Approximation in (Poly-) Logarithmic Space

Arindam Biswas
The Institute of Mathematical Sciences, HBNI, Chennai, India
barindam@imsc.res.in

Venkatesh Raman
The Institute of Mathematical Sciences, HBNI, Chennai, India
vraman@imsc.res.in

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, Chennai, India
University of Bergen, Bergen, Norway
saket@imsc.res.in

Abstract

We develop new approximation algorithms for classical graph and set problems in the RAM model under space constraints. As one of our main results, we devise an algorithm for d–Hitting Set that runs in time $n^{\mathrm{O}\left(d^2+(d/\epsilon)\right)}$, uses $\mathrm{O}\big((d^2+(d/\epsilon))\log n\big)$ bits of space, and achieves an approximation ratio of $\mathrm{O}((d/\epsilon)n^{\epsilon})$ for any positive $\epsilon \leq 1$ and any constant $d \in \mathbb{N}$. In particular, this yields a factor-$\mathrm{O}(d \log n)$ approximation algorithm which uses $\mathrm{O}\big(\log^2 n\big)$ bits of space. As a corollary, we obtain similar bounds on space and approximation ratio for Vertex Cover and several graph deletion problems. For graphs with maximum degree Δ, one can do better. We give a factor-2 approximation algorithm for Vertex Cover which runs in time $n^{\mathrm{O}(\Delta)}$ and uses $\mathrm{O}(\Delta \log n)$ bits of space.

For Independent Set on graphs with average degree d, we give a factor-$(2d)$ approximation algorithm which runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space. We also devise a factor-$\mathrm{O}\big(d^2\big)$ approximation algorithm for Dominating Set on d-degenerate graphs which runs in time $n^{\mathrm{O}(\log n)}$ and uses $\mathrm{O}\big(\log^2 n\big)$ bits of space. For d-regular graphs, we observe that a known randomized algorithm which achieves an approximation ratio of $\mathrm{O}(\log d)$ can be derandomized to run in polynomial time and use $\mathrm{O}(\log n)$ bits of space.

Our results use a combination of ideas from the theory of kernelization, distributed algorithms and randomized algorithms.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases approximation, logspace, logarithmic, log, space, small, limited, memory, ROM, read-only

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.16

Related Version A full version of this paper is available at `https://arxiv.org/abs/2008.04416`.

1 Introduction and Motivation

This paper examines the classical approximation problems Vertex Cover, Hitting Set and Dominating Set in the RAM model under additional polylogarithmic space constraints. We devise approximation algorithms for these problems which use polylogarithmic space in general and $\mathrm{O}(\log n)$ bits of space on certain special input types.

In the absence of space constraints, the greedy heuristic is a good starting point for many approximation algorithms. For Set Cover, it even yields optimal (under certain complexity-theoretic assumptions) approximation ratios [2, 17]. However, the heuristic inherently changes the input in some way. In a space-constrained setting however, this is

45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020).
Editors: Javier Esparza and Daniel Král'; Article No. 16; pp. 16:1–16:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

asking for too much: the input is immutable, and the amount of auxiliary space available (polylogarithmic in our case) is not sufficient to register changes to the input.

Linear programming is another tool that plays a central role in the design of approximation algorithms. While it yields competitive approximations in polynomial time when space is not constrained, it is known that under logarithmic-space reductions, it is P-complete to approximate the LINEAR PROGRAMMING problem to any constant factor [39]. Such a result can be shown even for positive linear programming [42].

Machine Model. We use the standard RAM model with an additional polylogarithmic space constraint. For inputs n bits in length, memory is organized as words of length $\mathrm{O}(\log n)$, which allows the entire input to be addressed using a single word of memory. Integer arithmetic operations on pairs of words and single-word memory access operations take constant time. The input (a graph or family of sets) is provided to the algorithm using some canonical encoding, which can be read but not modified, i.e. the algorithm has read-only access to the input.

The algorithm uses some auxiliary memory, to which it has read-write access, and in the setting of this paper, the amount of such memory available is bounded by a polynomial in $\log n$. Output is written to a stream: once something is output, the algorithm cannot read it back at a later point as it executes. We count the amount of auxiliary memory used in units of 1 bit, and the objective is to use as little auxiliary memory as possible.

Our Results

***d*–Hitting Set and Vertex Deletion Problems.** An instance of the d–HITTING SET problem consists of a universe and a family of size-d subsets of the universe, and the objective is to find a subset of the universe that has a non-empty intersection with each set in the family.

- We develop a factor-$\mathrm{O}((d/\epsilon)n^{\epsilon})$ approximation algorithm for d–HITTING SET which runs in time $n^{Ohd^2+(d/\epsilon)}$ and uses $\mathrm{O}\big((d^2 + (d/\epsilon)) \log n\big)$ bits of space (Section 3), where $\epsilon \leq 1$ is an arbitrary positive number and d is a fixed positive integer. In particular, this yields a factor-$\mathrm{O}(d \log n)$ approximation algorithm for the problem which uses $\mathrm{O}\big(\log^2 n\big)$ bits of space. As an application, we show how the algorithm can be used to approximate various *deletion* problems with similar space bounds. From this, we derive a factor-$\mathrm{O}((1/\epsilon)n^{\epsilon})$ (for arbitrary positive $\epsilon \leq 1$) approximation algorithm for VERTEX COVER that runs in time $n^{\mathrm{O}(1/\epsilon))}$ and uses $\mathrm{O}((1/\epsilon) \log n)$ bits of space.
- We give a simple factor-2 approximation algorithm for VERTEX COVER on graphs with maximum degree Δ which runs in time $n^{\mathrm{O}(\Delta)}$ and uses $\mathrm{O}(\Delta \log n)$ bits of space (Section 3.1).

Dominating Set. In the DOMINATING SET problem, the objective is to find a vertex set of minimum size in a graph such that all other vertices are adjacent to some vertex in the set.

- We give a factor-$\mathrm{O}(\sqrt{n})$ approximation algorithm for graphs excluding C_4 (a cycle on 4 vertices) as a subgraph, which runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space (Section 4.1).
- Graphs of bounded degeneracy form a large class which includes planar graphs, graphs of bounded genus, graphs excluding a fixed graph H as a (topological) minor and graphs of bounded expansion. For graphs with degeneracy d, we give a factor-$\mathrm{O}\big(d^2\big)$ approximation algorithm which uses $\mathrm{O}\big(\log^2 n\big)$ bits of space. (Section 4.2).

- Additionally, for graphs in which each vertex has degree d, i.e. d-regular graphs, we exhibit a factor-$O(\log d)$ approximation algorithm for Dominating Set (Section 4.3) which is an adaptation of known results to the constrained-space setting.

Independent Set. An instance of the Independent Set problem consists of a graph, and the objective is to find an *independent set* of maximum size i.e. a set of vertices with no edges between them. We show how a known factor-$(2d)$ approximation algorithm for Independent Set on graphs with average degree d can be implemented to run in polynomial time and use $O(\log n)$ bits of space (Section 5).

