Local Algorithms for Bounded Degree Sparsifiers in Sparse Graphs*

Shay Solomon

IBM Research, Yorktown Heights, NY, USA
solo.shay@gmail.com

Abstract

In graph sparsification, the goal has almost always been of *global* nature: compress a graph into a smaller subgraph (*sparsifier*) that maintains certain features of the original graph. Algorithms can then run on the sparsifier, which in many cases leads to improvements in the overall runtime and memory. This paper studies sparsifiers that have bounded (maximum) degree, and are thus *locally* sparse, aiming to improve local measures of runtime and memory. To improve those local measures, it is important to be able to compute such sparsifiers *locally*.

We initiate the study of local algorithms for bounded degree sparsifiers in unweighted sparse graphs, focusing on the problems of vertex cover, matching, and independent set. Let $\epsilon > 0$ be a slack parameter and $\alpha \geq 1$ be a density parameter. We devise local algorithms for computing:

1. A $(1+\epsilon)$-vertex cover sparsifier of degree $O(\alpha/\epsilon)$, for any graph of *arboricity* α.[1]
2. A $(1+\epsilon)$-maximum matching sparsifier and also a $(1+\epsilon)$-maximal matching sparsifier of degree $O(\alpha/\epsilon)$, for any graph of arboricity α.
3. A $(1+\epsilon)$-independent set sparsifier of degree $O(\alpha^2/\epsilon)$, for any graph of average degree α.

Our algorithms require only a single communication round in the standard message passing models of distributed computing, and moreover, they can be simulated locally in a trivial way. As an immediate application we can extend results from distributed computing and local computation algorithms that apply to graphs of degree bounded by d to graphs of arboricity $O(d/\epsilon)$ or average degree $O(d^2/\epsilon)$, at the expense of increasing the approximation guarantee by a factor of $(1+\epsilon)$. In particular, we can extend the plethora of recent local computation algorithms for approximate maximum and maximal matching from bounded degree graphs to bounded arboricity graphs with a negligible loss in the approximation guarantee.

The inherently local behavior of our algorithms can be used to amplify the approximation guarantee of any sparsifier in time roughly linear in its size, which has immediate applications in the area of dynamic graph algorithms. In particular, the state-of-the-art algorithm for maintaining $(2-\epsilon)$-vertex cover (VC) is at least linear in the graph size, even in dynamic forests. We provide a reduction from the dynamic to the static case, showing that if a t-VC can be computed from scratch in time $T(n)$ in any (sub)family of graphs with arboricity bounded by α, for an arbitrary $t \geq 1$, then a $(t+\epsilon)$-VC can be maintained with update time $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$, for any $\epsilon > 0$. For planar graphs this yields an algorithm for maintaining a $(1+\epsilon)$-VC with constant update time for any constant $\epsilon > 0$.

1998 ACM Subject Classification G.4. Design and analysis of algorithms

Keywords and phrases arboricity, bounded degree, local algorithm, sparsifier

Digital Object Identifier 10.4230/LIPIcs.ITCS.2018.52

[*] This work was partially supported by the Herman Goldstine Postdoctoral Fellowship.
[1] In a graph of arboricity α the average degree of any induced subgraph is at most 2α.

9th Innovations in Theoretical Computer Science Conference (ITCS 2018).
Editor: Anna R. Karlin; Article No. 52; pp. 52:1–52:19
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Graph sparsification has been extensively studied for many years, and is subject to increasingly growing interest due to the rapidly growing necessity of dealing with huge-sized graphs. Given such a graph $G = (V, E)$, we would like to *compress* G into a subgraph H of much smaller size that maintains certain features of G, such as distances, cuts or flows. Algorithms can then run on the compressed subgraph H, sometimes called *sparsifier*, rather than the original graph G, which may save significantly on important resources such as the overall runtime and memory of the algorithm, often at the expense of approximate rather than exact solutions or worse approximation guarantees. The most common type of sparsifiers are *edge sparsifiers*, such as graph spanners [49] and cut or spectral sparsifiers [11, 52], which span the original vertex set using a small number of edges. Another well-studied type of sparsifiers are *vertex sparsifiers*, such as flow or cut sparsifiers [32, 43, 39], which should span a small number of designated vertices.

The basic goal in this area has almost always been of *global* nature, i.e., of minimizing the overall size of the sparsifier and the overall time needed for computing it. One of the exceptions is in the area of spanners, where researchers have studied spanners of bounded (maximum) *degree*. While a sparse spanner has a low average degree, and is thus globally sparse, a bounded degree spanner has a low maximum degree, and is thus locally sparse. Although bounded degree spanners have been little studied thus far in general graphs [20, 19], they have been studied extensively in Euclidean low-dimensional spaces, see e.g., [23, 3, 24, 29, 25]). The spanner degree often determines local memory constraints when using spanners to construct network synchronizers [49] and efficient broadcast protocols [5, 6]. In compact routing schemes (e.g., [53, 18]), the use of low degree spanners may enable the routing tables to be of small size. Moreover, viewing vertices as processors, in many applications the degree of a processor represents its *load*, hence a low degree spanner guarantees that the load on all the processors in the network will be low.

This paper studies sparsifiers from a *local* perspective, aiming to improve local measures of runtime and memory. To improve those local measures, it is important to be able to compute such sparsifiers *locally*, in a manner to be defined shortly. We initiate the study of local algorithms for *bounded degree* sparsifiers in unweighted *sparse* graphs. The graphs that we consider are sparse either globally, i.e., of bounded average degree, or uniformly, i.e., of bounded *arboricity*, whence the average degree of any induced subgraph is bounded. In sparse graphs some vertices may have large degrees, as with the n-star graph. Our basic goal is to compute *locally* a sparsifier H for the original graph $G = (V, E)$, whose maximum degree is bounded in terms of the density of G and some slack parameter $\epsilon > 0$, and which approximately preserves a certain property or feature of the original graph; the sparsifier would ideally be a subgraph of G, but this cannot always be achieved. Algorithms, particularly local ones, can then run on the bounded degree sparsifier H rather than on the original graph G, which may save significantly on local resources of runtime and memory. For concreteness, we focus on the following combinatorial optimization problems: (approximate) minimum vertex cover (VC), maximum and maximal matching, and maximum independent set (IS). It would be only natural to extend the study of bounded degree sparsifiers to other fundamental problems.

For the maximum matching problem, a $(1+\epsilon)$-sparsifier for G is a *subgraph* $H = (V', E')$ of G, with $V' \subseteq V, E' \subseteq E$, such that the maximum matching size of H is within a factor of $1+\epsilon$ from that of G; thus a $(1+\epsilon)$-(approximate) maximum matching for H is a $(1+O(\epsilon))$-maximum matching for G. For the maximal matching problem the definition is similar; see Section 2.

We need to be more careful with the definitions of sparsifier for the minimum VC and maximum IS problems, since a VC (respectively, IS) for a subgraph H of G may not be a *valid* VC (resp., IS) for the entire graph G. Note that we are concerned with the validity of solutions obtained by the sparsifier rather than the approximations that they provide. Consequently, a sparsifier in these cases will not be simply a subgraph H of G, but rather a pair (H, V'), where H is a subgraph of G and V' is a vertex set of V, hereafter the *validating set* of the sparsifier, such that for any VC (resp., IS) for H, adding (resp., removing) the validating set V' to (resp., from) it provides a valid VC (resp., IS) for G; the role of the validating set is to translate the solution obtained by the sparsifier into a valid solution for G. We say that (H, V') is a $(1+\epsilon)$-sparsifier for G if for any $(1+\epsilon)$-(approximate minimum) VC (resp., (approximate maximum) IS) for H, denoted by C_H (resp., I_H), the set $C_H \cup V'$ is a valid $(1+O(\epsilon))$-VC (resp., $I_H \setminus V'$ is a $(1+O(\epsilon))$-IS) for G.

Given any $\epsilon > 0$ and any density parameter $\alpha \geq 1$, we devise *local* algorithms for computing:

1. A $(1+\epsilon)$-VC sparsifier of degree $O(\alpha/\epsilon)$, for any graph of *arboricity* bounded by α.
2. A $(1+\epsilon)$-maximum matching sparsifier and also a $(1+\epsilon)$-maximal matching sparsifier of degree $O(\alpha/\epsilon)$, for any graph of arboricity bounded by α.
3. A $(1+\epsilon)$-IS sparsifier of degree $O(\alpha^2/\epsilon)$, for any graph of average degree bounded by α.

