
- [51] Stefan Fafianie and Stefan Kratsch. Streaming kernelization. In *International Symposium on Mathematical Foundations of Computer Science*, pages 275–286. Springer, 2014.
- [52] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Departmental Papers (CIS)*, page 236, 2005.
- [53] András Frank. Some polynomial algorithms for certain graphs and hypergraphs. In *Proceedings of the Fifth British Combinatorial Conference*, pages 211–226. Congressus Numerantium XV, Eds, 1975.
- [54] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.
- [55] Michael R Garey and David S Johnson. A guide to the theory of np-completeness. *Computers and Intractability*, pages 37–79, 1990.
- [56] Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Evaluation of labeling strategies for rotating maps. *Journal of Experimental Algorithmics (JEA)*, 21: 1–21, 2016.
- [57] Bjarni V Halldórsson, Magnús M Halldórsson, Elena Losievskaja, and Mario Szegedy. Streaming algorithms for independent sets in sparse hypergraphs. *Algorithmica*, 76(2):490–501, 2016.
- [58] Magnús M Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.
- [59] Magnús M Halldórsson, Xiaoming Sun, Mario Szegedy, and Chengu Wang. Streaming and communication complexity of clique approximation. In *International Colloquium on Automata, Languages, and Programming*, pages 449–460. Springer, 2012.

- [60] Sarel Har-Peled and Kent Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. *SIAM Journal on Computing*, 46(6):1712–1744, 2017.
- [61] Avinatan Hassidim, Jonathan A Kelner, Huy N Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 22–31. IEEE, 2009.
- [62] Johan Håstad. Clique is hard to approximate within $1 - \epsilon$. *Acta Mathematica*, 182(1):105–142, 1999.
- [63] Teresa W Haynes, Stephen Hedetniemi, and Peter Slater. *Fundamentals of domination in graphs*. CRC press, 1998.
- [64] ST Hedetniemi and R Laskar. Connected domination in graphs. *Bollobás B (ed) Graph theory and combinatorics*, pages 209–217, 1984.
- [65] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.
- [66] Dorit S Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.
- [67] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The Collected Works of Wassily Hoeffding*, pages 409–426. Springer, 1994.
- [68] Ayaan Hossain, Eriberto Lopez, Sean M Halper, Daniel P Cetnar, Alexander C Reis, Devin Strickland, Eric Klavins, and Howard M Salis. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems. *Nature biotechnology*, 38(12):1466–1475, 2020.
- [69] Harry B Hunt III, Madhav V Marathe, Venkatesh Radhakrishnan, Shankar S Ravi, Daniel J Rosenkrantz, and Richard E Stearns. N_c -approximation schemes for np- and pspace-hard problems for geometric graphs. *Journal of algorithms*, 26(2):238–274, 1998.
- [70] Wilfried Imrich, Sandi Klavzar, and Douglas F Rall. *Topics in graph theory: Graphs and their Cartesian product*. CRC Press, 2008.

- [71] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.
- [72] Cao Jianxiang, Shi Minyong, Moo-Young Sohn, and Yuan Xudong. Domination in graphs with minimum degree six. *Journal of applied mathematics & informatics*, 26(5_6):1085–1100, 2008.
- [73] David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.
- [74] Mark Jones, Daniel Lokshtanov, MS Ramanujan, Saket Saurabh, and Ondřej Suchý. Parameterized complexity of directed steiner tree on sparse graphs. In *European Symposium on Algorithms*, pages 671–682. Springer, 2013.
- [75] Hossein Jowhari. An estimator for matching size in low arboricity graphs with two applications. *arXiv preprint arXiv:2011.11706*, 2020.
- [76] Daniel M Kane, Jelani Nelson, and David P Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1161–1178. SIAM, 2010.
- [77] Daniel M Kane, Jelani Nelson, and David P Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 41–52, 2010.
- [78] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology Stockholm, 1992.
- [79] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- [80] Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed time-dependent contraction hierarchies. In *International Symposium on Experimental Algorithms*, pages 83–93. Springer, 2010.
- [81] Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda. A linear algorithm for the domination number of a series-parallel graph. *Discrete Applied Mathematics*, 5(3):299–311, 1983.

-
- [82] Daniel J Kleitman and Douglas B West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics*, 4(1):99–106, 1991.
- [83] Alexandr V Kostochka and BY Stodolsky. An upper bound on the domination number of n -vertex connected cubic graphs. *Discrete Mathematics*, 309(5):1142–1162, 2009.
- [84] Rajesh Chitnis. Robert Krauthgamer. Streaming algorithms for problems on special graph classes. 2019.
- [85] B Küpper and Lutz Volkmann. Upper bounds on the domination number of a graph in terms of order, diameter and minimum degree. *Australasian Journal of Combinatorics*, 35:133, 2006.
- [86] Magdalena Lemańska. Lower bound on the domination number of a tree. *Discussiones Mathematicae Graph Theory*, 24(2):165–169, 2004.
- [87] Christoph Lenzen and Roger Wattenhofer. Minimum dominating set approximation in graphs of bounded arboricity. In *International symposium on distributed computing*, pages 510–524. Springer, 2010.
- [88] Jonathan S Li, Rohan Potru, and Farhad Shahrokhi. A performance study of some approximation algorithms for minimum dominating set in a graph. *arXiv preprint arXiv:2009.04636*, 2020.
- [89] Hailong Liu and Liang Sun. On domination number of 4-regular graphs. *Czechoslovak Mathematical Journal*, 54(4):889–898, 2004.
- [90] Daniel Lokshantov, Martin Vatshelle, and Yngve Villanger. Independent set in p -5-free graphs in polynomial time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on discrete algorithms*, pages 570–581. SIAM, 2014.
- [91] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.
- [92] William McCuaig and Bruce Shepherd. Domination in graphs with minimum degree two. *Journal of Graph Theory*, 13(6):749–762, 1989.
- [93] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