Related Work

Small-space models such as the streaming model and the in-place model have been the subject of much research over the last two decades (see [27, 15, 13] and references therein). In the streaming model, in addition to the space constraint, the algorithm is also required to read the input in a specific (possibly adversarial) sequence in one or more passes. The in-place model, on the other hand, allows the memory used for storing the input to be modified. The read-only RAM model we use is distinct from both these models. Historically, the read-only model has been studied from the perspective of time–space tradeoff lower bounds, particularly for problems like Sorting [8, 9, 5, 31, 30] and Selection [28, 20, 29, 34].

The earliest graph problems studied in this model were the undirected and directed graph reachability problems (resp. USTCON and STCON) in connection with the complexity classes L and NL. Savitch [38] showed that on input graphs with n vetices, STCON (and therefore also USTCON) can be solved in $O(\log^2 n)$ bits of space. This bound was gradually whittled down over more than two decades, a process culminating in the result of Reingold [37] which shows that USTCON can be solved using $O(\log n)$ bits of space.

Reif [36] showed that the problems of recognizing bipartite, chordal, interval and split graphs are reducible to USTCON. Later on, Allender and Mahajan [1] showed that planarity testing also reduces to USTCON. Thus, Reingold's result put all these problems in L. More recently, Elberfeld and Kawarabayashi [18] showed that the problems of recognizing and canonizing bounded-genus graphs were in L. The model was also studied by Yamakami [44] in relation to the complexity of search problems solvable in polynomial time, and by Tantau [40], who studied the approximation properties of search problems that can be solved in nondeterministic logarithmic space.

The other direction in which small-space problems and even the approximation problems we study have been investigated previously is in the context of fast parallel algorithms. By a known reduction, algorithms for these problems have sequential implementations that use polylogarithmic space. The PRAM algorithm of Luby [25] for finding maximal independent sets in a graph can be used to 2-approximate Vertex Cover (recall that a better than 2-approximate algorithm is known to be unlikely [23]). Implemented in the sequential RAM model, it uses $O(\log^2 n)$ bits of space. There have been attempts to generalize Luby's algorithm to hypergraphs, and to the best of our knowledge, an efficient deterministic parallel algorithm (an NC algorithm) to find maximal independent sets in hypergraphs is not known to exist (see [6] and references therein). Our scheme for d–Hitting Set trades approximation factor against space used to obtain a family of algorithms that use $O\big((d^2 + (d/\epsilon)) \log n\big)$ bits of space to obtain $O((d/\epsilon)n^{\epsilon})$-approximate solutions for any positive $\epsilon \leq 1$. As a corollary, we obtain an $O(d \log n)$-approximation algorithm that uses $O(\log^2 n)$ bits of space. On graphs with maximum degree Δ, our approximation algorithm for Vertex Cover uses $O(\Delta \log n)$ bits of space to obtain 2-approximate solutions.

Berger et al. [7] gave a PRAM algorithm for Set Cover which can be implemented in the sequential RAM model to $\mathrm{O}(\log n)$-approximate Dominating Set in $\mathrm{O}\big(\log^4 n\big)$ bits of space. See also [41, 26], which give parallel approximation algorithms for Linear Programming, and see [24], which gives tight approximation ratios for CSP's using semi-definite programming in the PRAM model. Our algorithms for Dominating Set are simpler and more direct, and work for a large class of graphs while using $\mathrm{O}\big(\log^2 n\big)$ bits of space.

Our Techniques

As noted earlier, the greedy heuristic causes changes to the input, which our model does not permit. To get around this, we use a *staggered* greedy approach in which the solution is constructed in a sequence of greedy steps to approximate Vertex Cover on graphs of bounded degree (Section 3.1). By combining this with data reduction rules from kernelization algorithms, we also obtain approximations for Vertex Cover and more generally d–Hitting Set (Section 3), and restricted versions of Dominating Set (Sections 4.1 and 4.2). In Sections 4 and 5, we use 2-universal hash families constructible in logarithmic space to approximate Independent Set on graphs of bounded average degree (Section 5) and Dominating Set on regular graphs (Section 4.3) in logarithmic space.

Full Version. Details for all items marked † can be found in the full version of this paper at `https://arxiv.org/abs/2008.04416`.

2 Preliminaries

Notation. $\mathbb{N}$ denotes the set of natural numbers $\{0, 1, \ldots\}$ and $\mathbb{Z}^+$ denotes the set of positive integers $\{1, 2, \ldots\}$. For $n \in \mathbb{Z}^+$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Let G be a graph. Its vertex set is denoted by $\mathrm{V}(G)$, and its edge set by $\mathrm{E}(G)$. The degree of a vertex v is denoted by $\deg(v)$, and for a set $S \subseteq \mathrm{V}(G)$ or a subraph H of G, $\deg_S(v)$ denotes the degree of v in $G[S]$ and $\deg_H(v)$ denotes the degree of v in H.

Known Results. We begin by considering the following result, which arises from a logarithmic-space implementation of the Buss kernelization rule [10] for Vertex Cover combined with the observation that the kernel produced is itself a vertex cover.

▶ **Proposition 1** (Cai et al. [11], Theorem 2.3)**.** *There is an algorithm which takes as input a graph G and $k \in \mathbb{N}$, and either determines that G has no vertex cover of size at most k or produces a vertex cover of size at most $2k^2$. The algorithm runs in time $\mathrm{O}\big(n^2\big)$ and uses $\mathrm{O}(\log n)$ bits of space.*

The Vertex Cover problem is a special case of d–Hitting Set ($d \in \mathbb{N}$, a constant), an instance of which comprises a family $\mathcal{F}$ of size-d subsets of a ground set. The objective is to compute a minimum *hitting set* for $\mathcal{F}$, i.e. a subset of the ground set which intersects each set in $\mathcal{F}$. The next proposition shows that a result similar to the one above also holds for this generalization.

▶ **Proposition 2** (Fafianie and Kratsch [19], Theorem 1)**.** *There is an algorithm which takes as input a family $\mathcal{F}$ of d-subsets ($d \in \mathbb{N}$, a constant) of a ground set U and $k \in \mathbb{N}$, and either determines that $\mathcal{F}$ has no hitting set of size at most k or produces an equivalent subfamily of the original family which has size $\mathrm{O}\Big((k+1)^d\Big)$. The algorithm runs in time $n^{\mathrm{O}\left(d^2\right)}$ and uses $\mathrm{O}\big(d^2 \log n\big)$ bits of space.*

2.1 Presenting modified graphs using oracles

Our algorithms repeatedly “delete” vertices or sets of vertices, but as they only have read-only access to the graph (or family of sets), we require a way to implement these deletions using a small amount of auxiliary space. Towards that, we prove the following theorem.

▶ **Theorem 3.** *Let $G = G_0 = (V, E)$ be a graph with n vertices, and let G_i ($i \in [k]$) be obtained from G_{i-1} by deleting a set $S_i \subseteq \mathrm{V}(G_{i-1})$ consisting of all vertices $v \in \mathrm{V}(G_{i-1})$ which satisfy a property that can be checked (given access to G_{i-1}) using $\mathrm{O}(\log n)$ bits of space.*

Given read-only access to G, one can, for each $i \in [k]$, enumerate and answer membership queries for S_i, $V_i = \mathrm{V}(G_i)$ and $E_i = \mathrm{E}(G_i)$ in time $n^{\mathrm{O}(i)}$ using $\mathrm{O}(i \log n)$ bits of space.