Aiming at enhancing the applicability and usefulness of our sparsifiers, we adhere to a strict notion of locality: For any vertex v, we want to be able to compute the adjacent edges of v that belong to the sparsifier by *probing* only v and a small number (bounded by the degree of the sparsifier) of its neighbors, where the probing procedure is context-dependent. In standard centralized settings such a procedure will simply examine the data structures of v and those neighbors, but in the message passing models of distributed computing, for example, the procedure may trigger the exchange of messages between v and those neighbors. Also, we want to determine if a vertex v belongs to the validating set of the sparsifier by probing only v. The advantage in using such a strict notion of locality is three-fold, as summarized here and described in more detail later on:

1. In the rapidly growing area of *local computation algorithms* (see, e.g., [50, 2, 26, 27]), a standard assumption is that the underlying graph has bounded degree. This assumption is required since a local computation algorithm would typically probe all vertices inside a small-radius ball around the queried vertex/edge. If the maximum degree is Δ and the ball radius is r, the probe complexity is bounded by $\Delta^{O(r)}$, and sometimes the total runtime and space will also be bounded by $\Delta^{O(r)}$. Due to the local nature of our sparsification algorithms, we can restrict the probing procedure only to the sparsifier edges, which directly enables us to extend known results from bounded degree graphs to uniformly sparse graphs.
2. In dynamic centralized graph algorithms, following an update of a vertex/edge, the update algorithm would typically scan all neighbors of the updated vertices (and usually more than just those vertices), either to obtain up-to-date information from the data structures of those neighbors or to update that information. Since the adversary may choose to focus its attention on few high degree vertices, this could lead to algorithms with a poor update time. Using our notion of locality, we show that the attention can be restricted to only few edges of the sparsifier, which leads to improvements in the update time.
3. In distributed networks, we can compute the sparsifier in a single communication round. Moreover, since each vertex communicates with only few of its neighbors, the *load* on all vertices (or processors) throughout the sparsifier's computation is low. After the sparsifier has been computed, running on it distributed algorithms rather than on the original

network may significantly reduce the total runtime of the algorithms and the load on the processors.

In addition to the above applications, our sparsification algorithms can be used more broadly in computational models where there are local memory constraints, such as the distributed communication model and the massively parallel computation (MPC) model, which is an abstraction of MapReduce-style frameworks (cf. [21, 4]). Another relevant model is the dynamic distributed model (cf. [46, 17]), where some graph structure (e.g., matching) is to be maintained in a dynamically changing distributed network using low local memory at processors.

1.1 Our sparsifiers

Perhaps the most important feature of our sparsification algorithms is their simplicity, which is partly why they can be computed under such a strict notion of locality.

For any $\Delta \geq 1$, let V^{Δ}_{high} and V^{Δ}_{low} be the sets of vertices of degree $\geq \Delta$ and $< \Delta$, respectively. When Δ is clear from the context, we may omit it from the superscript. For any vertex set V' in G, denote by $G[V']$ the subgraph induced by V'. Define $G_{high} = G[V_{high}]$ and $G_{low} = G[V_{low}]$.

1. For the minimum VC problem, we take the pair (G_{low}, V_{high}) as the $(1+\epsilon)$-sparsifier for G, where G_{low} is a subgraph of G and V_{high} is the validating set of the sparsifier. It is clear that the degree of G_{low} is at most Δ, and moreover, for any VC for G_{low}, its union with V_{high} is a valid VC for G. In Section 3.2 we show that for any graph of arboricity α, taking $\Delta = O(\alpha/\epsilon)$ guarantees the following: If VC_{low} is a $(1+\epsilon)$-VC for G_{low}, then $VC_{low} \cup V_{high}$ is a $(1+O(\epsilon))$-VC for G.
2. For the maximum and maximal matching problems, a (subgraph) $(1+\epsilon)$-sparsifier G_{Δ} for G with degree bounded by Δ can be obtained as follows: Mark up to Δ arbitrary adjacent edges on every vertex v, and add to G_{Δ} all edges that are marked by both endpoints. It is clear that the degree of G_{Δ} is at most Δ. (Note that if we took to G_{Δ} edges that are marked just once, the degree of G_{Δ} could explode.) In Section 3.1 we show that for any graph of arboricity α, taking $\Delta = O(\alpha/\epsilon)$ guarantees that the subgraph G_{Δ} is a $(1+\epsilon)$-sparsifier for G.
3. For the maximum IS problem, we take G_{low} as the $(1+\epsilon)$-sparsifier for G. (Although we may also use a validating set for the sparsifier, there is no need to do that here; thus in this case the IS sparsifier is a subgraph of G.) It is clear that the degree of G_{low} is at most Δ, and moreover, any IS for G_{low} is a valid IS for G. In Section 3.3 we show that for any graph of average degree α, taking $\Delta = O(\alpha^2/\epsilon)$ guarantees that any $(1+\epsilon)$-IS for G_{low} is a $(1+O(\epsilon))$-IS for G.

Note that our sparsifiers are obtained by essentially "ignoring" the high degree vertices, where what is meant by ignoring is context-dependent. For the minimum VC and maximum IS problems, we take all high degree vertices to the VC and take none of them to the IS, respectively, whereas for the maximum and maximal matching problems, we ignore all but at most Δ edges adjacent on any high degree vertex. This approach of ignoring the high degree vertices can be viewed as a general paradigm, and it would be interesting to apply it to additional fundamental graph problems.

1.2 Local computation algorithms

The model of *local computation algorithms* was introduced by Rubinfeld et al. [50], motivated by the fact that it is prohibitively expensive and sometimes infeasible for an algorithm to read and process the entire input as well as to report the entire output, when dealing with massive data sets. Local computation algorithms should answer *queries* regarding global solutions to computational problems by examining only a small part of the input. The goal is to reach a global solution by performing local (sublinear time) computations on the input, and answer only regarding the queried part of the output. If there are multiple possible solutions, the answers to all queries must be consistent with a single solution. (More technical details on this model are given in Section 2; see also [50].) For the $(1+\epsilon)$-maximum matching problem, each query is an edge in the graph, and the algorithm needs to answer whether the queried edge belongs to a $(1+\epsilon)$-maximum matching; note that the answers to all queries must be with respect to the same matching. [41] devised a randomized local computation algorithm for $(1+\epsilon)$-maximum matching with time and space complexities $\texttt{poly}(\log n) \cdot \exp(\Delta)$, where Δ is the maximum degree of the graph. This result was improved in [26] to a deterministic algorithm with time complexity $O(\log^* n) \cdot \exp(\Delta)$ and zero space complexity. [40] devised a randomized algorithms with time and space complexities of $\texttt{poly}(\log n, \Delta)$. [28] obtained a deterministic algorithm for $(2+\epsilon)$-maximum matching with time complexity $O(\log^* n) \cdot 2^{O(\Delta^2)}$. (We ignore the dependencies on ϵ in the results of [41, 26, 40, 28]; in fact, in some of these results it is assumed that ϵ is constant.)

We can extend the results of [41, 26, 40, 28] from graphs of bounded degree to graphs of bounded arboricity. Specifically, for any graph with arboricity bounded by α, our matching sparsifier G_Δ has a degree bounded by $\Delta = O(\alpha/\epsilon)$. We get this extension by exploiting the local nature of G_Δ, and in particular, the fact that for any vertex v, we can compute the adjacent edges of v that belong to G_Δ by probing only v and at most Δ of its neighbors. Any $(1+\epsilon)$-maximum matching computed for the sparsifier provides a $(1+\epsilon)^2 = (1+O(\epsilon))$-maximum matching for the original graph, thus there is only a negligible loss in the approximation guarantee. Since $\Delta = O(\alpha/\epsilon)$, the smaller ϵ is, the larger the time and space complexities get. Nonetheless, as long as ϵ is not too small, the loss here is quite negligible too. In this way reduce the problem of approximate maximum matching from graphs of arboricity bounded by α to graphs of degree bounded by $\approx \alpha$.

In the same way we reduce the problems of approximate minimum VC and maximum IS from bounded arboricity graphs and graphs of bounded average degree, respectively, to bounded degree graphs. These reductions show that if and when local computation algorithms for these problems are developed in bounded degree graphs (there are currently no such algorithms), they will immediately give rise to new algorithms in the respective wider families. Moreover, this can be viewed as a general paradigm: By locally computing a sparsifier for a combinatorial optimization problem in some family of graphs, we reduce the problem from that family to the family of bounded degree graphs, and the loss depends on the approximation guarantee of the sparsifier and on its degree.

1.3 Dynamic centralized graph algorithms

The problems of dynamically maintaining approximate minimum VC and maximum matching have been intensively studied in recent years, see e.g. [45, 10, 44, 30, 14, 13, 15]. The holy grail is for the approximation guarantee to approach 1 and for the (amortized or worst-case) update time to be $\texttt{poly}(\log n)$ and ideally a constant.

A dynamic algorithm for approximate VC (respectively, matching) should maintain a data structure that answers queries of whether a vertex is in the VC (resp., an edge is matched) or not in *constant* time. Constant query time is considered a standard requirement in this line of research, and the goal is to optimize the update time of the algorithm under this requirement. Almost all related works follow another requirement, of bounding the number of *changes* to the maintained structure per step, either in the amortized or in the worst-case sense. The update time of the algorithm, which is the time it needs to update the data structure, may be significantly lower than, and is bounded by, the number of changes to the maintained structure; for a motivation of this requirement, refer to [16, 1]. It is easy to see that maintaining an exact minimum VC or a maximum matching requires $\Omega(n)$ changes per update even in the amortized sense, and even for a simple path that changes dynamically in a straightforward way.

Except for general graphs, these problems have been studied mostly in bounded arboricity graphs [44, 38, 34, 12, 13, 48]. It was shown in [44] that a maximal matching can be maintained with amortized time $O(\log n/\log\log n)$ in constant arboricity graphs, and this bound was improved in [34] to $O(\sqrt{\log n})$. [38] achieved a worst-case update time of $O(\log n)$. The algorithms of [44, 34, 38] extend to graphs with arboricity bounded by α, with the update time depending on α. A randomized algorithm for maintaining a maximal matching in *general graphs* with constant amortized update time was given in [51]. A maximal matching provides a 2-approximation for both the maximum matching and the minimum VC.

