The Distributed Computing Column

by

Stefan Schmid

Aalborg University
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark

This distributed computing column features an interesting survey by Reut Levi and Moti Medina on the emerging field of centralized local algorithms, arising in the context of big graph analytics problems. The column also includes a report on a recent Dagstuhl seminar on another emerging field of interest to the TCS community: on the algorithmic foundations of programmable matter. Many thanks to the authors for their contribution and enjoy!

A (CENTRALIZED) LOCAL GUIDE

Reut Levi[1] and Moti Medina[1]

[1]Max Planck Institute for Informatics, Saarland Informatics Campus, Germany, {rlevi,mmedina}@mpi-inf.mpg.de

1 Introduction

Massive graphs, which contain millions and even billions of nodes, modeling the Internet, the World Wide Web, semantic networks and social networks, are being analyzed and processed constantly. As a result, classical models of computation, in which polynomial-time algorithms are considered efficient, may become inadequate. In the field of sublinear-time algorithms, the goal is to design extremely fast algorithms that probe only a minuscule portion of the input. Likewise, if the size of the output grows linearly in the size of the input, then it becomes desirable to provide access to parts of the output without having to store and compute it entirely. This is where *Local Computation Algorithms* (or the CENTLOCAL model), as defined by Rubinfeld et al. [57], comes into play.

In this survey we focus on CENTLOCAL algorithms for computational graph problems. A CENTLOCAL algorithm for a computational problem provides the user with *query* access to a global solution of the problem, while performing, per query, only a sublinear number of *probes* to the input. For example, in the maximal independent set problem (MIS) a query is “is vertex v part of an MIS?". The model requires that the answers to all queries must be consistent with a *single* solution regardless of the number of possible solutions. Usually, the challenge comes from this requirement. For instance, without the requirement for consistency, in the case where for each query *there is* a global solution for which the answer is “yes” (e.g., every vertex can be in some MIS), the CENTLOCAL algorithm can trivially answer “yes” to all the queries without performing any probe. Instead, for example in the MIS problem, the CENTLOCAL algorithm gives the “illusion” that it maintains a specific maximal independent set, that is, given a query v the algorithm answers whether v belongs to this specific MIS that it “has in mind” even though the CENTLOCAL algorithm cannot probe all of the input graph. For a formal definition of local algorithms in the context of graph problems, which is the focus of this survey, see Section 2.

There are many models of computation which are "local" such as: (1) distributed local computation by Linial [41], (2) property testing [12, 26], and (3) sublinear approximation algorithms (see [55] for a survey). Each of these models has some limitations on the way it accesses the input, and the way it produces the output, for example: (1) in the distributed local model each vertex is given a local input, and can explore its r-neighborhood via r rounds of synchronous communication with its neighbors. After these r rounds the vertex does a local computation and outputs its own part of the output. Note that each local output is consistent with a certain global output, and that the main resource here is the communication with other vertices. (2) An algorithm in the property testing model decides whether the input graph has a property or is far from having this property, according to some predefined distance measure. This testing algorithm can perform sublinear number of probes to the graph in order to reach its decision. (3) In the sublinear approximation model (see [55, 56] and the references within) an algorithm approximates the value of an optimal solution to a certain graph problem, e.g., what is the size of the optimal vertex cover of the input graph? Again, the algorithm, is allowed to perform sublinear number of probes in order to approximate the value of the optimal solution. We elaborate on studied connections of these models to the CentLocal model in Section 4. These connections also yield some useful tools for designing CentLocal algorithms.

Apart from the tools obtained by the connections to other local models, there are techniques which are frequently used for designing CentLocal algorithms, which we overview in Section 3. We then overview the current state-of-the-art algorithms for various graphs problems in terms of the already established tools (see Section 5). We do not cover all work in the field, moreover, there is a lot of research in related models which we do not cover at all in this survey, see, e.g., [2, 3, 11, 32, 58].

Access to the input graph of the CentLocal algorithm is given via probes to a graph oracle, for example, an algorithm asks the graph oracle "what are the neighbors of vertex v ?", and the oracle replies with a list of vertices. In all the algorithms surveyed here it is assumed that such an oracle is provided. This oracle can be to a fixed graph, or to a graph drawn from a certain distribution of random graphs. In Section 6 we consider the task of implementing an oracle that gives access to a graph which is drawn from the Barabási-Albert Preferential Attachment model [7] and the random recursive tree model [53]. In some cases these graphs are huge, calling for a *Local Graph Generator*. These generators are local in the sense that they generate the graph for each query on-the-fly, while giving the illusion of drawing the graph (completely) first, storing it in the oracle's memory, and then answering to these queries. Furthermore, not only that the graph is not stored in the local generator's memory, the number of bits that are stored by the generator are sublinear, for each query, as well as the number of consumed random bits, and the running time of the generator.

Notations. In the rest of this survey paper we are going to use the following notations. Let $G = (V, E)$ denote an undirected graph, and n denote the number of vertices in V. We denote the degree of v by $deg(v)$. Let Δ denote the maximum degree, i.e., $\Delta \triangleq \max_{v \in V}\{deg(v)\}$. The length of a path equals the number of edges along the path. For $u, v \in V$ let $dist(u, v)$ denote the length of the shortest path between u and v in the graph G. The ball of radius r centered at vertex v in the graph G is defined by $B_r(v) \triangleq \{u \in V \mid dist(v, u) \leq r\}$. For $k \in \mathbb{N}^+$ and $n > 0$, let $\log^{(k)}(n)$ denote the kth iterated logarithm of n, where $\log^{(0)}(n) \triangleq n$. For $n \geq 1$, define $\log^*(n) \triangleq \min\{i : \log^{(i)}(n) \leq 1\}$, that is, this function roughly counts how many times one can take a logarithm of n until the outcome is constant. We say that an event A occurs *with high probability (w.h.p)* if there exists a constant $c > 0$ such that $\Pr[A] \geq 1 - \frac{1}{n^c}$.