Proof. For each $i \in [k]$ let $\texttt{Check}_i(G_{i-1}, v)$ be the algorithmic check which, given (oracle) access to G_{i-1}, determines whether $v \in V_{i-1}$ satisfies the condition for inclusion in S_i. Note that this condition may be something that depends on the graph G_{i-1}, i.e. G_{i-1} must be accessible to $\texttt{Check}_i$.

To provide oracle access to G_i, V_i and E_i, it suffices to compute, for $v \in V$ and $uw \in E$, the predicates $[v \in V_i]$ and $[uw \in E_i]$. A vertex is in V_i if and only if it is in V_{i-1} and it is not in S_i. Similarly, an edge is in E_i if and only if it is in E_{i-1} and neither of its endpoints are in S_i. Thus, we have the following relations.

$$[v \in V_i] \equiv [v \in V_{i-1}] \wedge \neg \texttt{Check}_i(G_{i-1}, v) \tag{1}$$

$$[uw \in E_i] \equiv [uw \in E_{i-1}] \wedge \neg(\texttt{Check}_i(G_{i-1}, u) \vee \texttt{Check}_i(G_{i-1}, w)) \tag{2}$$

To compute each of these predicates for G_i, we require oracle access to G_{i-1}, which in turn involves computing the predicates $[v \in V_{i-1}]$ and $[uw \in E_{i-1}]$. Suppose the number of operations needed to compute $\texttt{Check}_i(G_{i-1}, v)$ is $r(n)$, where r is a polynomial (it uses $\mathrm{O}(\log n)$ bits of space, so it is polynomial-time). Let p_i (resp. q_i) be the amount of space used to compute the predicate $[v \in V_i]$ (resp. $[uw \in E_i]$), and let s_i (resp. t_i) be the time needed to compute the predicate $[v \in V_i]$ (resp. $[uw \in E_i]$). From Relations 1 and 2 and the fact that $\texttt{Check}_i$ accesses G_{i-1} at most $r(n)$ times, we see that these quantities satisfy the following relations.

$$p_i = p_{i-1} + \mathrm{O}(\log n), \qquad q_i = q_{i-1} + \mathrm{O}(\log n) \tag{3}$$

$$s_i = s_{i-1} + \mathrm{O}(r(n)(s_{i-1} + t_{i-1})), \qquad t_i = t_{i-1} + \mathrm{O}(r(n)(s_{i-1} + t_{i-1})) \tag{4}$$

It is easy to see that these recurrences solve to $p_i, q_i = \mathrm{O}(i \log n)$ and $s_i, t_i = n^{\mathrm{O}(i)}$, so both predicates can be computed in time $n^{\mathrm{O}(i)}$ using $\mathrm{O}(i \log n)$ bits of space.

With oracle access to G_{i-1}, the predicate $[v \in S_i]$ can be computed simply as $\texttt{Check}_i(G_{i-1}, v)$, from which enumerating V_i (resp. E_i and S_i) is straightforward: enumerate V (resp. E and V) and suppress vertices v (resp. edges uw and vertices z) which fail the predicate $[v \in V_i]$ (resp. $[uw \in E_i]$ and $[z \in S_i]$). As the most space-hungry operations are the membership queries, the enumeration can also be performed using $\mathrm{O}(i \log n)$ bits of space. The enumeration needs time $n^{\mathrm{O}(i)}$ for each element of V and E, and since $|V|, |E| = \mathrm{O}(n^2)$, the total time needed is also $n^{\mathrm{O}(i)}$. ◀

2.2 Universal Hash Families

Some of our algorithms use the trick of randomized sampling to obtain a certain structure with good probability and then derandomize this procedure by using a 2-*universal* family of hash functions. A 2-*universal hash family* is a family $\mathcal{F}$ of functions from $[n]$ to $[k]$ ($n, k \in \mathbb{N}$ and $k \leq n$) such that for any pair i and j of elements in $[n]$, the number of functions from $\mathcal{F}$ that map i and j to the same element in $[k]$ is at most $|\mathcal{F}|/k$.

The following proposition is a combination of a result of Carter and Wegman [12] showing the existence of such families, and the observation that these families can be computed in logarithmic space [43]. Later on, we use it to derandomize sampling procedures in some of our algorithms.

▶ **Proposition 4** (Carter and Wegman [12], Proposition 7). *Let $n, k \in \mathbb{N}$ with $n \geq k$. One can compute a 2-universal hash family for $[[n] \to [k]]$ in polynomial time using $\mathrm{O}(\log n)$ bits of space.*

3 Hitting Sets and Π-Deletion Problems

The d–Hitting Set problem is a generalization of Vertex Cover in which an instance consists of a family $\mathcal{F}$ of d-subsets of a ground set U, and the objective is to find a subset of U of minimum size which intersects all sets in $\mathcal{F}$.

Algorithms for the problem are useful as subroutines in solving various *deletion* problems, where the objective is to delete the minimum possible number of vertices from a graph so that the resulting graph satisfies a certain property. The following result is a corollary to Proposition 2.

▶ **Corollary 5.** *Let $\mathcal{F}$ be a family of d-subsets of a ground set U with n elements. One can compute an $\mathrm{O}\big(dn^{1-1/d}\big)$-approximate minimum hitting set for $\mathcal{F}$ in time $n^{\mathrm{O}\left(d^2\right)}$ using $\mathrm{O}\big(d^2 \log n\big)$ bits of space.*

Proof. Consider the following algorithm. Starting at $k = 1$, run the algorithm of Proposition 2 and repeatedly increment the value of k until $k = n^{1/d}$ or the algorithm returns a solution of size $\mathrm{O}\Big(d(k+1)^d\Big)$ (i.e. it does not return a `NO` answer) for the first time. If k is incremented until $n^{1/d}$, then simply return the entire universe as the solution. Clearly, the approximation ratio is $n^{1-1/d}$, as $OPT \geq n^{1/d}$ (and so the size of the solution returned is $n = n^{1-1/d} \cdot n^{1/d} \leq n^{1-1/d} \cdot OPT$, where OPT is the size of the minimum hitting set).

If $k < n^{1/d}$, then the size of the solution produced is $\mathrm{O}\Big(d(k+1)^d\Big)$, and we know that $OPT \geq k$, since the algorithm had returned `NO` answers until this point. So the size of the solution produced is $\mathrm{O}\Big(d(k+1)^d\Big) = \mathrm{O}\big(d(k+1)^{d-1} \cdot (OPT+1)\big) = \mathrm{O}\big(dn^{1-1/d} \cdot (OPT+1)\big)$. Thus, we have an $\mathrm{O}\big(dn^{1-1/d}\big)$-approximation. The bounds on running time and space used follow from the fact that the algorithm of Proposition 2 runs in time $n^{\mathrm{O}\left(d^2\right)}$ and uses $\mathrm{O}\big(d^2 \log n\big)$ bits of space. ◀

The next result is one of our main results en route to developing a space-efficient approximation algorithm for d–Hitting Set.