What about better-than-2 approximations? Improving upon [12, 13] and providing essentially the best result one can hope for in graphs of arboricity α, [48] showed that a $(1+\epsilon)$-maximum matching can be maintained with a worst-case update time of $O(\alpha)$. The $O(\alpha)$ bound in [48] also bounds the number of changes to the matching, and as mentioned it is impossible to maintain an exact matching with $o(n)$ matching changes even for a dynamic path. In addition, [48] showed that a $(2+\epsilon)$-VC can be maintained with a worst-case update time of $O(\alpha)$. This improves the update time of the 2-VC algorithms of [44, 34, 38] in every aspect, at the expense of increasing the approximation guarantee from 2 to $(2+\epsilon)$.

Note that in general graphs, a better-than-2 approximation to the minimum VC cannot be maintained efficiently under the unique games conjecture [37]. Although this hardness result does not apply to bounded arboricity graphs, there is currently no dynamic algorithm for maintaining a better-than-2 approximate VC with update time $o(n)$ even in the amortized sense, and even in dynamic forests![2] In fact, the only known way to maintain a better-than-2 VC dynamically is to apply the fastest static algorithm from scratch following every update step.

The local nature of our sparsification algorithms can be used to amplify the approximation guarantee of our VC sparsifier in time roughly linear in its size. As a corollary, we provide a reduction from the dynamic to the static case, showing that if a t-VC can be computed from scratch in time $T(n)$ in any (sub)family of graphs with arboricity bounded by α, for any $t \geq 1$, then a $(t+\epsilon)$-VC can be maintained with a worst-case update time of $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$. This bound of $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$ also bounds the amortized number of changes to the VC. For planar graphs this yields an algorithm for maintaining an $(1+\epsilon)$-VC with a constant worst-case update time for any constant $\epsilon > 0$, which is essentially the best one can hope for. For graphs of arboricity bounded by α we can maintain a VC of approximation guarantee $\approx 2 - \frac{1}{\alpha}$ with a worst-case update time of $O(\sqrt{n}\cdot\alpha^2)$.

We can also amplify our matching and IS sparsifiers and obtain reductions from the dynamic to the static case. These reductions are not useful to obtain new time bounds for the

[2] The only exception is an algorithm for maintaining a maximum matching in dynamic forests with a worst-case update time of $O(\log n)$ [31]. As a result one can maintain the *size* of the minimum VC (by Konig's theorem) in logarithmic update time. On the negative side, one cannot efficiently maintain the VC itself or even a poor approximation of it using [31], and more importantly, the result of [31] requires a logarithmic query time, hence it does not follow the standard constant query time requirement. (We believe that [31] is the only paper in this line of research that does not follow this requirement.)

maximum matching and IS problems. In particular, for approximate matchings, the result of [48] is already the best one can hope for; nevertheless, our reduction for the maximum matching problem can be used to obtain simpler and cleaner algorithms and arguments than those of [48], as discussed in Section 4.2.

1.4 Distributed networks

Our sparsification algorithms can be implemented within a single communication round in distributed networks, where each processor sends and receives a single $O(1)$-bit message along each of its adjacent edges. Moreover, if Δ is the maximum degree of the sparsifier, each processor may send messages along just Δ of its adjacent edges, which ensures that the *load* on all the processors will be low throughout the sparsifier's computation. After the sparsifier has been computed, we can run on it the required distributed algorithm rather than on the original network, which may significantly reduce the total runtime of the algorithm, the load on the processors, and in some settings it may also reduce the local memory usage at a processor.

Our distributed sparsification algorithms directly extend results from bounded degree graphs to bounded arboricity graphs or to graphs of bounded average degree, for all the problems studied in this paper. Since the performance of many distributed algorithm depend on the maximum degree of the underlying network and as our sparsification algorithms are extremely simple, we anticipate that they will be used and implemented in practice.

For the distributed approximate VC problem, [8] showed how to compute a $(2+\epsilon)$-VC in $O(\log \Delta / (\epsilon \log \log \Delta))$ rounds, where Δ is the maximum degree in the graph. We can plug our reduction to extend the result of [8] to graphs of arboricity bounded by α, getting a $(2+\epsilon)$-VC in $O(\log(\alpha/\epsilon)/(\epsilon \log \log(\alpha/\epsilon)))$ rounds.

For the distributed approximate matching problem, a reduction from bounded arboricity graphs to bounded degree graphs was already given in [22]. Nonetheless, our reduction has several advantages over that of [22] (see Section 4), one of which is that it is much simpler, another is that our degree bound has better dependence on ϵ. In particular, [27] devised a distributed algorithm for computing a $(1+\epsilon)$-maximum matching in $\Delta^{O(1/\epsilon)} + O(\epsilon^{-2}) \cdot \log^* n$ rounds. Plugging our reduction (instead of that from [22]), we easily extend the result of [27] to graphs of arboricity bounded by α to get a $(1+\epsilon)$-maximum matching in $(\alpha/\epsilon)^{O(1/\epsilon)} + O(\epsilon^{-2}) \cdot \log^* n$ rounds.

A reduction from bounded arboricity graphs to bounded degree graphs was given in [9] for the problems of maximal matching, maximal IS, vertex coloring and ruling sets. The reduction of [9] is based on different ideas than ours (their algorithm is randomized, the number of rounds required by their algorithm is polylogarithmic in the maximum degree, etc), and moreover, it appears that the reduction of [9] cannot be efficiently applied to the problems studied in this paper.

1.5 Organization

The definitions and notation used throughout are given in Section 2. Our matching, VC and IS sparsifiers are presented in Sections 3.1, 3.2 and 3.3, respectively. Finally, in Section 4 we provide some applications of our sparsifiers.

2 Preliminaries

Consider an unweighted graph $G = (V, E)$. A matching M for G is said to be *almost-maximal* w.r.t. some parameter $\eta > 0$, or *η-maximal*, if at most $\eta \cdot |M|$ edges can be added to it while keeping it a valid matching for G. A $(1+\epsilon)$-maximal matching sparsifier for G is a subgraph H of G, such that any η-maximal matching for H is an $(\epsilon + O(\eta))$-maximal matching for G; in particular, a maximal matching for H is ϵ-maximal for G, and a ϵ-maximal matching for H is $O(\epsilon)$-maximal for G.

For a vertex v in G, let $\Gamma_G(v)$ denote the set of neighbors (or *neighborhood*) of v in G. For any vertex set $V' \subseteq V$ in G, denote by $G[V']$ the subgraph induced by V'.

A graph $G = (V, E)$ has *arboricity* α if $\alpha = \max_{U \subseteq V} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$, where $E(U) = \{(u, v) \in E \mid u, v \in U\}$. Alternatively, the arboricity of a graph is the minimum number of edge-disjoint forests into which it can be partitioned. The family of bounded arboricity graphs contains planar and bounded genus graphs, bounded tree-width graphs, and in general all graphs excluding fixed minors.

As mentioned in Section 1.1, for any $\Delta \geq 1$, we write V^{Δ}_{high} and V^{Δ}_{low} to denote the sets of vertices of degree $\geq \Delta$ and $< \Delta$, respectively, omitting Δ from the superscript when it is clear from the context. Define $G_{high} = G[V_{high}]$ and $G_{low} = G[V_{low}]$.

3 Our Sparsifiers

Note that a graph with arboricity α has an average degree at most 2α. Throughout we use α as an arboricity parameter and β as an average degree parameter. The next observation will be useful.

▶ **Observation 1.** *Let $G = (V_1 \cup V_2, E)$ be a graph with average degree bounded by β, and suppose that each vertex of V_1 has degree at least $(c+1)\beta$, for any c. Then $|V_1| \leq |V_2|/c$.*

Proof. Observe that $2|E| \leq \beta(|V_1|+|V_2|)$. Since every vertex in U has degree at least $(c+1)\beta$, we have $2|E| \geq |V_1| \cdot (c+1)\beta$, hence $|V_1| \cdot (c+1)\beta \leq \beta(|V_1| + |V_2|)$, and so $|V_1| \leq |V_2|/c$. ◀

3.1 The matching sparsifier

Let G be a graph of arboricity bounded by α, set $\Delta = 5(5/\epsilon + 1)2\alpha$, and define the sets V_{high}, V_{low} and the subgraphs G_{low}, G_{high} accordingly. We assume that $\epsilon \leq 1$; the argument works also for larger ϵ, by increasing Δ appropriately. (We did not try to optimize the constants in the definition of Δ.)

Recall our definition of the matching sparsifier G_Δ: Mark up to Δ arbitrary adjacent edges on every vertex v, and add to G_Δ all edges that are marked by both endpoints. It is clear that the degree of G_Δ is at most Δ. To prove that G_Δ is a matching sparsifier, we use Hall's marriage theorem.

▶ **Theorem 2** (Hall's marriage theorem [33]). *Let G be a bipartite graph with sides X and Y. There is a matching that entirely covers X if and only if for every subset W of X, $|W| \leq |\Gamma_G(W)|$, where $\Gamma_G(W) = \bigcup_{v \in W} \Gamma_G(v)$ is the* neighborhood *of W.*

The following theorem shows that G_Δ is a $(1+\epsilon)$-maximum matching sparsifier.