2 The CentLocal Model

The model of centralized local computations was defined by Rubinfeld et al. [57]. In this section we give the definition for the CentLocal model for problems over labeled graphs as it appears in Even et al. [17] (with slight variations) complemented by a discussion about query-order-oblivious, stateless, and stateful CentLocal algorithms. In the definition by Reingold and Vardi [54, Sec. 3] the running time of the CentLocal algorithm is also considered, and the space required for the state and for the bits of the random seed are counted separately.

Labeled graphs. An undirected graph $G = (V, E)$ is labeled if: (1) Vertex names are distinct and each have description of at most $O(\log n)$ bits. For simplicity, assume that the vertex names are in $\{1, \ldots, n\}$. We denote the vertex whose name is i by v_i. (2) Each vertex v holds a list of $deg(v)$ pointers, called *ports*, that point to the neighbors of v. The assignment of ports to neighbors is arbitrary and fixed.

Problems over labeled graphs. Let Π denote a computational problem over labeled graphs (e.g., maximum matching, maximal independent set, vertex coloring). A solution for problem Π over a labeled graph G is a function, the domain and range of which depend on Π and G. For example: (1) In the Maximal Matching problem, a solution is an indicator function $M : E \to \{0, 1\}$ of a maximal matching in G. (2) In the problem of coloring the vertices of a graph by $(\Delta + 1)$ colors, a solution is a coloring $c : V \to \{1, \ldots, \Delta + 1\}$. Let $sol(G, \Pi)$ denote the set of solutions of problem Π over the labeled graph G.

Probes. In the CentLocal model, access to the labeled graph is limited to probes. A *probe* is a pair (v, i) that asks "who is the ith neighbor of v?". The answer to a probe (v, i) is as follows. (1) If $deg(v) < i$, then the answer is "null". (2) If $deg(v) \geq i$, then the answer is the (ID of) vertex u that is pointed to by the ith port of v.

Online Property of CentLocal-algorithms. The input of an algorithm ALG for a problem Π over labeled graphs in the CentLocal model consists of three parts: (1) access to a labeled graph G via probes, (2) the number of vertices n and the maximum degree Δ of the graph G, and (3) a sequence $\{q_i\}_{i=1}^N$ of queries. Each query q_i is a request for an evaluation of $f(q_i)$ where $f \in sol(G, \Pi)$. Let y_i denote the output of ALG to query q_i. We view algorithm ALG as an online algorithm because it must output y_i without any knowledge of subsequent queries.

Consistency. We say that ALG is *consistent with* (G, Π) if

$$\exists f \in sol(G, \Pi) \text{ s.t. } \forall N \in \mathbb{N} \ \forall \{q_i\}_{i=1}^N \ \forall i \ : \ y_i = f(q_i) \,. \tag{1}$$

Examples. Consider the problem of computing a maximal independent set. The CentLocal-algorithm is input a sequence of queries $\{q_i\}_i$, each of which is a vertex. For each q_i, the algorithm outputs whether q_i is in I, for an arbitrary yet fixed maximal independent set $I \subseteq V$. Consistency means that I is fixed for all queries. The algorithm has to satisfy this specification even though it does not probe all of G, and obviously does not store the maximal independent set I. Moreover, a stateless algorithm does not even remember the answers it gave to previous queries. Note that if a vertex is queried twice, then the algorithm must return the same answer. If two queries are neighbors, then the algorithm may not answer that both are in the independent set. If the algorithms answers that q_i is not in the independent set, then there must exist a neighbor of q_i for which the algorithm would answer affirmatively. If all vertices are queried, then the answers constitute the maximal independent set I.

Resources and Performance Measures. The resources used by a CentLocal-algorithm are: probes, space, and random bits. The running time used to answer a query is not counted. The main performance measure is the *maximum number of probes* that the CentLocal-algorithm performs per query.

The *state* of algorithm ALG is the information that ALG saves between queries. The *space* of algorithm ALG is the maximum number of bits required to encode the state of ALG. A CentLocal-algorithm is *stateless* if the algorithm does not store any information between queries. In particular, a stateless algorithm does not store previous queries, answers to previous probes, or answers given to previous queries.

Definition 1. *An online algorithm is a* CentLocal $[q, s]$ *algorithm for* Π *if (1) it is consistent with* (G, Π)*, (2) it performs at most q probes, and (3) the space of the algorithm is bounded by s.*

The goal in designing algorithms in the CentLocal model is to minimize the number of probes and the space (in particular $q, s = o(n)$). A CentLocal $[q, s]$ algorithm with $s = 0$ is called a stateless CentLocal $[q]$ algorithm. Stateless algorithms are useful in the case of uncoordinated distributed servers that answer queries without communicating with each other.

Randomized local algorithms. A randomized CentLocal-algorithm is also parameterized by the *failure probability* δ. We say that alg is a CentLocal $[q, s, \delta]$ algorithm for Π if the algorithm is consistent, performs at most q probes, and uses space at most s with probability at least $1 - \delta$. The standard requirement is that $\delta = 1/\text{poly}(n)$.

The number of random bits used by a randomized algorithm is also a resource. One can distinguish between two types of random bits: (1) random bits that the algorithm must store between queries, that is, between the 1st and the 2nd and so on, and (2) random bits that are not stored between queries. We use the convention that information that is stored between queries is part of the state. Hence, random bits, that are chosen before the first query, are not included in the state, even if they are stored between queries, e.g., in [1, 45, 46, 54] the state does not change during the execution of the CentLocal algorithm. Thus, a randomized algorithm can be stateless although $s \neq 0$.