▶ **Lemma 6.** *Let $\epsilon \leq 1$ be a positive number. There is an algorithm which takes as input a family $\mathcal{F}$ of d-subsets of a ground set U of n elements and $k \in \mathbb{N}$, and either determines $\mathcal{F}$ has no hitting set of size at most k or produces a hitting set of size $\mathrm{O}\big((d/\epsilon)k^{1+\epsilon}\big)$. The algorithm runs in time $n^{\mathrm{O}\left(d^2+(d/\epsilon)\right)}$ and uses $\mathrm{O}\big((d^2+(d/\epsilon))\log n\big)$ bits of space.*

Proof. Let $i = \lceil (d-1)/\epsilon \rceil$. The algorithm performs i rounds of computation, each using $\mathrm{O}(\log n)$ bits of space to determine a set of elements (accessible by oracle) to be removed in the next round, or determine that $\mathcal{F}$ has no hitting set of size at most k.

1. Use the algorithm of Proposition 2 to obtain a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ over the ground set $U' \subseteq U$ such that
   - $|\mathcal{F}'| \leq c(k+1)^d$, $|U'| = cd(k+1)^d$, and
   - there exists a hitting set $S \subseteq U$ of size at most k in $\mathcal{F}$ if and only if there exists a hitting set $S' \subseteq U'$ and S' is a hitting set for $\mathcal{F}'$.
2. Set $U_0 = U'$ and $\mathcal{F}_0 = \mathcal{F}'$. For $j = \{1, 2, \ldots, i-1\}$, perform the following steps.
   - Determine S_j, the set of all elements in U_{j-1} which appear in at least $c(k+1)^{d-1-j\epsilon}$ sets in $\mathcal{F}_{j-1}$.
   - Let $U_j = U_{j-1} \setminus S_j$ and $\mathcal{F}_j = \{A \in \mathcal{F}_{j-1} \mid A \cap S_j = \emptyset\}$. If there are more than $c(k+1)^{d-j\epsilon}$ sets in $\mathcal{F}_j$, then return `NO`.
3. Determine S_i, the set of all elements in U_{i-1} which are in some set in $\mathcal{F}_{i-1}$. Output $S = \bigcup_{j=1}^{i} S_j$.

We now prove the correctness of the algorithm. In Step 1, the algorithm obtains the ground set U' and the family$\mathcal{F}'$, using the algorithm of Proposition 2. Let $l \in [i-1]$ such that the algorithm answers `NO` in Step 2 for $j = l$, and otherwise let $l = i$ if it never returns a `NO` answer in Step 2.

▷ Claim 7. For all $j \in [l]$, $\mathcal{F}_j$ has at most $c(k+1)^{d-j\epsilon}$ sets.

Consider the case when the algorithm does not return a `NO` answer. Observe that the claim holds for the base case $j = 1$: $\mathcal{F}_0$ has $c(k+1)^d$ sets, and since the algorithm does not return a `NO` answer, we have $|\mathcal{F}_1| \leq c(k+1)^{d-j\epsilon}$. For induction, observe that whenever $|\mathcal{F}_j| \leq c(k+1)^{d-j\epsilon}$, the algorithm ensures that $|\mathcal{F}_{j+1}| \leq c(k+1)^{d-(j+1)\epsilon}$; otherwise, it returns a `NO` answer.

Suppose the algorithm returns a `NO` answer at some value of j in Step 2, then there are more than $c(k+1)^{d-j\epsilon}$ sets in $\mathcal{F}_j$, which have survived the repeated removal of sets from $\mathcal{F}_0$ up to this point, and they cannot be hit by any k of the elements in U_j, since each element can hit at most $c(k+1)^{d-1-j\epsilon}$ sets in $\mathcal{F}_j$. Thus, the algorithm correctly infers that the input does not have a hitting set of size at most k.

Once the algorithm has reached Step 3, the number of sets in the residual family, $\mathcal{F}_{i-1}$ is at most $(k+1)^{d-(\lceil (d-1)/\epsilon \rceil -1)\cdot\epsilon} < k^{d-((d-1)/\epsilon-1)\cdot\epsilon} = k^{1+\epsilon}$. The set S_i of elements in U_{i-1} that appear in some set in $\mathcal{F}_{i-1}$ is trivially also a hitting set. Observe that the sets of elements removed in earlier stages, i.e. $S_0, \ldots, S_{i-1}$ together hit all sets in $\mathcal{F}$ not appearing in $\mathcal{F}_{i-1}$. Thus, the set $S = \bigcup_{j=0}^{i} S_j$ output by the algorithm is a hitting set for $\mathcal{F}$.

▷ Claim 8. The set S output by the algorithm has at most $((d-1)/\epsilon + d)k^{1+\epsilon}$ elements.

For each $j \in [i-1]$, the algorithm ensures that $|\mathcal{F}_{j-1}| \leq c(k+1)^{d-(j-1)\epsilon}$ (otherwise, it returns a `NO` answer). Thus, the number of elements which appear in at least $c(k+1)^{d-1-j\epsilon}$ sets is at most $\left(c(k+1)^{d-(j-1)\epsilon}\right)/\left(c(k+1)^{d-1-j\epsilon}\right) = k^{1+\epsilon}$, i.e. $|S_j| \leq k^{1+\epsilon}$.

In Step 3, the algorithm ensures that $|\mathcal{F}_{i-1}| \leq k^{d-(i-1)\epsilon} \leq k^{1+\epsilon}$. Each set in $\mathcal{F}_{i-1}$ edges and each of these edges can span at most d elements. Thus, the number of elements in U_{i-1} which appear in some set in $\mathcal{F}_{i-1}$ $dk^{1+\epsilon}$, i.e. $|S_i| \leq dk^{1+\epsilon}$. Therefore, the total number of elements output by the algorithm in all three phases is $|S| = \sum_{j=1}^{i} |S_j| \leq (i-1)k^{1+\epsilon} + dk^{1+\epsilon} \leq (\lceil (d-1)/\epsilon \rceil + d)k^{1+\epsilon}$.

▷ Claim 9. The algorithm runs in time $n^{\mathrm{O}\left(d^2+(d/\epsilon)\right)}$ and uses $\mathrm{O}\left((d^2 + (d/\epsilon)) \log n\right)$ bits of space.

Observe that in Step 1, the family $\mathcal{F}_0$ is obtained using the algorithm of Proposition 2, which runs in time $n^{\mathrm{O}\left(d^2\right)}$ and uses $\mathrm{O}\left(d^2 \log n\right)$ bits of space (for any constant d). The output of the algorithm can now be used as an oracle for G_0.

In Step 2, each successive family $\mathcal{F}_j$ ($j \in [i-1]$) is obtained from $\mathcal{F}_{j-1}$ by deleting sets containing elements which appear in at least $k^{1-j\epsilon}$ sets (this test can be performed using $\mathrm{O}(\log n)$) bits of space. Thus, given oracle access to $\mathcal{F}_{j-1}$, an oracle for $\mathcal{F}_j$ can be provided which runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space.

Step 3 involves writing out all elements in U_{i-1} that appear in some set in $\mathcal{F}_{i-1}$, which can also be done in $\mathrm{O}(\log n)$ bits of space given oracle access to G_{i-1}. Since the number of oracles created in Step 2 is $i-1$, the various oracles together run in time $n^{\mathrm{O}(i)}$ and use $\mathrm{O}(i \log n) = \mathrm{O}((d/\epsilon) \log n)$ bits of space (Theorem 3). Combined with the $n^{\mathrm{O}\left(d^2\right)}$ time and $\mathrm{O}\left(d^2 \log n\right)$ bits of space used by the oracle of Step 1, this gives bounds of $n^{\mathrm{O}\left(d^2+(d/\epsilon)\right)}$ on the running time and $\mathrm{O}\left((d^2 + (d/\epsilon)) \log n\right)$ bits on the total space used by the algorithm. ◀

The next theorem follows from the above lemma.