▶ **Theorem 3.** *Let G be a graph of arboricity bounded by α and define G_Δ as above, for $\Delta = 5(5/\epsilon + 1)2\alpha, \epsilon \leq 1$. Also, denote by $\mathcal{M}^*$ and $\mathcal{M}^*_\Delta$ the maximum matchings for G and G_Δ, respectively. Then $|\mathcal{M}^*| \leq (1+\epsilon) \cdot |\mathcal{M}^*_\Delta|$. (In particular, any t-matching for G_Δ is a $t(1+\epsilon)$-matching for G.)*

Proof. We shall construct a matching $\mathcal{M}_\Delta$ for G_Δ satisfying $|\mathcal{M}^*| \leq (1+\epsilon) \cdot |\mathcal{M}_\Delta|$.

Let $\mathcal{M}^*_1$ be the subset of $\mathcal{M}^*$ of all edges that belong to G_Δ, and initialize $\mathcal{M}_\Delta$ to $\mathcal{M}^*_1$. Define $\mathcal{M}^*_2 = \mathcal{M}^* \setminus \mathcal{M}^*_1$ as the complementary subset of $\mathcal{M}^*$. We next show that sufficiently many additional edges of G_Δ can be added to $\mathcal{M}_\Delta$ while keeping it a valid matching.

Note that any edge with two endpoints in V_{low} must belong to G_Δ. Since the edges of $\mathcal{M}^*_2$ do not belong to G_Δ by definition, any edge of $\mathcal{M}^*_2$ is adjacent on at least one vertex of V_{high}.

Let $V^{in} = V^{in}_{high}$ be the subset of V_{high} of all vertices with at least $2\Delta/5$ neighbors in V_{high}, and let $V^{out} = V^{out}_{high} = V_{high} \setminus V^{in}$ be the complementary subset of V_{high}. Observe that G_{high} has arboricity at most α, and thus average degree at most 2α. Moreover, every vertex in V^{in} has degree at least $2\Delta/5 = 2(5/\epsilon+1)2\alpha \geq (10/\epsilon+1)2\alpha$ in G_{high}. Observation 1 thus yields

$$|V^{in}| \ \leq \ \epsilon/10 \cdot |V^{out}|. \tag{1}$$

▶ **Observation 4.** *Each vertex in V^{out} has more than $3\Delta/5$ neighbors in V_{low} within G_Δ.*

Proof. First, note that any vertex in V_{low} marks all its $< \Delta$ adjacent edges. Since each vertex in V^{out} has $< 2\Delta/5$ neighbors in V_{high} but a degree of $\geq \Delta$, the remaining $> 3\Delta/5$ neighbors must be in V_{low}. Each vertex of V^{out} marks Δ edges, and any of the $> 3\Delta/5$ edges adjacent to a vertex of V_{low} is also marked by that endpoint in V_{low}, and is thus added to G_Δ. ◀

For a vertex v, we will refer to its neighbors within G_Δ as its *G_Δ-neighbors.* Let $U = U^{out}$ be the set of vertices in V^{out} that are free w.r.t. $\mathcal{M}^*_1$. By Observation 4, each vertex of U has more than $3\Delta/5$ G_Δ-neighbors in V_{low}. Denote by $\Gamma = \Gamma(U)$ the set of all G_Δ-neighbors of vertices from U in V_{low}. We next partition Γ into two sets, the sets $\Gamma_{matched}$ and Γ_{free} of vertices that are matched and free w.r.t. $\mathcal{M}^*_1$, respectively. A vertex $u \in U$ is called *risky* if at least $2\Delta/5$ of its G_Δ-neighbors in Γ are in $\Gamma_{matched}$, otherwise it is *safe*, and then more than $\Delta/5$ of its G_Δ-neighbors are in Γ_{free}. Let U_{risky} and U_{safe} be the sets of all risky and safe vertices of U, respectively.

▶ **Claim 5.** $|U_{risky}| \leq \epsilon/5 \cdot |\mathcal{M}^*_1|$.

Proof. For each vertex $u \in U_{risky}$, let $\Gamma_{matched}(u)$ be the set of its at least $2\Delta/5$ G_Δ-neighbors in $\Gamma_{matched}$. Let $\Gamma_{risky} = \bigcup_{u \in U_{risky}} \Gamma_{matched}(u)$ denote the union of the sets $\Gamma_{matched}(u)$ over all vertices $u \in U_{risky}$. Since all vertices in Γ_{risky} are matched w.r.t. the matching $\mathcal{M}^*_1$, $|\Gamma_{risky}| \leq 2|\mathcal{M}^*_1|$. Observe that the subgraph G_{risky} of G_Δ induced by $U_{risky} \cup \Gamma_{risky}$ has arboricity at most α, and thus average degree at most 2α. Moreover, each vertex in U_{risky} has at least $2\Delta/5$ neighbors in Γ_{risky}, and thus its degree in G_{risky} is at least $2\Delta/5 = 2(5/\epsilon+1)2\alpha \geq (10/\epsilon+1)2\alpha$. Observation 1 thus yields $|U_{risky}| \ \leq \ \epsilon/10 \cdot |\Gamma_{risky}| \leq \epsilon/5 \cdot |\mathcal{M}^*_1|$. ◀

Note that any edge in G_Δ between a vertex in U_{safe} and a vertex in Γ_{free} is vertex-disjoint to all edges of $\mathcal{M}^*_1$. Consequently, the following claim implies that at least $|U_{safe}|$ edges of G_Δ can be added to the matching $\mathcal{M}_\Delta$ while preserving its validity.

▶ **Claim 6.** *There exists a matching $\mathcal{M}_{safe}$ in G_Δ that entirely covers U_{safe} by edges between U_{safe} and Γ_{free}. In particular, $|\mathcal{M}_{safe}| = |U_{safe}|$.*

Proof. For each vertex $u \in U_{safe}$, let $\Gamma_{free}(u)$ be the set of its at least $\Delta/5$ G_Δ-neighbors in Γ_{free}. Let $\Gamma_{safe} = \bigcup_{u \in U_{safe}} \Gamma_{free}(u)$ denote the union of the sets $\Gamma_{free}(u)$ over all vertices $u \in U_{safe}$. Consider the induced bipartite subgraph G_{safe} of G_Δ with sides U_{safe} and Γ_{safe}. We argue that for any subset $W \subseteq U_{safe}$, its neighborhood in G_{safe}, namely,

$$\Gamma_{G_{safe}}(W) \;=\; \bigcup_{v \in W} \Gamma_{G_{safe}}(v) \;=\; \bigcup_{v \in W} \Gamma_{free}(v),$$

is of larger size. Obsserve that the subgraph G^W_{safe} of G_{safe} induced by the vertex set $W \cup \Gamma_{G_{safe}}(W)$ has arboricity at most α, and thus average degree at most 2α. Moreover, each vertex of W has at least $\Delta/5$ neighbors in $\Gamma_{G_{safe}}(W)$, i.e., its degree in G^W_{safe} is at least $\Delta/5 = (5/\epsilon + 1)2\alpha$. Observation 1 thus yields $|W| \le \epsilon/5 \cdot |\Gamma_{G_{safe}}(W)| < |\Gamma_{G_{safe}}(W)|$. By Hall's marriage theorem (Theorem 2), there exists a matching $\mathcal{M}_{safe}$ in G_{safe} that entirely covers U_{safe}. Claim 6 follows. ◀

We add all edges in the matching $\mathcal{M}_{safe}$ guaranteed by Claim 6 to $\mathcal{M}_\Delta$, so that $\mathcal{M}_\Delta = \mathcal{M}^*_1 \cup \mathcal{M}_{safe}$. Since $\mathcal{M}^*$ is a maximum matching for G that is a disjoint union of $\mathcal{M}^*_1$ and $\mathcal{M}^*_2$ and as $\mathcal{M}_\Delta$ is a disjoint union of $\mathcal{M}^*_1$ and $\mathcal{M}_{safe}$, it follows that $|\mathcal{M}^*_2| \ge |\mathcal{M}_{safe}|$. Although the vertices of V^{in} may be matched w.r.t. $\mathcal{M}^*_2$, we have $|V^{in}| \le \epsilon/10 \cdot |V^{out}|$ by Equation (1). By definition, all vertices of $V^{out} \setminus U$ are matched w.r.t. $\mathcal{M}^*_1$, thus $|V^{out} \setminus U| \le 2|\mathcal{M}^*_1||$. Hence $|V^{out}| \le 2|\mathcal{M}^*_1| + |U|$, and so

$$|V^{in}| \;\le\; \epsilon/10 \cdot |V^{out}| \;\le\; \epsilon/10 \cdot (2|\mathcal{M}^*_1| + |U_{risky}| + |U_{safe}|). \tag{2}$$