Query-Order-Oblivious and Stateless Algorithms. An important property of a CentLocal algorithm is that it is *query-order-oblivious* (QOO) [57, 1]. In particular, this property is useful if the CentLocal algorithm is used as a subroutine in Property Testing or in sublinear time approximation (see Sections 4.2 and 4.3). We say that a CentLocal algorithm is QOO if the (global) solution that the algorithm computes does not depend on the input sequence of queries. Being QOO and being stateless are equivalent. A simple observation [17] is that a stateless CentLocal algorithm is also QOO. The converse [27, Sec. 4.1, Obs. 1] is also true, that is, every QOO CentLocal algorithm can be simulated by a stateless CentLocal algorithm without incurring any overhead to the probe-complexity. The simulation is as follows: the stateless simulation invokes the QOO algorithm with its initial state and the same random seed for each query.

The Power of being Stateful. In the full version of [27, Sec. 5.3, Thm. 5] a variant of the Leader Election problem is considered. A stateful algorithm that uses $O(\log n)$ bits of state size is compared with *any* deterministic stateless CentLocal algorithm. It is shown that any stateless CentLocal algorithm requires $\Omega(n)$ times more probes than the considered stateful algorithm. This implies a separation between stateful and stateless CentLocal algorithms. In this survey, unless stated otherwise, the described CentLocal algorithms are stateless.

3 Techniques

In this section we survey some of the known techniques. We demonstrate the usage these techniques in the context of specific problems in Section 5, where we survey the state-of-the-art algorithms.

3.1 Localization of Local-Sequential Algorithms

In this section we overview a way of transforming sequential algorithm with certain properties to CentLocal algorithms [47, 1, 18, 17] which we call *local-sequential algorithms*. We start with an example and then give more precise explanation of the technique.

Consider the input graph in Figure 1. For now, disregard the orientation of the edges, the numbers within the circles, and the colors of the vertices. A local-sequential (greedy) algorithm for the mis problem is as follows: scan the vertices according to an arbitrary, but fixed, order and add vertex v to the mis if its neighbors that came before v in that order are *not* already in the mis and so on. Hence, answering the question whether v is in the mis in the context of sequential algorithms is quite simple: execute a greedy mis algorithm on the input graph and check if at some iteration v is in the mis and then answer YES, or if the algorithm already considering a vertex which is after v in the algorithm's scanning order, in this case the answer is NO.

In CentLocal algorithms we cannot afford a possible linear number of probes and, of course, cannot simply "save" a global ordering of the vertices (too costly in terms of space). We would like, some how, to have a global ordering of the vertices which all agree on *without* saving it or waste linear number of probes in order to compute it. The nice observation is that one can locally compute a "local" ordering that all the vertices of the graph can "fill in the details" to obtain a global ordering that is "good enough" and with small overheads. The idea is as follows. Let us assume that we can orient the edges in a CentLocal way so that the obtained directed graph is acyclic as in Figure 1. This orientation defines a partial order on the vertices, e.g., the vertex with rank 0.8 is "bigger" than the vertices with ranks 0.6, and 0.5 but no relation is defined to any other vertex. This orientation captures the notion of v came before u in the fixed but unknown global order if there is a directed edge (u, v). Now, the query of whether v is in the MIS is answered in the following way. The vertex v asks whether the neighbors which it points to are in the MIS. If none of these vertices are in the MIS then the answer to the query is YES otherwise the answer is NO (since at least one neighbor is in the MIS). This continues recursively until this “scanning” of the vertices reaches a sink vertex w.r.t. the acyclic orientation at hand. Similarly to the sequential algorithm the, as there is no vertex before the sink in the ordering, then this sink is in the MIS. Now all the answers to the recursive queries resolve, an eventually we answer the (initial) query by NO. In fact, the scanning that we have just made to the vertices is simply a DFS, where the decision of which vertex is in or out of the MIS are made in the “backtracking” phase of the DFS. Also note, that the number of inner queries (or probes) that the CentLocal algorithm made are equal to the number of vertices that are reachable from v, e.g., the upper right and the upper left vertices were not scanned since they are after v in the order. Hence, having a vertex with a reachability set which is linear in n would be very bad (for example consider an input graph of a path and the orientation is from “left” to “right”, the rightmost vertices have small probe complexity, while the leftmost ones have linear probe complexity). This calls for another problem that need to be solved beforehand: how do we locally orient the edges of the input graph so that the obtained orientation is both acyclic and with bounded reachability for *all* vertices? We call this problem *acyclic orientation with bounded reachability* and we deal with this question in Section 5.3 where both randomized and deterministic algorithms are discussed.

Hence, given a reachability set of size at most r, which is computed by a CentLocal $[p]$ algorithm for the OBR problem, then the total probe complexity of simulating a local-sequential greedy is simply $p \cdot r$. Plugging in the induced OBR deterministic, stateless CentLocal $\left[\Delta^{O(\sqrt{\Delta}\cdot\log^{2.5}\Delta)} \cdot \log^* n\right]$-algorithm by Fraigniaud, Heinrich, and Kosowski [23] (see Section 5.1, and Section 5.3) which obtains a reachability of $\Delta^{O(\Delta)}$ gives a CentLocal $\left[\Delta^{O(\Delta)} \cdot \log^* n\right]$-algorithm for any local-sequential algorithm.