▶ **Theorem 10.** *Let $\epsilon \leq 1$ be a positive number. For instances $(U, \mathcal{F})$ of d–Hitting Set with $|U| = n$, one can compute an $\mathrm{O}((d/\epsilon)n^\epsilon)$-approximate minimum hitting set in time $n^{\mathrm{O}\left(d^2+(d/\epsilon)\right)}$ using $\mathrm{O}\left((d^2 + (d/\epsilon)) \log n\right)$ bits of space.*

Proof. Starting with $k = 1$, iteratively apply the algorithm of Lemma 6 and increment k's value until the algorithm returns a family of size $\mathrm{O}\left((d/\epsilon)k^{1+\epsilon}\right)$ or $k = n^{1-\epsilon}$. When $k = n^{1-\epsilon}$ return the entire universe as the solution. As, in this case, $OPT \geq n^{1-\epsilon}$, the size of the solution produced, which is $n \leq n^\epsilon OPT$, and so we have a factor-n^ϵ approximation algorithm.

When the algorithm returns a family of size $\mathrm{O}\left((d/\epsilon)k^{1+\epsilon}\right)$ for some k, note that $OPT \geq k$ (as the algorithm returned `NO` so far), and so the solution produced is of size $\mathrm{O}((d/\epsilon)k^\epsilon k)$, which is $\mathrm{O}((d/\epsilon)n^\epsilon OPT)$ resulting in a factor-$\mathrm{O}((d/\epsilon)n^\epsilon)$ approximation algorithm. As we merely reuse the procedure of Lemma 6, the running time is $\mathrm{O}\left((d^2 + (d/\epsilon)) \log n\right)$ and the amount of space used is $\mathrm{O}\left((d^2 + (d/\epsilon)) \log n\right)$ bits. ◀

The above theorem allows us to devise space-efficient approximation algorithms for a number of *graph deletion* problems. Let Π be a hereditary class of graphs, i.e. a class closed under taking induced subgraphs. Let Φ be a set of forbidden graphs for Π such that a graph G is in Π if and only no induced subgraph of G is isomorphic to a graph in Φ. Consider the problem Del–Π (described below), defined for classes Π with finite sets Φ of forbidden graphs.

Instance G, a graph
Solution a set of vertices smallest size whose deletion yields a graph in Π

The next result is a combination of the fact that Del–Π can be formulated as a certain hitting set problem and the procedure of Theorem 10.

▶ **Lemma 11.** † *Let $\epsilon \leq 1$ be a positive number. On graphs with n vertices, one can compute $\mathrm{O}((1/\epsilon)n^\epsilon)$-approximate solutions for Del–Π in time $n^{\mathrm{O}(1/\epsilon)}$ using $\mathrm{O}((1/\epsilon) \log n)$ bits of space.*

By setting ϵ to a small positive constant or $(1/\log n)$, we obtain the following corollary, owing to the fact that for all problems appearing in it, the target graph classes are known to be characterized by a finite set of forbidden induced subgraphs (see e.g. Cygan et al. [16]) and so the problems can be formulated as Del–Π.

▶ **Corollary 12.** † *On graphs with n vertices, one can compute*

- $\mathrm{O}(n^\epsilon)$*-approximate solutions in time* $n^{\mathrm{O}(1/\epsilon)} = n^{\mathrm{O}(1)}$ *using* $\mathrm{O}((1/\epsilon)\log n) = \mathrm{O}(\log n)$ *bits of space for any positive constant* $\epsilon \leq 1$*, and*
- $\mathrm{O}(\log n)$*-approximate solutions in time* $n^{\mathrm{O}(\log n)}$ *using* $\mathrm{O}\big(\log^2 n\big)$ *bits of space*

for the problems Vertex Cover*,* Triangle-Free Deletion*,* Threshold Deletion*,* Cluster Deletion*,* Split Deletion*,* Cograph Deletion *and* Tournament FVS*.*

3.1 Vertex Cover on Graphs of Bounded Degree

We begin this section with the observation that in a directed graph with maximum outdegree 1, every connected component contains (as an induced subgraph or otherwise) at most one (undirected) cycle. For such a directed graph D, consider the graph G obtained by ignoring arc directions. Because every connected component in G also has at most one cycle, one can find a *minimum* vertex cover for G in polynomial time and logarithmic space using a modified post-order traversal procedure on the connected components. The following lemma formalizes this discussion.

▶ **Lemma 13.** † *Let D be a directed graph on n vertices with maximum outdegree* 1 *and let G be the undirected graph obtained by ignoring the arc directions in D. One can find a minimum vertex cover for G in polynomial time using* $\mathrm{O}(\log n)$ *bits of space.*

We now prove that by layering multiple applications of the above lemma, one can compute a 2-approximate minimum vertex cover in a bounded-degree graph. Our approach is inspired by a *local* distributed algorithm of Polishchuk and Suomela [33] which computes factor-3 approximations.

▶ **Theorem 14.** *There is an algorithm which takes as input a graph G on n vertices in which every vertex has degree at most Δ, and computes a* 2*-approximate minimum vertex cover for G. The algorithm runs in time* $n^{\mathrm{O}(\Delta)}$ *and uses* $\mathrm{O}(\Delta \log n)$ *bits of space.*

Proof. Set $G_0 = G$ and $V_0 = \mathrm{V}(G)$. The algorithm works in stages $1, \ldots, \Delta$ as follows. In Stage i, it enumerates the subgraph H_{i-1} of G_{i-1} in which each vertex of u of G_{i-1} only retains the edge to its i^{th} neighbour v (if it exists) in G. Observe that directing every such edge from u to v yields a directed graph R with maximum outdegree 1.

Applying the procedure of Lemma 13 with $D = R$ and $G = H_{i-1}$, the algorithm now computes a minimum vertex cover S_i for H_{i-1} in polynomial time using $\mathrm{O}(\log n)$ bits of space. It then produces the graph G_i by removing the vertex set S_i from G_{i-1} and outputs the vertices in S_i. At the end of Stage Δ, the algorithm terminates.

We now prove the bounds in the claim. Observe that the vertex set of G_i ($i \in [\Delta]$) is precisely $\mathrm{V}(G_{i-1}) \setminus S_i$. In Stage i, the algorithm only considers the vertices in G_{i-1}, so the vertex cover generated by it has no neighbours in vertex covers generated in earlier stages, i.e. $S_i \cap S_j = \emptyset$ for $j < i$.

For each H_{i-1}, consider a maximal matching M_i in H_{i-1}. From the way the various sets S_i are generated, it is easy to see that $S = \bigcup_{i=1}^{\Delta} S_i$ forms a vertex cover for G and additionally, $M = \bigcup_{i=1}^{\Delta} M_i$ is a maximal matching in G. Observe that the each set S_i also covers the matching M_i in H_{i-1}. Since S_i is a minimum vertex cover for H_{i-1}, and the endpoints of edges in M_i form a vertex cover for H_{i-1}, we have $|S_i| \leq 2|M_i|$.