Since any edge of $\mathcal{M}^*_2$ is adjacent on at least one vertex of V_{high} and as all vertices of $V^{out} \setminus U$ are matched w.r.t. $\mathcal{M}^*_1$, $|\mathcal{M}^*_2| \le |V^{in}| + |U_{risky}| + |U_{safe}|$. Combined with Equation (2) and Claim 5,

$$\begin{aligned}
|\mathcal{M}^*_2| &\le |V^{in}| + |U_{risky}| + |U_{safe}| \;\le\; \epsilon/10 \cdot (2|\mathcal{M}^*_1| + |U_{risky}| + |U_{safe}|) + |U_{risky}| + |U_{safe}| \\
&\le \epsilon/10 \cdot (2|\mathcal{M}^*_1| + \epsilon/5 \cdot |\mathcal{M}^*_1| + |M^*_2|) + \epsilon/5 \cdot |\mathcal{M}^*_1| + |M_{safe}| \\
&= (\epsilon/5 + \epsilon^2/50 + \epsilon/5) \cdot |\mathcal{M}^*_1| + \epsilon/10 \cdot |M^*_2| + |M_{safe}| \\
&\le \epsilon/2 \cdot |M^*_1| + \epsilon/10 \cdot |M^*_2| + |M_{safe}|,
\end{aligned}$$

yielding

$$\begin{aligned}
|\mathcal{M}_\Delta| &= |\mathcal{M}^*_1| + |\mathcal{M}_{safe}| \;\ge\; |\mathcal{M}^*_1| + |\mathcal{M}^*_2| - \epsilon/2 \cdot |\mathcal{M}^*_1| - \epsilon/10 \cdot |\mathcal{M}^*_2| \\
&\ge (1 - \epsilon/2) \cdot (|\mathcal{M}^*_1| + |\mathcal{M}^*_2|) \;=\; (1 - \epsilon/2) \cdot |\mathcal{M}^*|.
\end{aligned}$$

To complete the proof of Lemma 3, observe that

$$|\mathcal{M}^*| \le \frac{1}{1 - \epsilon/2} \cdot |\mathcal{M}_\Delta| \;\le\; (1+\epsilon) \cdot |\mathcal{M}_\Delta| \;\le\; (1+\epsilon) \cdot |\mathcal{M}^*_\Delta|.$$ ◀

The following theorem shows that G_Δ is a $(1+\epsilon)$-maximal matching sparsifier.

▶ **Theorem 7.** *Let G be a graph of arboricity bounded by α and G_Δ defined as above, for $\Delta = 5(5/\epsilon + 1)2\alpha, \epsilon \le 1$. Any η-maximal matching for G_Δ is an $(\epsilon + 3\eta)$-maximal matching for G, for any $\eta > 0$.*

Proof. The proof of this theorem is similar to that of Theorem 3, thus we aim for conciseness.

Consider an arbitrary η-maximal matching $\mathcal{M}_\Delta = \mathcal{M}^\eta_\Delta$ for G_Δ, and let $\mathcal{M}'_\Delta$ be any matching for G obtained by adding edges to $\mathcal{M}_\Delta$. We will show that $|\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta| \le (\epsilon + 2\eta) \cdot |\mathcal{M}_\Delta|$.

Since $\mathcal{M}_\Delta$ is η-maximal w.r.t. G_Δ, at most $\eta \cdot |\mathcal{M}_\Delta|$ edges of $\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta$ may belong to G_Δ; in what follows we refer to those edges as *special edges* and to the remaining edges of $\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta$ as ordinary edges. Denote the set of ordinary edges in $\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta$ by O. Since any edge with two endpoints in V_{low} belongs to G_Δ, it follows that any edge of O has at least one endpoint in V_{high}. Denote the set of vertices in V_{high} that are free w.r.t. $\mathcal{M}_\Delta$ by F_{high}, and note that $|O| \le |F_{high}|$.

Defining the sets V^{in} and V^{out} as in the proof of Theorem 3, Equation (1) yields $|V^{in}| \le \epsilon/10 \cdot |V^{out}|$. Notice that $|V^{out}| \le 2|\mathcal{M}_\Delta| + |F_{high} \cap V^{out}|$, hence

$$|F_{high} \cap V^{in}| \ \le \ |V^{in}| \ \le \ \epsilon/10 \cdot |V^{out}| \ \le \ \epsilon/10 \cdot (2|\mathcal{M}_\Delta| + |F_{high} \cap V^{out}|). \qquad (3)$$

Observation 4 from the proof of Theorem 3 remains valid, and it implies that each vertex in $F_{high} \cap V^{out}$ has more than $3\Delta/5$ G_Δ-neighbors in V_{low}. Denote by $\Gamma = \Gamma(F_{high} \cap V^{out})$ the set of all G_Δ-neighbors of vertices from $F_{high} \cap V^{out}$ in V_{low}. We next partition Γ into two sets, the sets $\Gamma_{matched}$ and Γ_{free} of vertices in Γ that are matched and free w.r.t. $\mathcal{M}_\Delta$, respectively. A vertex $f \in F_{high} \cap V^{out}$ is called *risky* if at least $2\Delta/5$ of its G_Δ-neighbors are in $\Gamma_{matched}$, otherwise it is *safe*, and then more than $\Delta/5$ of its G_Δ-neighbors are in Γ_{free}. Let F_{risky} and F_{safe} be the sets of all risky and safe vertices of $F_{high} \cap V^{out}$, respectively. Following similar lines as those in the proof of Claim 5, we get $|F_{risky}| \le \epsilon/5 \cdot |\mathcal{M}_\Delta|$. Also, following similar lines as those in the proof of Claim 6, we establish the existence of a matching $\mathcal{M}_{safe}$ in G_Δ that entirely covers F_{safe} by edges between F_{safe} and Γ_{free}. Since all edges of $\mathcal{M}_{safe}$ belong to G_Δ and can be added to $\mathcal{M}_\Delta$ while keeping it a valid matching for G_Δ, the fact that $\mathcal{M}_\Delta$ is η-maximal w.r.t. G_Δ yields $|F_{safe}| \le \eta \cdot |\mathcal{M}_\Delta|$. It follows that $|F_{high} \cap V^{out}| = |F_{risky}| + |F_{safe}| \le \epsilon/5 \cdot |\mathcal{M}_\Delta| + \eta \cdot |\mathcal{M}_\Delta|$. Recall that $|O| \le |F_{high}|$. Combined with Equation (3), we conclude that

$$\begin{aligned} |O| \ &\le \ |F_{high}| \ = \ |F_{high} \cap V^{in}| + |F_{high} \cap V^{out}| \\ &\le \ \epsilon/10 \cdot (2|\mathcal{M}_\Delta| + |F_{high} \cap V^{out}|) + \epsilon/5 \cdot |\mathcal{M}_\Delta| + \eta \cdot |\mathcal{M}_\Delta| \\ &\le \ \epsilon/10 \cdot (2|\mathcal{M}_\Delta| + \epsilon/5 \cdot |\mathcal{M}_\Delta| + \eta \cdot |\mathcal{M}_\Delta|) + \epsilon/5 \cdot |\mathcal{M}_\Delta| + \eta \cdot |\mathcal{M}_\Delta| \\ &\le \ (\epsilon/5 + \epsilon^2/50 + \epsilon \cdot \eta/10 + \epsilon/5 + \eta) \cdot |\mathcal{M}_\Delta| \ < \ (\epsilon + 2\eta) \cdot |\mathcal{M}_\Delta|, \end{aligned}$$

i.e., the number $|O|$ of ordinary edges in $\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta$ is at most $(\epsilon + 2\eta) \cdot |\mathcal{M}_\Delta|$. Recall that the number of special edges in $\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta$ is at most $\eta \cdot |\mathcal{M}_\Delta|$, thus $|\mathcal{M}'_\Delta \setminus \mathcal{M}_\Delta| \le (\epsilon + 3\eta) \cdot |\mathcal{M}_\Delta|$. ◀

3.2 The VC sparsifier

Let G be a graph of arboricity bounded by α, set $\Delta = (1/\epsilon + 1) \cdot 2\alpha$, and define the sets V_{high}, V_{low} and the subgraph G_{low} accordingly. To prove that the pair (G_{low}, V_{high}) is a VC sparsifier, we use the next lemma.

▶ **Lemma 8.** *Let VC be an arbitrary VC for G, and let $U = U_{high}$ be the set of vertices in V_{high} that are not in VC. Then $|U| \le \epsilon \cdot |VC|$.*

Proof. Denote by $\Gamma = \Gamma_{high}$ the set of neighbors in $G \setminus U$ of all vertices of U, and note that $\Gamma \subseteq VC$, as otherwise VC cannot be a VC for G. Observe that the subgraph $G' = (U \cup \Gamma, E') = G[U \cup \Gamma]$ induced by $U \cup \Gamma$ has arboricity at most α, and thus average degree at most 2α. Moreover, every vertex in U has degree at least Δ. Observation 1 thus yields $|U| \ \le \ \epsilon \cdot |\Gamma| \ \le \ \epsilon \cdot |VC|$. ◀

The following theorem shows that (G_{low}, V_{high}) is a $(1 + \epsilon)$-VC sparsifier.