-
- [94] Andrew McGregor and Hoa T Vu. Better streaming algorithms for the maximum coverage problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019.
- [95] Dirk Meierling and Lutz Volkmann. A lower bound for the distance k -domination number of trees. *Results in Mathematics*, 47(3-4):335–339, 2005.
- [96] Tijana Milenković, Vesna Memišević, Anthony Bonato, and Nataša Pržulj. Dominating biological networks. *PloS one*, 6(8):e23016, 2011.
- [97] Morteza Monemizadeh, Shan Muthukrishnan, Pan Peng, and Christian Sohler. Testable bounded degree graph properties are random order streamable. *arXiv preprint arXiv:1707.07334*, 2017.
- [98] Jose C Nacher and Tatsuya Akutsu. Dominating scale-free networks with variable scaling exponent: heterogeneous networks are not difficult to control. *New Journal of Physics*, 14(7):073005, 2012.
- [99] George L Nemhauser and Leslie Earl Trotter. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.
- [100] Huy N Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE, 2008.
- [101] Paolo Nobili and Antonio Sassano. An $\mathcal{O}(n^2 \log n)$ algorithm for the weighted stable set problem in claw-free graphs. *Mathematical Programming*, pages 1–29, 2020.
- [102] O. Ore. *Theory of Graphs*. American Mathematical Society colloquium publications. American Mathematical Society, 1962. ISBN 9780821810385.
- [103] Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.
- [104] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sub-linear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

-
- [105] C. Payan. Sur le nombre d'absorption d'un graphe simple. *Cahiers Centre Etudes Recherche Op' er.*, 17:307–317, 1975.
- [106] Pan Peng and Christian Sohler. Estimating graph parameters from random order streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2449–2466. SIAM, 2018.
- [107] Tayler Pino, Salimur Choudhury, and Fadi Al-Turjman. Dominating set algorithms for wireless sensor networks survivability. *IEEE Access*, 6:17527–17532, 2018.
- [108] Bruce Reed. Paths, stars and the number three. *Combinatorics, Probability & Computing*, 5:277–295, 1996.
- [109] Ryan A Rossi and Nesreen K Ahmed. Networkrepository: A graph data repository with visual interactive analytics. In *29th AAAI Conference on Artificial Intelligence, Austin, Texas, USA*, pages 25–30, 2015.
- [110] E Sampathkumar and HB Walikar. The connected domination number of a graph. *J. Math. Phys*, 1979.
- [111] Chao Shen and Tao Li. Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 984–992, 2010.
- [112] Sebastian Siebertz. Greedy domination on biclique-free graphs. *Information Processing Letters*, 145:64–67, 2019.
- [113] Moo-Young Sohn and Yuan Xudong. Domination in graphs of minimum degree four. *Journal of the Korean Mathematical Society*, 46(4):759–773, 2009.
- [114] Kazuhiko Takamizawa, Takao Nishizeki, and Nobuji Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM (JACM)*, 29(3):623–641, 1982.
- [115] Paul Turán. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48(436-452):137, 1941.
- [116] Elad Verbin and Wei Yu. The streaming complexity of cycle counting, sorting by reversals, and other problems. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 11–25. SIAM, 2011.

-
- [117] Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.
- [118] VK Wei. A lower bound on the stability number of a simple graph. Technical report, Bell Laboratories Technical Memorandum, 1981.
- [119] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.
- [120] Jie Wu, Mihaela Cardei, Fei Dai, and Shuhui Yang. Extended dominating set and its applications in ad hoc networks using cooperative communication. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):851–864, 2006.
- [121] Mingyu Xiao and Hiroshi Nagamochi. Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs. *Theoretical Computer Science*, 469:92–104, 2013.
- [122] Mingyu Xiao and Hiroshi Nagamochi. Exact algorithms for maximum independent set. *Information and Computation*, 255:126–146, 2017.
- [123] Hua-Ming Xing, Liang Sun, and Xue-Gang Chen. Domination in graphs of minimum degree five. *Graphs and Combinatorics*, 22(1):127–143, 2006.
- [124] Jiguo Yu, Nannan Wang, Guanghui Wang, and Dongxiao Yu. Connected dominating sets in wireless ad hoc and sensor networks—a comprehensive survey. *Computer Communications*, 36(2):121–134, 2013.
- [125] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 681–690, 2006.