The algorithms which this simulation holds are those that the decision (or output) of each vertex depends only on its neighbors that came before him in the ordering, for example, greedy $(\Delta + 1)$-coloring, greedy maximal matching, etc. For these kind of algorithms Even, Medina and Ron [18, 17] showed that any *linearization* of the partial order induced by the orientation (the role of the mentioned DFS above) gives the same outputs for each of the vertices. For a formal definition of the class of local-sequential algorithms see Even, Medina and Ron [18, 17]. For application of this framework in the context of online algorithms

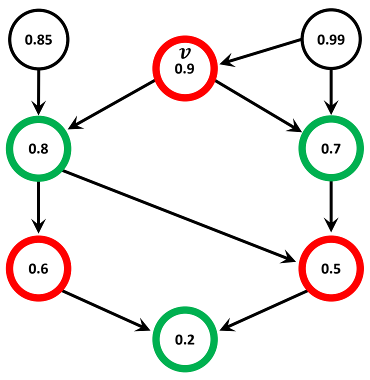

Figure 1: Localization of the greedy MIS algorithm. The orientation of the edges of the graph is acyclic. The query that is answered is: "Is vertex v is in the MIS?". Each vertex is depicted by a circle, the rank of which is in this circle. A vertex which is in the MIS is depicted by a green circle, and a circle which is not in the MIS is depicted by a red circle. A vertex that was not probed in order to answer the query is depicted by a black circle.

which are in the class of local-sequential, see Mansour et al. [45], Reingold and Vardi [54], and London et al. [42].

3.2 Shattering

Another way of reducing a problem of finding a solution on a large graph to finding independent solutions on small induced connected components is by shattering the graph. More precisely, in the *shattering phase*, a partial solution is found on a subset of the vertices (or edges) of the graph. Conditioning on this partial solution, finding the entire solution then reduces to finding solutions to each connected component independently.

In some sense, in the shattering phase we remove vertices from the graph (because the function on these vertices has already been determined). Usually the shattering phase proceeds in iterations. If in each iteration, each vertex is removed with constant probability, then after $\Omega(\log \Delta)$ independent iterations, each vertex remains in the graph with probability at most $\text{poly}(1/\Delta)$. Assuming that the events of removing vertices are mutually independent, this is usually suffices in order to shatter the graph into connected components of size $O(\log n)$. However, it is usually the case that these events are not mutually independent. Fortunately, usually the dependencies are confined only to small neighborhoods, in which case a more careful analysis, also referred to as Beck's approach [9], can show that the graphs is shattered in this case as well. For example, if the removal of a vertex depends only on its 2-hop neighborhood then the following lemmas (see also in [57, 1, 25]) are useful in order to prove that the graph is shattered by the union bound.

Lemma 1 ([40]). *There are at most $n(4\Delta^5)^s$ distinct trees T of size s embedded on the distance-5 graph.*

Lemma 2 ([40]). *Any connected sub-graph $H \subseteq G$ of size of at least sd^4 contains some set S of size s such that $dist_G(u, v)$ for every distinct $u, v \in S$.*

3.3 Partitioning

Providing access to a partition of the graph such that the number of edges in the cut of the partition is small is a natural method for designing local algorithms for approximation problems. The following definition is considered in the context of partition oracles [28, 15, 36].

Definition 2. *For $\varepsilon \in (0, 1], k \geq 1$ and a graph $G = (V, E)$, we say that a partition $\mathcal{P} = (V_1, \ldots, V_t)$ of V is an (ε, k)-partition (with respect to G), if the following conditions hold:*

1. *For every $1 \leq i \leq t$ it holds that $|V_i| \leq k$;*
2. *For every $1 \leq i \leq t$ the subgraph induced by V_i in G is connected;*
3. *The total number of edges whose endpoints are in different parts of the partition is at most $\varepsilon|V|$ (that is, $|\{\{v_i, v_j\} \in E : v_i \in V_j, v_j \in V_j, i \neq j\}| \leq \varepsilon|V|$).*

Given access to an (ε, k)-partition of the graph where k is a constant (w.r.t. $|V|$), designing local algorithms, in the bounded degree model, for a vast variety of optimization problems becomes easy. The solution will be approximate as the edges in the cut are usually ignored. Since the edges in the cut are ignored then on each of the remaining connected component a solution can be found independently. This gives constant query and time local algorithms even if a brute-force approach is applied.

An (ε, k)-partition, where k is a constant, does not necessarily exist for any bounded degree graph, e.g., when the graph is an expander. However, for some problems, families of partitions with more relaxed requirements can be useful. For examples, if one wants to provide local access to a spanner of a graph (see more in Section 5.7), then requirement 3 can be relaxed to requiring that the number of edges in $G/\mathcal{P}$ is at most $\varepsilon|E|$. where $G/\mathcal{P}$ is the graph on the vertex set $\{V_1, \ldots, V_t\}$ where $\{V_i, V_j\}$ is an edge, iff there exist $v_i \in V_i$ and $v_j \in V_j$ such that $\{v_i, v_j\} \in E$. Requirement 1 is then relaxed as to require that each induced subgraph has small diameter.

3.3.1 Partition via Contraction of Edges

Given a graph $G = (V, E)$, one can obtain a partition of V, such that the subgraph induced on each part is connected, by contraction of edges. The partition is then defined as follows: every pair of vertices u, v for which there is a path of contracted edges that connects u and v, are in the same part of the partition. In order to obtain parts which are not too big, contraction can be done in iterations, where in each iteration the edges contracted correspond to a matching (not necessarily a maximal one) between the subgraphs induced on each part of the partition we constructed so far, $\mathcal{P}$ (namely, a matching in $G/\mathcal{P}$). For more details see [28] and [36].

3.3.2 Partition via Centers

Another common way to obtain a partition is by selecting random centers, e.g., each vertex is selected as a center independently at random with probability $\varepsilon \in (0, 1)$. The partition is then defined by the Voronoi partition with respect to the selected centers. Namely, each vertex is in the same part as the center which is closest to it (where a rule for breaking ties should be defined). In the bounded degree model, each vertex can find its center, after exploring $O(1/\varepsilon)$ vertices, in expectation. To obtain a partition such that every vertex can find its center with probe complexity $\tilde{O}(1/\varepsilon)$, it was shown in [39] that if after exploring $\tilde{O}(1/\varepsilon)$ vertices, a center is not found, then the vertex can become a singleton in the partition. Namely, it was shown that the total number of parts in the partition remains $O(\varepsilon|V|)$, even after refining the partition by introducing singletons.