As M is a maximal matching in G, the endpoints of edges in M form a vertex cover for G, and we have $|S| = \sum_{i=1}^{\Delta}|S_i| \leq 2 \cdot \sum_{i=1}^{\Delta}|M_i| \leq 2 \cdot \sum_{i=1}^{\Delta} \tau(G)$, where $\tau(G)$ is the vertex cover number of G. Thus, the set S output by the algorithm is a 2-approximate vertex cover.

Now observe that for all $i \in [\Delta]$, G_i and S_i satisfy the hypothesis of Theorem 3. Thus, one can compute each of the sets S_i in time $n^{\mathrm{O}(i)}$ using $\mathrm{O}(i \log n)$ bits of space. Since the maximum value i takes on is Δ, the algorithm runs in time $n^{\mathrm{O}(\Delta)}$ and uses a total of $\mathrm{O}(\Delta \log n)$ bits of space. ◀

4 Dominating Sets

In this section, we describe approximation algorithms for Dominating Set restricted to certain graph classes. A problem instance consists of a graph $G = (V, E)$ and $k \in \mathbb{N}$, and the objective is to determine if there is a *dominating set* of size at most k, i.e. a set $S \subseteq V$ of at most k vertices such that $S \cup \mathrm{N}(S) = V$.

The first result of this section concerns graphs excluding C_4 (a cycle on 4 vertices) as a subgraph. On such graphs, one can compute $\mathrm{O}(\sqrt{n})$-approximations using $\mathrm{O}(\log n)$ bits of space using a known kernelization algorithm [35].

4.1 C_4-Free Graphs

Any vertex $v \in \mathrm{V}(G)$ of degree at least $2k + 1$ must be in any dominating set of size at most k, as any other vertex (including a neighbour of v) can dominate at most 2 vertices in the neighbourhood (as there will be a C_4 otherwise). Using this, we establish the following result.

▶ **Lemma 15.** † *There is an algorithm which takes as input a C_4-free graph G on n vertices and $k \in \mathbb{N}$, and either determines that G has no dominating set of size at most k, or outputs a dominating set of size $\mathrm{O}(k^2)$. The algorithm runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space.*

The proof of the following corollary uses arguments very similar to those in the proof of Theorem 10, so we omit it.

▶ **Corollary 16.** *There is an algorithm which takes as input a C_4-free graph G on n vertices, and computes an $\mathrm{O}(\sqrt{n})$-approximate minimum dominating set for G. The algorithm runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space.*

4.2 Graphs of Bounded Degeneracy

A graph is called d-degenerate if there is a vertex of degree at most d in every subgraph of G. A graph with maximum degree d is clearly d-degenerate. Planar graphs are 5-degenerate. There is a generalization of the polynomial kernel for Dominating Set on C_4-free graphs to $K_{i,j}$-free graphs for any fixed $i, j \in \mathbb{N}$ [32] ($K_{i,j}$ is the complete bipartite graph with i vertices in one part and j vertices in the other). The class of $K_{i,j}$-free graphs includes C_4-free graphs and for $i \leq j$, $(i + 1)$-degenerate graphs.

This kernel however, does not seem amenable to modifications that would allow its use in computing approximate solutions using logarithmic or even polylogarithmic space. To design a space-efficient approximation algorithm for d-degenerate graphs, we resort instead to the $\mathrm{O}(d^2)$-approximation algorithm of Jones et al. [22]. To achieve an $\mathrm{O}(\log^2 n)$ bound on the space used, several adaptations are necessary.

Let G be a d-degenerate graph on n vertices. As every subgraph of G has a vertex with degree at most d, the number of edges in G is at most dn. It follows that

▶ **Observation 1.** In any subgraph of p vertices of a d degenerate graph, at least $p/2$ vertices are of degree at most $2d$.

The algorithm starts by picking the neighbours of all vertices of degree at most $2d$, and works by repeatedly finding such vertices in smaller and smaller sugraphs of G and picking all their neighbours in the solution. As the vertex or one of its neighbours must be in any dominating set, this will result in an $O(d)$ approximation if we manage to find a vertex that dominates (at least one and) at most $2d$ of the undominated vertices. This may not happen in the intermediate steps as more and more vertices are dominated by those vertices picked earlier. So we do some careful partitioning of the vertices and find low degree vertices in appropriate subgraphs.

Let Y be the set of vertices picked at any point, B be the set of vertices (other than those in Y) dominated by Y, and W be the set of vertices in $V \setminus (Y \cup B)$. The goal is to dominate vertices in W, and we try to do so by finding (the neighbours of) low degree vertices from $B \cup W$. So we start finding low degree (at most $2d$) vertices in $B \cup W$ to pick their neighbours. First we look for such vertices in B, and so we further partition B into B_h, those vertices of B with at least $2d + 1$ neighbours in W and $B_l = B \setminus B_h$.

First, we remove (for later consideration) vertices of W that have no neighbours in $W \cup B_h$, let they be W_l and focus on the induced subgraph $G[B_h \cup W_h]$ where $W_h = W \setminus W_l$. Here, we are bound to find low degree vertices from W_h (as vertices in B_h have high degree) as long as W_h is non-empty, and so we repeat the above procedure of picking the neighbours of all low degree vertices from W_h. Finally, when W_h is empty, if W_l is non-empty, we simply pick all vertices of W_l into the solution. This completes the description of the algorithm.

If we treat a round as the step where we find all vertices in W_h with at most $2d$ neighbours in W_h, then as at least a fraction of the vertices of W_h are dominated in each round due to Observation 1, the number of rounds is $O(\log n)$. Each round just requires identifying vertices based on their degrees in the resulting subgraph, the i-th round can be implemented in $O(i \log n)$ bits using Theorem 3 resulting in an $O(\log^2 n)$ bits implementation.

The approximation ratio of $O(d^2)$ can be proved formally using a charging argument (see Jones et al. [22], Theorem 4.9). We give an informal explanation here. First we argue the approximation ratio of $(2d + 1)$ for the base case when W_h is empty. Isolated vertices in W_l are isolated vertices in G and hence they need to be picked in the solution. The number of non-isolated vertices in W_l is at most $2d|B_l|$ as their neighbours are only in B_l (otherwise, by definition, those vertices will be in W_h). As vertices in B_l have degree at most $2d$, $|W_l| \leq 2d|B_l|$ and as at least one vertex of $B_l \cup W_l$ must be picked to dominate a vertex in W_l, we have the approximation ratio of $(2d + 1)$ for those vertices.

In the intermediate step, if we did not ignore vertices in B_l to dominate a vertex in W_h, a $(2d + 1)$- approximation is clear. For, a vertex or one of its at most $2d$ neighbours must be picked in the dominating set. However, a vertex in W_h maybe dominated by a vertex in B_l, but by ignoring B_l, we maybe picking $2d$ vertices to dominate it. As a vertex in B_l can dominate at most $2d$ such vertices of W_h, we get an approximation ratio of $O(d^2)$.

The next theorem formalizes the above discussion.

▶ **Theorem 17.** † *There is an algorithm which takes as input a d-degenerate graph G on n vertices and computes an $O(d^2)$-approximate minimum dominating for G. The algorithm uses $O(\log^2 n)$ bits of space and runs in time $n^{O(\log n)}$.*

4.3 Regular Graphs

On regular graphs, we can achieve a better approximation ratio in logarithmic space by derandomizing a result of Alon and Spencer [3] on the size of a dominating set on graphs with minimum degree d.