▶ **Theorem 9.** *Let G be a graph of arboricity bounded by α. Also, let VC^* be a minimum VC for G, let VC^t_{low} be a t-VC for G_{low}, for any $t \geq 1$, and define $\widetilde{VC} = VC^t_{low} \cup V_{high}$. Then $\widetilde{VC}$ is a $(t+\epsilon)$-VC for G.*

Proof. Since VC^* is a VC for G, $VC^* \cap V_{low}$ must be a VC for G_{low}. Hence $|VC^t_{low}| \leq t \cdot |VC^* \cap V_{low}|$. Denoting by $U = U_{high}$ the set of vertices in V_{high} that are not in VC^*, we have $|V_{high}| = |VC^* \cap V_{high}| + |U|$. Also, Lemma 8 yields $|U| \leq \epsilon \cdot |VC^*|$. Since $|VC^t_{low}| \leq t \cdot |VC^* \cap V_{low}|$, it follows that

$$\begin{aligned}|\widetilde{VC}| = |VC^t_{low}| + |V_{high}| &= |VC^t_{low}| + |VC^* \cap V_{high}| + |U| \\ &\leq t \cdot |VC^* \cap V_{low}| + |VC^* \cap V_{high}| + \epsilon \cdot |VC^*| \leq (t+\epsilon) \cdot |VC^*|.\end{aligned}$$ ◀

3.3 The IS sparsifier

Let G be a graph of average degree bounded by β (the arboricity may be much larger than β). Set $\Delta = ((\beta+1)/\epsilon + 1) \cdot \beta$, and define the sets V_{high}, V_{low} and the subgraph G_{low} accordingly. We assume that $\beta \geq 1$, as we may ignore isolated vertices (adding all of them to the independent set). To show that G_{low} is an IS sparsifier, we make the following observation.

▶ **Observation 10.** *Let V_1 be any set of vertices in an arbitrary graph $G = (V, E)$, and let V_2 be the complementary subset of V. Let IS^*, IS^*_1 and IS^*_2 be maximum independent sets for the graph G and its subgraphs $G[V_1]$ and $G[V_2]$, respectively. Then $|IS^*_1| + |IS^*_2| \geq |IS^*|$.*

Proof. Both $IS^* \cap V_1$ and $IS^* \cap V_2$ are ISs, hence $|IS^* \cap V_1| \leq |IS^*_1|, |IS^* \cap V_2| \leq |IS^*_2|$. ◀

The following theorem shows that G_{low} is an $(1+\epsilon)$-IS sparsifier

▶ **Theorem 11.** *Let G be a graph of average degree bounded by β. Also, let IS^* (respectively, IS^*_{low}) be a maximum IS for G (resp., G_{low}), let IS^t_{low} be a t-IS for G_{low}, for any $t \geq 1$. Then IS^t_{low} is a $t(1+\epsilon)$-IS for G, for any $\epsilon < \beta$.*

Proof. Since IS^t_{low} is an IS for G_{low}, it must also be an IS for G.

Since the average degree in $G = (V_{low} \cup V_{high}, E)$ is bounded by β and as every vertex in V_{high} has degree at least $\Delta = ((\beta+1)/\epsilon + 1) \cdot \beta$, Observation 1 implies that $|V_{high}| \leq \epsilon \cdot (|V_{low}|/(\beta+1))$.

Since $\epsilon < \beta$, we have $\Delta = ((\beta+1)/\epsilon + 1) \cdot \beta > \beta$, hence there is at least one vertex of degree less than Δ, i.e., V_{low} is non-empty. Denote the average degree in G_{low} by β_{low}. Since every vertex in V_{high} has degree at least $\Delta = ((\beta+1)/\epsilon + 1) \cdot \beta$, we have $|V_{high}| \cdot ((\beta+1)/\epsilon + 1) \cdot \beta + |V_{low}| \cdot \beta_{low} \leq 2|E| \leq \beta(|V_{low}| + |V_{high}|)$. It follows that $|V_{high}| \cdot ((\beta+1)/\epsilon) \leq |V_{low}|(1 - \frac{\beta_{low}}{\beta})$, which, together with the fact that V_{low} is non-empty, yields $\beta_{low} \leq \beta$. By Turan's theorem, $|IS^*_{low}| \geq V_{low}/(\beta_{low}+1)$, hence

$$|V_{high}| \leq \epsilon \cdot (|V_{low}|/(\beta+1)) \leq \epsilon \cdot (|V_{low}|/(\beta_{low}+1)) \leq \epsilon \cdot |IS^*_{low}|.$$

By Observation 10, $|IS^*| \leq |V_{high}| + |IS^*_{low}|$, so $|IS^*| \leq (1+\epsilon) \cdot |IS^*_{low}| \leq t(1+\epsilon) \cdot |IS^t_{low}|$. ◀

4 Applications

4.1 Local computation algorithms

Each vertex v is represented as a unique ID from $\{1, \ldots, n\}$, and the graph $G = (V, E)$ is given through an adjacency list oracle $\mathcal{O}_G$ that answers neighbor queries: given a vertex

$v \in V$ and an index i, the ith neighbor of v is returned if v's degree is $\geq i$; otherwise a null sign is returned. Consider first our matching sparsifier G_Δ, obtained by marking up to Δ arbitrary adjacent edges on every vertex v, and adding to G_Δ all edges that are marked by both endpoints. Recall that $\Delta = O(\alpha/\epsilon)$, where α is a bound on the arboricity of G. Our goal is to simulate the execution of any local computation algorithm for approximate matching entirely within our bounded degree sparsifier, and in this way to reduce the problem from bounded arboricity graphs to bounded degree graphs. To this end we simply need to be *consistent* about the adjacent edges of a vertex v that we mark, e.g., for every vertex v, mark its first Δ neighbors on its adjacency list. Whenever a vertex is queried/probed, there is no need to probe any other neighbor of this vertex besides the first Δ on its adjacency list, since none of the edges that lead to the other neighbors is in the sparsifier. To determine which among these neighbors is also its neighbor in the sparsifier, we perform a symmetric probe for each of the (at most) Δ neighbors. In this way any probing procedure of the original graph (which may contain high degree vertices) is restricted to the matching sparsifier, at the cost of increasing the time complexity by at most a factor of Δ; this loss is considered negligible, since all the time and space complexities of algorithms in this area are anyway at least polynomial in Δ. Moreover, by Theorems 3 and 7, any $(1+\epsilon)$-maximum (respectively, ϵ-maximal) matching computed for G_Δ provides a $(1+O(\epsilon))$-maximum (resp., $O(\epsilon)$-maximal) matching for the original graph, thus there is only a negligible loss in the approximation guarantee. We remark that the local computation algorithm of [28] is actually an almost-maximal matching algorithm, and the $(2+\epsilon)$-approximation guarantee holds as any ϵ-maximal matching is also a $(2+O(\epsilon))$-maximum matching. In this way we reduce the problems of approximate-maximum matching and almost-maximal matching from graphs of arboricity bounded by α to graphs of degree bounded by $O(\alpha/\epsilon)$.

▶ **Corollary 12.** *For any graph G with arboricity bounded by α and for any constant $\epsilon > 0$:*

- *There is a deterministic local computation algorithm for $(1+\epsilon)$-maximum matching with time complexity $O(\log^* n) \cdot \exp(\alpha)$ and zero space complexity. (An extension of [26].)*
- *There is a randomized local computation algorithms for $(1+\epsilon)$-maximum matching with time and space complexities of $\mathtt{poly}(\log n, \alpha)$. (An extension of [40].)*
- *There is a deterministic local computation algorithm for $(2+\epsilon)$-maximum matching with time complexity $O(\log^* n) \cdot 2^{O(\alpha^2)}$. (An extension of [28].)*

For our VC and IS sparsifiers, things are even simpler. The IS sparsifier is simply G_{low}, i.e., the subgraph induced on the low degree vertices, hence simulating the execution of a local computation algorithm entirely within the sparsifier can be naturally done without having to probe more than Δ neighbors of any vertex. As for the VC sparsifier, we also need to reason about the validating set V^{high}, but for any vertex we can determine if it belongs to V^{high} or not by making one query to the oracle $\mathcal{O}_G$.

4.2 Dynamic centralized algorithms

In this section we employ our sparsifiers to get efficient dynamic algorithms. The starting point is a "lazy scheme" due to [30] for maintaining approximate maximum matching for general graphs, which was refined for bounded arboricity graphs in [48]. This scheme exploits a basic *stability* property of matchings: The size of the maximum matching changes by at most 1 following each update. Thus if we have a *large* matching of size close to the maximum, it will remain close to it throughout a long update sequence, or formally:

▶ **Lemma 13 (Lemma 3.1 in [30]).** *Let $\epsilon, \epsilon' \leq 1/2$. Suppose that $\mathcal{M}_i$ is a $(1+\epsilon)$-MCM for G_i. For $j = i, i+1, \ldots, i + \lfloor \epsilon' \cdot |\mathcal{M}_i| \rfloor$, let $\mathcal{M}_i^{(j)}$ denote the matching $\mathcal{M}_i$ after removing from* *it all edges that got deleted during the updates $i+1,\dots,j$. Then $\mathcal{M}_i^{(j)}$ is a $(1+2\epsilon+2\epsilon')$-MCM for the graph G_j.*

Hence, we can compute a $(1+\epsilon/4)$-maximum matching $\mathcal{M}_i$ at a certain update i, and use the same matching $\mathcal{M}_i^{(j)}$ throughout all updates $j = i, i+1, \dots, i' = i + \lfloor \epsilon/4 \cdot |\mathcal{M}_i| \rfloor$. (By Lemma 13, $\mathcal{M}_i^{(j)}$ is a $(1+\epsilon)$-maximum matching for all graphs G_j.) Next compute a fresh $(1+\epsilon/4)$-maximum matching $\mathcal{M}_{i'}$ following update i' and use it throughout all updates $i', i'+1, \dots, i' + \lfloor \epsilon/4 \cdot |\mathcal{M}_{i'}| \rfloor$, and repeat.