For many applications, it is not always sufficient to only find the center, but it is also necessary to find all the vertices in the respective part. If the size of the parts is too large, then one possible solution is to decompose the parts into smaller parts in a local manner (see e.g. [39]).

3.4 Amplification via Far Probes

Algorithms in the CentLocal-model can probe the graph (or the object at hand) in any location, even if the queried vertex is far a way. This is a major difference from distributed systems, where typically each processor can only communicate with its immediate neighbors (excluding the CONGESTED-CLIQUE model). A result by Göös et al. [27] shows that for a large family of natural problems this extra power is not significant (see Section 4.1.2).

However, the ability to probe the graph at random locations is useful for obtaining an estimate of a value. Specifically, it can be used for amplification in the following straight-forward way. Suppose that we have a randomized approximation CentLocal-algorithm that gives an approximate solution with probability 2/3. We can obtain an algorithm that gives an approximate solution with higher probability by adding to the algorithm a pre-processing phase. In the pre-processing phase, we pick random seeds and for each seed we estimate the size of the solution by sampling random vertices (or edges) and evaluating the function for the sample. For an appropriate sample size and number of trials, this pre-processing phase is likely to provide a random seed for which the obtained approximation ratio is as guaranteed. For an example of an application see the randomized algorithm for approximated Maximum Matching in Section 5.5.

3.5 Partial Exploration of Reachability Set

3.5.1 Simple Pruning of the Reachability Set

Given a CentLocal approximation algorithm where the expected probe complexity of each vertex is bounded by x, one can obtain a CentLocal-algorithm with query complexity which is proportional to x and with slightly worse approximation guarantee by simply applying Markov's inequality. Namely, the new algorithm simulates the original algorithm but whenever the number of queries goes over some predetermined threshold the algorithm stops the simulation and outputs a predetermined answer which does not violate the feasibility of the solution.

3.5.2 Efficient Exploration of the Reachability Set

The expected size of the reachability set of a local-sequential algorithms (see Section 3.1) for classical problems such as Maximal Independent Set and Maximum Matching was shown to be exponential in the maximum degree, Δ [47]. Fortunately, the size of the reachability set does not immediately determined the query complexity of the simulation. More specifically, it was conjectured by Nguyen and Onak [47] and was proved by Yoshida, Yamamoto, and Ito [61], that when the query-tree is explored in a specific order then the expected query complexity can be bounded by $O(\Delta^2)$ (where the expectation is over the vertices and the random orderings). Roughly speaking, the vertices with lower rank should be examined first by the simulation since the sequential algorithm examines these vertices first too. For this variant of simulation, the key lemma in [61] bounds the expected query complexity of a vertex of rank i. The Lemma shows that the bound only grows linearly with the rank.

3.6 Local Improvement

The *local improvement* technique was introduced by Nguyen and Onak [47] for the design of sublinear approximation algorithms. This technique is then applied in the context of sublinear approximation algorithms to vertex cover, maximum cardinality matching, maximum weighted matching, Set cover, etc.

In the context of CentLocal this technique was applied in maximum cardinality matching, maximum weighted matching (see Section 5.5).

The local improvement technique transforms an approximation algorithm which works in k phases into a CentLocal algorithm. More precisely, let us assume that the approximation algorithm works in *k phases*, where the input of each phase is the output of the previous one, e.g., the input for the first phase is simply the input to the algorithm and the output of the algorithm is the output of the last phase. Note that these phases are not necessarily the same phase which is executed over and over again. The transformation deals with implementing k CentLocal oracles for each of these phases, where the ith oracle gives a random access to the output of the ith phase. In turn, providing a random access to the output of the approximation algorithm is equivalent to answering a query to the kth CentLocal oracle. This query generates "inner" queries to the $(k-1)$th oracle and so on, moreover, each of these oracles also performs probes to the graph in order to compute the output of the corresponding phase, given the input from the previous one. For concrete applications of this technique see Section 5.5.

4 Connections to other Models

In the previous section we elaborated on known techniques that are used in the CentLocal model. In this section we present "connections" to other models, namely, given an algorithm in the CentLocal model how can we "translate" it to an algorithm in a different model? or the other way around? These connections are useful in two ways: (1) upper bounds: one can compile a "new" algorithm quickly, maybe in the first steps of one's research on a new problem, (2) lower bounds: translation from model A to model B can help carrying lower bounds from B to A.

4.1 Relation to the Distributed Local Model (DistLocal)

Unlike the "probe based" CentLocal model the classical DistLocal model [41, 52] is based on synchronous message passing, that is, each vertex communicates only with its neighbors in the graph in synchronized rounds, where in each round vertices send messages to their neighbors (not necessarily the same message to all of them), receive messages, and perform local computation. We say that a distributed algorithm is a DistLocal $[r]$-*algorithm* if the number of communication rounds is r. This implies that the information collected by the DistLocal algorithm after r rounds in each vertex v is simply $B_r(v)$. Compare this to the

CentLocal model where after t probes the CentLocal algorithm, which is queried on vertex v, knows $\{v\} \cup \bigcup_{i=1}^{t} N_1(p_i)$, where p_i is the ith probes made by the CentLocal algorithm. Hence, Having the ability to probe *anywhere* in the graph seems much more powerful than "just" collecting information from your neighbors. We discuss on this matter in more depth in Section 4.1.2. The CentLocal model is defined in Section 2. For a formal definition of the DistLocal model see [41, 52]. For a survey on DistLocal algorithms see [59]. We now outline the similarities and differences between the two models in Table 1.