▶ **Proposition 18** (Alon and Spencer [3], Theorem 1.2.2). *Any graph on n vertices with minimum degree d has a dominating set of size at most $n(\log(d+1)+1)/(d+1)$.*

On a d-regular graph, because the size of any dominating set is at least $n/(d+1)$, the approximation ratio achieved is $\log(d+1)+1$.

Now we outline the proof of the above proposition to show how it can be derandomized. Consider a d-regular graph G on n vertices. Picking each vertex of G with probability $p = \log(d+1)/(d+1)$ yields a set S with expected size $\mathbb{E}[|S|] = np$. By adding in the vertices not dominated by S, we obtain a dominating set $W = S \cup (V \setminus (S \cup \mathrm{N}(S)))$. The expected size of this set is $\mathbb{E}[|W|] \leq n(p + (1-p)^{d+1})$, and it can be shown that this quantity is $n(\log(d+1)+1)/(d+1)$.

Note that the expectation bounds only need the sampling of the vertices to be pairwise independent. Consider a 2-universal hash family $\mathcal{F}$ for $[[n] \to [d+1]]$, and define $S_f = \{v \in \mathrm{V}(G) \mid f(v) \leq \log(d+1)+1\}$ and $W_f = S_f \cup (V \setminus (S_f \cup \mathrm{N}(S_f)))$. Over functions $f = \mathcal{F}$, the sampling probability $\mathrm{P}(v \in S_f)$ is $\lfloor(\log(d+1)+1)/(d+1)\rfloor$. Because $\mathcal{F}$ is a 2-universal hash family, there is a function $f \in \mathcal{F}$ for which W_f achieves the expectation bound for $|W|$ above.

The sampling procedure can now be derandomized as follows. Compute $\mathcal{F}$ in logarithmic space using Proposition 4 and enumerate it. For each $f \in \mathcal{F}$, determine $|W_f|$, and output W_f for the first function f for which $|W_f| \geq n(\log(d+1)+1)/(d+1)$.

We thus have the following result.

▶ **Theorem 19.** *There is an algorithm which takes as input a d-regular graph G on n vertices, and computes a $(\log(d+1)+1)$-approximate minimum dominating set for G. The algorithm runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space.*

5 Independent Sets by Randomization

In this section, we consider the Independent Set problem restricted to graphs with bounded average degree. On general graphs, the problem is unlikely to have a non-trivial (factor-$(n^{1-\epsilon})$) approximation algorithm [21]. However, if the graph has average degree d, then an independent set satisfying the bound of the next lemma is a $(2d)$-approximate solution. Note that graphs of bounded average degree encompass planar graphs and graphs of bounded degeneracy. It is also known that $2d$ is the best approximation ratio possible up to polylogarithmic factors in d [4, 14].

▶ **Proposition 20** (Alon and Spencer [3], Theorem 3.2.1). *If a graph on n vertices has average degree d, then it has an independent set of size at least $n/(2d)$.*

In what follows, we develop a logarithmic-space procedure that achieves the above bound. Let $G = (V, E)$ be a graph on n vertices with average degree d. Consider a set $S \subseteq V$ obtained by picking each vertex in V independently with probability $p = 1/d$. Let m_S be the number of edges with both endpoints in S. The following bound appears as an intermediate claim in the proof of Proposition 20 (see Alon and Spencer [3], Theorem 3.2.1). We use it here without proof.

▶ **Lemma 21.** $\mathbb{E}[|S| - m_S] = n/(2d)$.

Consider the set I obtained by arbitrarily eliminating an endpoint of each edge in $G[S]$. Observe that $G[I]$ has no edges, i.e. I is an independent set whose expected size is $\mathbb{E}[|S| - m_S] = n/(2d)$.

Derandomizing this sampling procedure is simple: simply run through the functions of a 2-universal hash family $\mathcal{F}$ for $[[n] \to [d]]$ and for each $f \in \mathcal{F}$, pick a vertex $v \in V$ into S if and only if $f(v) = 1$. Because the range of the functions is $[d]$, the sampling probability is $\mathrm{P}(v \in S) = 1/d$. Recall that Lemma 21 only requires the sampling procedure to be pairwise independent, so the expectation bound remains the same: $\mathbb{E}[|S| - m_S] = n/(2d)$. While going through $\mathcal{F}$, select the function $f \in \mathcal{F}$ which maximizes $|S| - m_S$, where $S = \{v \in V \mid f(v) = 1\}$ and m_S is the number of edges $uv \in E$ with $f(u) = f(v) = 1$. Using the construction of Proposition 4, this step can be performed in polynomial time using $\mathrm{O}(\log n)$ bits of space and f can be used as an oracle for S at the same space cost.

The next step, in which vertices are deleted arbitrarily from each pair of adjacent vertices in the sample S, is tricky to carry out in small space. This is because for any edge uv in $G[S]$, it is not possible to determine whether either of the endpoints survive the deletion procedure without additional information about the other edges incident with u and v. However, there is a simple fix for this: retain only those vertices in S which are the smallest vertices in their neighbourhoods in $G[S]$. Using this, we prove the following lemma.

▶ **Lemma 22.** † *Let T be the set of vertices $v \in S$ such that v is the smallest vertex (in the original arbitrary labelling) in its neighbourhood in $G[S]$. The set T is independent in G, has size $|T| \geq |S| - m_S$, and one can compute S in polynomial time using $\mathrm{O}(\log n)$ bits of space.*

We now have the following theorem as a direct consequence of the above results.

▶ **Theorem 23.** *There is a an algorithm which takes as input a graph G on n vertices with average degree d, and computes a $(2d)$-approximate maximum independent set in G. The algorithm runs in polynomial time and uses $\mathrm{O}(\log n)$ bits of space.*

6 Conclusion

We devised space efficient approximation algorithms for d–Hitting Set (and its restriction Vertex Cover), Independent Set and Dominating Set in some special classes of graphs.

We consider our contribution as simply drawing attention to a direction in the study of approximation algorithms, and believe that it should be possible to improve the approximation ratios and the space used for the problems considered here. Obtaining a constant-factor or even factor-$\mathrm{O}(\log n)$ approximation algorithm for Vertex Cover and a factor-$\mathrm{O}(\log n)$ approximation algorithm for Dominating Set on general graphs using $\mathrm{O}(\log n)$ bits of space are some specific open problems of interest.

References

1 Eric Allender and Meena Mahajan. The complexity of planarity testing. *Information and Computation*, 189(1):117–134, February 2004. doi:10.1016/j.ic.2003.09.002.
2 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, April 2006. doi:10.1145/1150334.1150336.
3 Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, Hoboken, NJ, USA, third edition, 2008.
4 Per Austrin, Subhash Khot, and Muli Safra. Inapproximability of Vertex Cover and Independent Set in Bounded Degree Graphs. *Theory of Computing*, 7(1):27–43, March 2011. doi:10.4086/toc.2011.v007a003.
5 Paul Beame. A general Sequential Time-Space Tradeoff for Finding Unique Elements. *SIAM Journal on Computing*, 20(2):270–277, April 1991. doi:10.1137/0220017.