In this way the static time complexity of computing a $(1+\epsilon)$-maximum matching $\mathcal{M}$ is amortized over $1 + \lfloor \epsilon/4 \cdot |\mathcal{M}| \rfloor = \Omega(\epsilon \cdot |\mathcal{M}|)$ updates. The key insight behind the schemes of [30, 48] is not to compute the approximate matching on the entire graph, but rather on a matching sparsifier, which is derived from an $O(1)$-VC that is maintained dynamically *by other means*. [48] showed that an ϵ-maximal matching, and thus a $(2+\epsilon)$-VC, denoted by $\widetilde{VC}$, can be maintained with $O(\alpha/\epsilon)$ update time, and the argument used by [48] was quite tricky; we will get back to this point soon. Specifically, the sparsifier $\tilde{G}$ of [48] contains (1) all edges in the subgraph $G[\widetilde{VC}]$ induced by $\widetilde{VC}$, and (2) for each vertex $v \in \widetilde{VC}$, (up to) $O(\alpha/\epsilon)$ edges connecting it with arbitrary neighbors outside $\widetilde{VC}$. Although the resulting sparsifier may have large degree, it is easy to verify that it contains $O(|\mathcal{M}| \cdot \alpha/\epsilon)$ edges, and it can be computed in time linear in its size. Since a $(1+\epsilon)$-maximum matching can be computed for the sparsifier $\tilde{G}$ in time $O(|\tilde{G}|/\epsilon) = O(|\mathcal{M}| \cdot \alpha/\epsilon^2)$ [36, 42, 54], the resulting amortized update time is $O(\alpha \cdot \epsilon^{-3})$. Also, one can easily translate the amortized bound into a worst-case bound, as shown in [30].

The aforementioned stability property also applies to the minimum VC and maximum IS problems. We next consider the minimum VC problem; see Section 4.2.1 for a discussion on the maximum IS problem. Thus the size of the minimum VC changes by at most 1 following each update. In extending the lazy scheme [30, 48] to the minimum VC problem, the challenge is to efficiently compute a high quality VC sparsifier. In particular, the $(1+\epsilon)$-matching sparsifier $\tilde{G}$ of [48] cannot be used as such a VC sparsifier, since a VC for it (regardless of its approximation guarantee) may not provide a *valid* VC for the graph G.

Fix an arbitrary parameter $t \geq 1$. Consider an arbitrary (sub)family of n-vertex graphs $\mathcal{G}$ with arboricity bounded by α that is closed under edge removals (such as the family of planar graphs), and suppose that we can compute a t-VC from scratch in time $T(n)$ for any graph in this family. (More generally, we assume that for any j-vertex subgraph H of any $G \in \mathcal{G}$, for any $1 \leq j \leq n$, we can compute in time $T(j)$ a t-VC for H.) Next, we adapt the lazy scheme [30, 48] to maintain a $(t+2\epsilon)$-VC in dynamically changing graphs of $\mathcal{G}$.

Lemma 13 easily extends to the minimum VC problem and to any approximation $t \geq 1$, i.e., any $(t+\epsilon)$-VC continues to provide a $(t+2\epsilon)$-VC throughout a long update sequence. Once the approximation guarantee of that VC, denoted by VC_{old}, becomes too poor (i.e., reaches $t+2\epsilon$), we shall *amplify* it (i.e., reduce it back to $t+\epsilon$) by computing a $(t+\epsilon)$-VC within time close to linear in $|VC_{low}|$. Having done that, we can then re-use the new amplified VC, denoted by VC_{new}, throughout the subsequent $\epsilon \cdot |VC_{new}|$ update steps, and repeat.

The computation of the new amplified cover VC_{new} employs the old cover VC_{old} and our VC sparsifier from Section 3.2 as follows. Recall that the vertex set V_{high} is the validating set of our VC sparsifier. It is straightforward to maintain all vertices in V_{high} dynamically with $O(\alpha/\epsilon)$ worst-case update time; we initialize VC_{new} as V_{high}. We next need to compute the subgraph G_{low} induced on the vertices V_{low} of degree $< \Delta = O(\alpha/\epsilon)$. This can be done by simply adding, for each vertex of VC_{old} of degree $< \Delta$, all its adjacent edges to vertices of degree $< \Delta$. Although some vertices of V_{low} may not belong to VC_{old}, we must have added all edges of G_{low}, as VC_{old} is a VC for G. Clearly, the number $|V_{low}|$ of vertices in G_{low}, as well as the time needed to compute it, are bounded by $O(|VC_{old}| \cdot \alpha/\epsilon)$. We proceed by computing a t-VC for G_{low}, denoted by VC^t_{low}; the runtime of this static computation is $T(|V_{low}|)$. Finally, we add all vertices of VC^t_{low} to VC_{new}, thus we have $VC_{new} = VC^t_{low} \cup V_{high}$. By Theorem 9, VC_{new} is a $(t+\epsilon)$-VC for the entire graph G.

Observe that the overall runtime of computing VC_{new} is bounded by $O(|VC_{old}| \cdot \alpha/\epsilon) + T(|V_{low}|) \leq T(O(|VC_{old}| \cdot \alpha/\epsilon))$. Since this runtime is amortized over $\Theta(\epsilon \cdot |VC_{old}|)$ update steps, the amortized update time is bounded by $T(O(|VC_{old}| \cdot \alpha/\epsilon))/\Theta(\epsilon \cdot |VC_{old}|)$, which is no greater than $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$. Note that the amortized number of changes to the VC is also bounded by $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$. Moreover, one can easily translate the amortized update time into the same (up to a constant factor) worst-case update time, as shown in [30].

Summarizing, we have proved the following result.

▶ **Theorem 14.** *Fix arbitrary $t \geq 1, \epsilon > 0$. If a t-VC can be computed in time $T(n)$ in an arbitrary (sub)family of n-vertex graphs with arboricity bounded by α that is closed under edge removals, then a $(t+\epsilon)$-VC can be maintained with a worst-case update time of $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$. Moreover, the amortized number of changes to the VC is also bounded by $\frac{T(n)}{O((n/\alpha)\cdot\epsilon^2)}$.*

Remarks.

1. For planar graphs, one can compute a $(1+\epsilon)$-VC in time $O(n)$, for any constant $\epsilon > 0$ [7]. By Theorem 14, we can maintain a $(1+\epsilon)$-VC with a constant worst-case update time for any constant $\epsilon > 0$.
2. For the family of graphs with arboricity bounded by α, one can compute a $(2 - \frac{2}{\beta+1})$-VC in time $O(n^{3/2} \cdot \alpha)$, where β is the average degree in some (carefully computed) subgraph, and is thus bounded by 2α [35]. By Theorem 14, we can maintain a VC with an approximation of roughly $2 - \frac{1}{\alpha}$ and a worst-case update time of $O(\sqrt{n} \cdot \alpha^2)$.

Our matching sparsifier from Section 3.1 can be used to simplify the algorithms of [48] and their analysis. First, as mentioned, [48] used a tricky argument to maintain a $(2+\epsilon)$-VC with $O(\alpha/\epsilon)$ update time. An alternative simpler approach is to maintain a maximal matching on top of our bounded degree matching sparsifier, which can be done naively with update time linear in the degree of the sparsifier, namely, $O(\alpha/\epsilon)$. By Theorem 7, this yields an ϵ-maximal matching, which is translated into an $(2+\epsilon)$-VC in the obvious way. Second, as explained in [48], the lazy scheme of [30] is inherently non-local. Since local algorithms are advantageous, [48] also devised a local algorithm for maintaining a $(1+\epsilon)$-maximum matching, which is quite intricate. An alternative simpler approach is to maintain a $(1+\epsilon)$-maximum matching on top of our bounded degree matching sparsifier, by dynamically excluding augmenting paths of length $O(1/\epsilon)$. By Theorem 3, this yields an $(1+O(\epsilon))$-maximum matching, and the update time of the resulting algorithm is $(\alpha/\epsilon)^{O(1/\epsilon)}$, just as in [48].

4.2.1 The dynamic approximate maximum IS problem

The size of the maximum IS changes by at most 1 following each update, hence we can apply the lazy scheme of [30, 48] also for this problem. Notice, however, that there is no need to compute a sparsifier here, since the maximum IS is of size at least $n/(\beta+1)$ by Turan's theorem, where β is the average degree in the graph. Hence, one can simply compute an approximate maximum IS on the entire graph, and amortize the cost of this static computation over $\Theta(\epsilon \cdot (n/\beta))$ update steps. Consequently, using the lazy scheme, we get a simple reduction from the dynamic to the static case: The update time (both amortized and worst-case) for maintaining a $(t+\epsilon)$-IS is smaller than the static time complexity of computing a t-IS by a factor of $\Omega((n/\beta)\cdot\epsilon)$, for any $t \geq 1$ and $\epsilon > 0$.

Remarks.