Criterion	**CentLocal**	**DistLocal**
Randomness	Single source.	Each vertex has its own randomness.
IDs	Are known to the algorithms.	Are unknown.
Maximum degree Δ	Known.	Unknown .
Vertex labeling	Each vertex is labeled.	
Port numbering	$N_1(v)$ of each v are numbered from 1 to $\deg(v)$ in an arbitrary but fixed manner.	
Structure of the input G	Structure of the input graph $G = (V, E)$ is unknown.	
Accessing the input G	Centrally, via probes.	Each processor communicates with its neighbors in synchronous rounds: in each round, each processor sends messages, receives messages, and performs local computation.
Accessing the Output	Via queries.	After termination each processor knows its own part of the output.
Resources	State size, random Seed. computation is "for free" in both models.	
Complexity measures	# probes, state size, seed length.	# rounds.

Table 1: Comparing the DistLocal and the CentLocal models. Typically, the number of probes in the CentLocal model is $o(n)$, and the number of rounds in the DistLocal model is o(diameter).

4.1.1 Mutual Simulations between DistLocal and Stateless CentLocal

Simulating one model by the other model is useful both for designing algorithms (e.g., one favors a central model over a distributed one or vice versa) and for carrying lower bounds from one to the other. In this section we show how one can simulate algorithms over graphs in one model by algorithms in the other model.

In this section we focus on CentLocal algorithms that answer queries regarding vertices of a graph (this can be extended to edge queries by simulating the line-graph of the input graph). We say that a an algorithm in one model simulates an algorithm in the other model if the local outputs (either the output given by a query access or the local output of DistLocal algorithm) are the same.

Simulation of DistLocal by CentLocal (see [51]). The definition of the DistLocal $[r]$ model implies that each vertex v computes, after r rounds, a function of its $B_r(v)$. Hence, in order to simulate a DistLocal $[r]$- algorithm by a stateless CentLocal algorithm, the CentLocal algorithm can first "learn" the topology and IDs of $B_r(v)$ for the queried vertex v, and then simulate the DistLocal algorithm in "its head". Learning a ball $B_r(v)$ centered at vertex v takes at most $O(\Delta^r)$ probes, where Δ is a bound on the maximum degree of the graph. Note that in case the maximum degree of $B_r(v)$ is 2 (in case of a simple path or a cycle) then learning $B_r(v)$ takes only $2r$ probes. Reingold and Vardi [54] considered a class of graphs which they call Δ-*light graphs*. Roughly speaking, these graphs include bipartite graphs in which on one side the vertices have degree Δ and one the other side the distribution of the degree of the vertices is Δ on average and it is concentrated around the mean. For example, in the context of stable matching, a bipartite graph where one side corresponds to men and the other side corresponds to women and each man chooses Δ women uniformly at random is a Δ-light graph. For more details on the family Δ-light graphs see [54, Sec. 4]. Reingold and Vardi [54, Thm. 9.3] showed that simulating an r rounds DistLocal-algorithm, when the input graph is a Δ-light graph, results with $O(\Delta)^r \cdot \log n$ number of probes while using $O(\log n)$ space with high probability.

Simulation of CentLocal by DistLocal (see [47, 19], [17, Prop. 1]). Simulation of stateless CentLocal algorithms by DistLocal algorithms can be "tricky". Recall that a CentLocal algorithm can probe anywhere - what is the radius of the simulating DistLocal algorithm? In case the stateless CentLocal algorithm probes within a radius of r of any queried vertex then we can simulate it by, again, collecting *all* the topology and IDs in $B_r(v)$ in r rounds and then simulate the CentLocal algorithm locally in vertex v. Since the CentLocal algorithm at hand is stateless the simulation does not need any more information in order to simulate the CentLocal algorithm correctly. We discuss the case where the simulated CentLocal algorithm can probe anywhere in the graph in Section 4.1.2.

4.1.2 Simulating Far Probes in the DistLocal Model

In this section we deal with the question of whether we can simulate a stateless CentLocal algorithm by a deterministic DistLocal algorithm so that the number of rounds is sublinear in the diameter of the graph even if the CentLocal algorithm can have "far" probes.

Is there a General Simulation Argument? In general, the answer to the above question is: no! [27] For example, consider the following *binary consensus* problem: the local input for each vertex is in $\{0, 1\}$, the output of all the vertices is the same and should equal to the local input of at least one vertex. In the CentLocal model there is an algorithm that requires a *single* probe: for every query simply output the local input of vertex number 1. On the other hand, every DistLocal algorithm requires at least $\Omega(n)$ rounds: any algorithm that performs sublinear in n rounds errs on a input of a path that its left half has input 1 and the right half has input 0, that is, the rightmost vertex outputs 0 and the leftmost vertex outputs 1.

Nice Graph Problems. Due to the example of binary consensus [27] limit the class of problems that are considered for the simulation argument and define the following class of Nice graph problems. Roughly speaking, a graph problem is in Nice if (1) every solution to the problem implies that a relabeling of the vertices of the solution is also a solution for the problem at hand, and (2) every solution of an input G is also a solution to its maximal connected components (for a more rigorous definition see [27]. Note, that binary consensus is not in Nice. For example, Nice include: maximal independent sets, minimal dominating sets, minimal vertex covers, $(\Delta + 1)$-coloring of the vertices of the graph, maximal matchings, and edge colorings, etc. Göös et al. [27, Full ver. Sec. 4.7] also extended Nice to approximation algorithms for problems such as $(1 - \varepsilon)$-approximated maximum (weighted) matching.