6 Ioana O. Bercea, Navin Goyal, David G. Harris, and Aravind Srinivasan. On Computing Maximal Independent Sets of Hypergraphs in Parallel. *ACM Transactions on Parallel Computing*, 3(1):1–13, January 2017. doi:10.1145/2938436.

7 Bonnie Berger, John Rompel, and Peter W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. *Journal of Computer and System Sciences*, 49(3):454–477, December 1994. doi:10.1016/S0022-0000(05)80068-6.

8 Allan Borodin and Stephen A. Cook. A Time-Space Tradeoff for Sorting on a General Sequential Model of Computation. *SIAM Journal on Computing*, 11(2):287–297, May 1982. doi:10.1137/0211022.

9 Allan Borodin, Michael J. Fischer, David G. Kirkpatrick, Nancy A. Lynch, and Martin Tompa. A time-space tradeoff for sorting on non-oblivious machines. *Journal of Computer and System Sciences*, 22(3):351–364, June 1981. doi:10.1016/0022-0000(81)90037-4.

10 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, June 1993. doi:10.1137/0222038.

11 Liming Cai, Jianer Chen, Rodney G. Downey, and Michael R. Fellows. Advice classes of parameterized tractability. *Annals of Pure and Applied Logic*, 84(1):119–138, March 1997. doi:10.1016/S0168-0072(95)00020-8.

12 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, April 1979. doi:10.1016/0022-0000(79)90044-8.

13 Sankardeep Chakraborty, Anish Mukherjee, Venkatesh Raman, and Srinivasa Rao Satti. A Framework for In-place Graph Algorithms. In *26th Annual European Symposium on Algorithms*, volume 112, pages 13:1 – 13:16, Helsinki, Finland, August 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/lipics.esa.2018.13.

14 Siu On Chan. Approximation Resistance from Pairwise-Independent Subgroups. *Journal of the ACM*, 63(3):1–32, September 2016. doi:10.1145/2873054.

15 Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Selection and Sorting in the "Restore" Model. *ACM Transactions on Algorithms*, 14(2):1–18, June 2018. doi:10.1145/3168005.

16 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer-Verlag, Cham, Switzerland, 2015.

17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th Annual Symposium on Theory of Computing*, pages 624–633, New York, NY, USA, May 2014. ACM Press. doi:10.1145/2591796.2591884.

18 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proceedings of the 46th Annual Symposium on Theory of Computing*, pages 383–392, New York, NY, USA, May 2014. ACM Press. doi:10.1145/2591796.2591865.

19 Stefan Fafianie and Stefan Kratsch. A Shortcut to (Sun)Flowers: Kernels in Logarithmic Space or Linear Time. In *Mathematical Foundations of Computer Science*, volume 9235, pages 299–310, Milan, Italy, August 2015. Springer-Verlag. doi:10.1007/978-3-662-48054-0_25.

20 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System Sciences*, 34(1):19–26, February 1987. doi:10.1016/0022-0000(87)90002-X.

21 Johan Håstad. Clique is hard to approximate within $n\hat{(}1 - \epsilon)$. *Acta Mathematica*, 182(1):105–142, March 1999. doi:10.1007/BF02392825.

22 Mark Jones, Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Ondřej Suchý. Parameterized Complexity of Directed Steiner Tree on Sparse Graphs. *SIAM Journal on Discrete Mathematics*, 31(2):1294–1327, January 2017. doi:10.1137/15M103618X.

23 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2 - ϵ. *Journal of Computer and System Sciences*, 74(3):335–349, May 2008. doi:10.1016/j.jcss.2007.06.019.

24 Jason Li and Ryan O'Donnell. Bounding Laconic Proof Systems by Solving CSPs in Parallel. In *Proceedings of the 29th Annual Symposium on Parallelism in Algorithms and Architectures*, pages 95–100, Washington, DC, USA, July 2017. ACM Press. `doi:10.1145/3087556.3087557`.

25 Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1053, November 1986. `doi:10.1137/0215074`.

26 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the 25th Annual Smposium on Theory of Computing*, pages 448–457, San Diego, CA, USA, May 1993. ACM Press. `doi:10.1145/167088.167211`.

27 Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, May 2014. `doi:10.1145/2627692.2627694`.

28 J. Ian Munro and Michael S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, November 1980. `doi:10.1016/0304-3975(80)90061-4`.

29 J. Ian Munro and Venkatesh Raman. Selection from read-only memory and sorting with minimum data movement. *Theoretical Computer Science*, 165(2):311–323, October 1996. `doi:10.1016/0304-3975(95)00225-1`.

30 Rasmus Pagh and Jakob Pagter. Optimal time-space trade-offs for non-comparison-based sorting. In *Proceedings of the 13th Annual Symposium on Discrete Algorithms*, pages 9–18, Philadelphia, PA, USA, January 2002. SIAM. `doi:10.5555/545381.545383`.

31 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 264–268, Palo Alto, CA, USA, November 1998. IEEE Comput. Soc. Press. `doi:10.1109/SFCS.1998.743455`.

32 Geevarghese Philip, Venkatesh Raman, and Somnath Sikdar. Polynomial kernels for dominating set in graphs of bounded degeneracy and beyond. *ACM Transactions on Algorithms*, 9(1):1–23, December 2012. `doi:10.1145/2390176.2390187`.

33 Valentin Polishchuk and Jukka Suomela. A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, 109(12):642–645, May 2009. `doi:10.1016/j.ipl.2009.02.017`.

34 Venkatesh Raman and Sarnath Ramnath. Improved upper bounds for time-space trade-offs for selection. *Nordic Journal of Computing*, 6(2):162–180, June 1999. `doi:10.5555/762350.762354`.

35 Venkatesh Raman and Saket Saurabh. Short Cycles Make W-hard Problems Hard: FPT Algorithms for W-hard Problems in Graphs with no Short Cycles. *Algorithmica*, 52(2):203–225, October 2008. `doi:10.1007/s00453-007-9148-9`.

36 John H. Reif. Symmetric Complementation. *Journal of the ACM*, 31(2):401–421, March 1984. `doi:10.1145/62.322436`.

37 Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55(4):1–24, September 2008. `doi:10.1145/1391289.1391291`.

38 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, April 1970. `doi:10.1016/S0022-0000(70)80006-X`.

39 Maria Serna. Approximating linear programming is log-space complete for P. *Information Processing Letters*, 37(4):233–236, February 1991. `doi:10.1016/0020-0190(91)90194-M`.

40 Till Tantau. Logspace Optimization Problems and Their Approximability Properties. *Theory of Computing Systems*, 41(2):327–350, August 2007. `doi:10.1007/s00224-007-2011-1`.

41 Luca Trevisan. Parallel Approximation Algorithms by Positive Linear Programming. *Algorithmica*, 21(1):72–88, May 1998. `doi:10.1007/PL00009209`.

42 Luca Trevisan and Fatos Xhafa. The Parallel Complexity of Positive Linear Programming. *Parallel Processing Letters*, 08(04):527–533, December 1998. `doi:10.1142/S0129626498000511`.

43 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag, Berlin, Germany, 1999.

44 Tomoyuki Yamakami. Uniform-Circuit and Logarithmic-Space Approximations of Refined Combinatorial Optimization Problems. In *Combinatorial Optimization and Applications*, volume 8287, pages 318–329, Chengdu, China, December 2013. Springer-Verlag. `doi:10.1007/978-3-319-03780-6_28`.