1. For planar graphs, one can compute a $(1+\epsilon)$-IS in time $O(n)$, for any constant $\epsilon > 0$ [7]. We can thus maintain a $(1+\epsilon)$-IS with a constant worst-case update time for any constant $\epsilon > 0$.
2. For graphs with average degree bounded by β, one can compute a $(k+1)/2$-IS in time $O(n^{3/2}\cdot\beta)$ [35]. We can thus maintain an IS with an approximation of roughly $(k+1)/2$ and a worst-case update time of $O(\sqrt{n}\cdot\beta^2)$.

4.3 Distributed networks

We consider the standard $\mathcal{LOCAL}$ and $\mathcal{CONGEST}$ models of communication (cf. [47]), which are standard distributed computing models capturing the essence of spatial locality and congestion. All processors wake up simultaneously, and computation proceeds in fault-free synchronous rounds during which every processor exchanges messages of either unbounded size (in the $\mathcal{LOCAL}$ model) or of $O(\log n)$-bit size (in the $\mathcal{CONGEST}$ model). It is easy to see that our sparsification algorithms can be implemented in distributed networks using a single communication round, even using $O(1)$-bit messages, during which each processor sends messages along at most Δ of its adjacent edges, where Δ is the degree bound of the sparsifier. Hence, the results mentioned in Section 1.4 hold w.r.t. both the $\mathcal{LOCAL}$ and the $\mathcal{CONGEST}$ models of communication. In this way we provide a clean and simple reduction from either bounded arboricity graphs (for the distributed approximate maximum matching and minimum VC problems) or from bounded average degree graphs (for the distributed approximate maximum IS problem) to bounded degree graphs.

For the distributed approximate VC problem, [8] showed that a $(2+\epsilon)$-VC can be computed in $O(\log\Delta/(\epsilon\log\log\Delta))$ rounds, where Δ is the maximum degree in the graph. We can plug our reduction to extend the result of [8] to graphs of arboricity bounded by α.

▶ **Theorem 15.** *For any graph of arboricity bounded by α and any $\epsilon > 0$, there is a distributed algorithm for computing a $(2+\epsilon)$-VC in $O(\log(\alpha/\epsilon)/(\epsilon\log\log(\alpha/\epsilon)))$ rounds.*

As mentioned in Section 1.4, for the distributed approximate maximum matching problem, a reduction from bounded arboricity graphs to bounded degree graphs was already given in [22]. The reduction of [22] starts by computing carefully chosen vertex sets in the graph, using which a bounded degree subgraph is computed, in $O(1)$ communication rounds. A distributed approximate matching algorithm is then run on that subgraph. Based on the matching returned as output to that algorithm, another carefully chosen bounded degree subgraph is computed, in $O(1)$ more communication rounds. A distributed approximate matching algorithm is then run on the new subgraph, and the final matching is the union of the first matching and the second. Our reduction is obtained as an immediate corollary of our matching sparsifier from Section 3.1, and it has several advantages over the one of [22]. First and foremost, our reduction is much simpler. Second, it requires a single communication round, whereas the number of rounds in the reduction of [22] depends on the time required to compute an approximate matching. In particular, throughout the computation of our sparsifier, the *load* (as well as node congestion) on all processors is low, since any vertex sends messages only along Δ of its adjacent edges is a single round, where Δ is the degree bound of the sparsifier. Moreover, the load on processors will remain low by definition throughout the subsequent run of any distributed approximate matching algorithm, since that algorithm is run on a bounded degree subgraph. In contrast, the computation of the subgraphs in [22] involves the exchange of messages between high degree vertices, and this "high-load" message exchange proceeds throughout $O(1)$ rounds; then a distributed approximate matching algorithm is run on a bounded degree subgraph, which may require many rounds to complete, and later another "high-load" message exchange proceeds throughout $O(1)$ more rounds. Third, the degree of our matching sparsifier is at most $O(\alpha/\epsilon)$, whereas the degree bound of the subgraphs of [22] is $O(\alpha/\epsilon^2)$, i.e., there is a gap of factor $1/\epsilon$, and this gap may be amplified significantly in applications where the runtime of the distributed algorithm depends exponentially on the maximum degree. In particular, [27] devised a distributed algorithm for computing a $(1+\epsilon)$-maximum matching in $\Delta^{O(1/\epsilon)} + O(\epsilon^{-2}) \cdot \log^* n$ rounds, for graphs with degree bounded by Δ. Due to our matching sparsifier, we extend the result of [27] to graphs of arboricity bounded by α to get a $(1+\epsilon)$-maximum matching in $(\alpha/\epsilon)^{O(1/\epsilon)} + O(\epsilon^{-2}) \cdot \log^* n$ rounds, which has a better dependence on ϵ than if the reduction of [22] were to be used.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. 57th FOCS*, pages 335–344, 2016.

2 Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 23rd SODA*, pages 1132–1139, 2012.

3 S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. H. M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. of 27th STOC*, pages 489–498, 1995.

4 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076, 2017.

5 B. Awerbuch, A. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. In *Proc. of 9th PODC*, pages 177–187, 1990.

6 B. Awerbuch, A. Baratz, and D. Peleg. Efficient broadcast and light-weight spanners. *Technical Report CS92-22, Weizmann Institute*, October, 1992.

7 Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

8 Reuven Bar-Yehuda, Keren Censor-Hillel, and Gregory Schwartzman. A distributed $(2+\epsilon)$-approximation for vertex cover in o($\log\delta/\epsilon$ log log δ) rounds. In *Proc. PODC*, pages 3–8, 2016.

9 Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The locality of distributed symmetry breaking. *J. ACM*, 63(3):20:1–20:45, 2016.

10 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proc. of 52nd FOCS*, pages 383–392, 2011.

11 András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in $\tilde{O}(n^2)$ time. In *Proc. 28th STOC*, pages 47–55, 1996.

12 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proc. 42nd ICALP*, pages 167–179, 2015.

13 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proc. of 26th SODA*, pages 692–711, 2016.

14 S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proc. 26th SODA*, pages 785–804, 2015.

15 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proc. 48th STOC*, pages 398–411, 2016.

16 Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *Proc. 55th FOCS*, pages 384–393, 2014.

17 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016.

18 Shiri Chechik. Compact routing schemes with improved stretch. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 33–41, 2013.

19 Eden Chlamtác and Michael Dinitz. Lowest degree k-spanner: Approximation and hardness. In *Proc. of APPROX/RANDOM*, pages 80–95, 2014.

20 Eden Chlamtac, Michael Dinitz, and Robert Krauthgamer. Everywhere-sparse spanners via dense subgraphs. In *PRof. 53rd FOCS*, pages 758–767, 2012.

21 Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Round compression for parallel matching algorithms. *CoRR*, abs/1707.03478, 2017.

22 Andrzej Czygrinow, Michal Hanckowiak, and Edyta Szymanska. Fast distributed approximation algorithm for the maximum matching problem in bounded arboricity graphs. In *PRoc. 20th ISAAC*, pages 668–678, 2009.

23 G. Das, P. J. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proc. of 9th SOCG*, pages 53–62, 1993.

24 G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *Int. J. Comput. Geometry Appl.*, 7(4):297–315, 1997.

25 Michael Elkin and Shay Solomon. Optimal euclidean spanners: Really short, thin, and lanky. *J. ACM*, 62(5):35:1–35:45, 2015.

26 Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Proc. 22th ESA*, pages 394–405, 2014.

27 Guy Even, Moti Medina, and Dana Ron. Distributed maximum matching in bounded degree graphs. In *Proc. ICDCN*, pages 18:1–18:10, 2015.

28 Manuela Fischer. Improved deterministic distributed matching via rounding. *CoRR*, abs/1703.00900v2, 2017.

29 J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002.

30 M. Gupta and R. Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proc. of 54th FOCS*, pages 548–557, 2013.

31 M. Gupta and A. Sharma. An $o(\log(n))$ fully dynamic algorithm for maximum matching in a tree. *CoRR*, abs/0901.2900, 2009.

32 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.*, 57(3):366–375, 1998.

33 P. Hall. On representatives of subsets. *J. London Math. Soc*, 10(1):26–30, 1935.

34 M. He, G. Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proc. 25th ISAAC*, pages 128–140, 2014.

35 Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6(3):243–254, 1983.

36 J. E. Hopcroft and R. M. Karp. An $\mathrm{n}^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

37 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

38 T. Kopelowitz, R. Krauthgamer, E. Porat, and S. Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proc. 41st ICALP*, pages 532–543, 2014.

39 Frank Thomson Leighton and Ankur Moitra. Extensions and limits to vertex sparsification. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 47–56, 2010.

40 Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994, 2017.

41 Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Proc. APPROX/RANDOM*, pages 260–273, 2013.

42 S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. of 21st FOCS*, pages 17–27, 1980.

43 Ankur Moitra. Approximation algorithms for multicommodity-type problems with guarantees independent of the graph size. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 3–12, 2009.

44 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. of 45th STOC*, pages 745–754, 2013.

45 K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proc. of 42nd STOC*, pages 457–464, 2010.

46 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proc. of 27th SODA*, pages 220–239, 2016.

47 D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

48 D. Peleg and S. Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Proc. of 27th SODA*, pages 712–729, 2016.

49 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

50 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Prof. of ICS*, pages 223–238, 2011.

51 S. Solomon. Fully dynamic maximal matching in constant update time. In *Proc. 57th FOCS*, pages 325–334, 2016.

52 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. 36th STOC*, pages 81–90, 2004.

53 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. of 13th SPAA*, pages 1–10, 2001.

54 V. V. Vazirani. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.