An implicit simulation of CentLocal algorithms in the DistLocal model. Göös et al. [27, Thm. 2] showed that for every stateless CentLocal $[t(n)]$-algorithm that solves a problem in Nice where $t(n) = o(\sqrt{\log n})$, *there exists* a deterministic DistLocal $\left[t\left(\Theta\left(n^{\log n}\right)\right)\right]$-algorithm that simulates the CentLocal algorithm. The proof idea of this simulation argument is as follows. Introduce a virtual graph of $\Theta\left(n^{\log n}\right)$ vertices, which is known to all the vertices in the DistLocal algorithm. The simulation performs a random reshuffling of the IDs, currently the reshuffling is known to all vertices - this is *out* of the DistLocal model, since we cannot coordinate all the vertices instantaneously. We show momentarily how to lift this IDs reshuffling coordination assumption. Now, if a probe is made by the simulated CentLocal algorithm it will "land" w.h.p. on the virtual graph, which does not reveal new information to the simulating DistLocal algorithm. In turn, only the probes that induce a connected subgraph are "real" probes in the sense that they explore the input graph (rather than the virtual one). In order, to lift the reshuffling assumption, it is shown that there is a "good" reshuffling for all input bounded degree graphs. Due to this last stage the simulation we have just overviewed is existential, which suffices for carrying lower bounds - see Section 4.1.3. Göös et al. [27, Sec. 5, Thm. 3] also showed hot to obtain a *constructive* simulation which translates a CentLocal algorithm with far probes to a randomized CentLocal algorithm that its probes induce a connected subgraph (including the query). The constructive simulation has moderate overheads in the required seed length and in the number of probes.

4.1.3 Carrying Lower bounds from the DistLocal model to the CentLocal model

The implicit simulation argument implies that lower bounds for algorithms in the DistLocal model can be applied to the CentLocal model for deterministic and stateless algorithms that solve problems in Nice. For example, if there is a lower bounds in the DistLocal model of $\Omega(\log\log n)$ then the same (asymptotic) lower bounds holds for stateless CentLocal algorithms. Recall that QOO CentLocal algorithms can be simulated by a stateless CentLocal algorithms (see Section 2) without any overheads. Hence, lower bounds can be carried to QOO algorithms as well. As a result, the $\Omega(\log^* n)$ lower bound on maximal independent set, maximal matching, and $(\Delta+1)$-coloring by Linial [41] and the lower bounds on approximate maximum cardinality matching and maximum weighted matching of $\Omega(\log^* n)$ by Czygrinow et al. [14], and by Lenzen and Wattenhofer [34] can be applied on stateless (or QOO) CentLocal algorithm. Compare this with the state-of-the-art algorithms in Section 5.

4.2 Relation to Property Testing

In *Property Testing* typically the task is to distinguish between objects that have a certain property from objects which are far from having the property. Therefore it can be considered as a relaxation of a decision problem. Usually, as a result of this relaxation, the testers give an answer after inspecting only an extremely small portion of the object. Therefore, the testers answer whether the object have some global property by inspecting the object locally. Naturally, many techniques are common for the field of Property Testing and the CentLocal model. In the following we demonstrate how to borrow lower bounds from Property Testing to the CentLocal model (as appears in [37]).

A Reduction from testing Cycle-freeness with one-sided error. Goldreich and Ron showed a lower bound of $\Omega(\sqrt{n})$ for the problem of one-sided error testing of cycle-freeness in bounded-degree graphs [26, Proposition 3.4]. Their construction holds even if the input graph is promised to be connected. A one-sided tester must accept all cycle-free graphs and must reject with probability at least 2/3 every graph that is ε-far from being cycle free. In the bounded-degree model, the distance measure is with respect to the maximum possible number of edges in the graph. In particular, a graph is said to be ε-far from being cycle-free if it is required to remove at least εdn edges from the graph in order to remove all cycles.

Specifically, if the graph is connected then this implies that the graph contains at least $(n-1)+\varepsilon dn$ edges. An LSSG algorithm, $\mathcal{A}$, with sparsity parameter set to $\varepsilon d/2$, must return YES on at most $(1+\varepsilon d/2)n$ edges, with high probability. Therefore, given a graph which is ε-far from being cycle-free, $\mathcal{A}$ answers NO on a constant fraction of the edges. On the other hand, if the graph is cycle-free, $\mathcal{A}$ always returns YES on all the edges. Therefore, one can distinguish these two cases by making a constant number (depends only on ε) of queries to $\mathcal{A}$.

4.3 The Relation to Sub-linear approximation

A *sublinear approximation algorithm* for a certain graph parameter (e.g., the size of a minimum vertex cover) is given probe access to the input graph and outputs an approximation of this graph parameter with high constant probability. Many such algorithms obtain their estimation by invoking a "CentLocal- like" algorithm that answers queries for the problem that corresponds to the parameter. For example, if the approximated parameter is the size of the minimum vertex cover of the graph, then the corresponding problem is the Minimum Vertex Cover, and hence a query to this CentLocal-like algorithm is "does a given vertex belong to a fixed small vertex cover?" (see, e.g., [51, 47, 61, 49]). In turn, the sublinear approximation algorithm estimates the graph parameter by performing queries to this CentLocal-like algorithm. Usually the success probability of the CentLocal-like algorithms is constant, which is sufficient in the context of sub-linear approximation because it is possible to amplify the success probability. In order to obtain the desired $1-1/\text{poly}(|V|)$ success probability in the context of the CentLocal model, one can usually apply amplification as described in Section 3.4). For more details on this relation see [17, Sec. 1.4].

5 State-of-the-Art Algorithms

In this section we overview studied graph problems, such as: graph coloring, acyclic orientation, maximal independent set, maximal matching, approximate maximum (weighted) matching, vertex cover, local sparse spanning graphs, stable matching, and machine scheduling. For each problem we give the state-of-the-art results and a brief description of the algorithm and its analysis by using the above-mentioned techniques, and complement the discussion with a relevant lower bound, if one is known.

5.1 Graph Coloring Algorithms

We focus in this section on *vertex c-coloring* of a given graph $G = (V, E)$ (one extend the discussion to edge coloring by considering the line-graph of G). Essential, a c-coloring of G is a function $c : V \to [c]$, where $c \in \mathbb{N}$. A coloring is *legal* if for each edge $\{u, v\} \in E$ it holds that $c(u) \neq c(v)$.

In Sections 3.1 and 5.3 we described the connection between CentLocal coloring and simulating local-sequential algorithms, in a nutshell, the number of colors c affects the radius in which the CentLocal algorithm probes so that it can simulate the greedy algorithm, hence, improvement of the coloring algorithm might yield and improved simulation of sequential greedy algorithms.

In this section we consider c-graph coloring with $c = O(\Delta^2)$ colors and with $c = \Delta + 1$ colors which we call Δ^2-Color and $(\Delta + 1)$-Color, respectively.

CentLocal algorithms for Δ^2-Color. Applying the simulation of a DistLocal by a CentLocal algorithm (see Section 4.1.1) on the DistLocal $[O(\log^* n)]$-algorithm by Linial [41] gives a deterministic, stateless CentLocal $\left[\Delta^{O(\log^* n)}\right]$-algorithm for the Δ^2-Color. Even, Medina and Ron [18, 17] designed a deterministic, stateless CentLocal $\left[O(\Delta^4 \cdot \log^* n)\right]$-algorithm. Note, that the number of probes in this algorithm grows (slightly) slower as a function of n and is polynomial in Δ. How do one reduce the dependency in Δ a polynomial in Δ? evidently, in the Δ^2-Color problem one can reduce the dependency by partitioning the graph into edge-disjoint subgraphs of degree 2 [8], and then simulating the DistLocal $[O(\log^* n)]$-algorithm for Δ^2-Color [41] by applying the simulation of a DistLocal by a CentLocal algorithm (see Section 4.1.1) on each of these subgraphs. Since, the degree in each of this subgraphs is bounded by 2, there is no exponential blow-up in the probe complexity. Moreover, each subgraph is colored with constant number of colors. Now, each vertex has "many" colors, e.g., from each path that traverse the vertex. This palette of colors translates into a coloring with exponential number of colors. The exponential coloring can be reduced to Δ^2-coloring by applying color reduction techniques that also appear in [41], which require a constant number of rounds in the DistLocal model and hence require polynomial in the maximum degree probes in the CentLocal model. For more details about this algorithm and the partitioning technique which it employs, see [17, Sec. 3.1].

CentLocal algorithms for $(\Delta + 1)$-Color. Applying the Localization of Local-Sequential Algorithms technique (see Section 3.1) on greedy $(\Delta+1)$-coloring gives a deterministic, stateless CentLocal $\left[\Delta^{O(\Delta^2)} \cdot \log^* n\right]$-algorithm. Recently, Fraigniaud, Heinrich, and Kosowski [23] obtained an improved deterministic, stateless

CentLocal $\left[\Delta^{O(\sqrt{\Delta}\cdot\log^{2.5}\Delta)}\cdot\log^* n\right]$-algorithm as follows: given a Δ^2-coloring of the graph Fraigniaud et al. [23] designed an algorithm that requires $O\left(\sqrt{\Delta}\cdot\log^{2.5}\Delta\right)$ rounds in order to transform the Δ^2-coloring to a $(\Delta+1)$-coloring. Now, using the Δ^2-Color algorithm described above and applying the simulation of a DistLocal by a CentLocal algorithm (see Section 4.1.1) on their new color reduction technique, they obtained the improved probe complexity for the $(\Delta+1)$-Color problem. In fact, this new color reduction technique is more general and applies to *conflict-coloring*.

5.2 2-Coloring of Bipartite Graphs

Recently, Czumaj, Mansour and Vardi proposed the problem of locally coloring a bipartite graph in 2 colors [13]. They studied both the problem of exact coloring where violations are not allowed (namely, monochromatic edges are not allowed) and the relaxed version where only a small fraction of the edges are allowed to be monochromatic. They show a CentLocal algorithm for exact 2-coloring with query complexity $\tilde{O}(\sqrt{n})$ for graphs with good mixing time and a lower bound of $\Omega(n)$ for deterministic algorithms which are stateless. The upper bound proceeds as follows. An *anchor* vertex is selected, for which the color is determined. On query v, a pair of random walks from v and from the anchor, of length $\tilde{O}(\sqrt{n})$, are performs. With high probability, there is a collision between the random walks and hence the color of v can be determined by the length of the path between v and the anchor.

For the 2-dimensional grid they provide a deterministic algorithm whose probe complexity is $O(\sqrt{n})$ by taking a difference approach. They show that it is possible to traverse the grid in a straight line. This leads to a deterministic 2-coloring algorithm as follows. They traverse the grid in both directions from the queried vertex, v. Then they traverse the grid from an anchor vertex in one of the directions. Again, since these paths intersect, the color of v is determined by the length of the path between v and the anchor and the color of the anchor. The result is extended to k-dimensional grids where the probe complexity is $O(kn^{\frac{k-1}{k}})$.

In general graphs they obtain a tradeoff between the number of anchors and the size of the memory their algorithm uses. In this case they introduce a *pre-processing* step to the algorithm. In the pre-processing step they read the entire graph and determine the color of all the anchors which are selected uniformly at random. Then the color of each vertex can be determined by the length of a shortest path to the nearest anchor, which can be found, on expectation, after exploring $O(|V|/\alpha)$ vertices.

For trees, they take a similar approach but provide a difference trade-off. Their algorithm is allowed to err on a small number edges. Specifically they obtain the following trade-off: For an input graph $T = (V, E)$, which is a tree, there exists a randomized algorithm (with no preprocessing) which has probe complexity $O(n \log n/\alpha)$ and for which the number of monochromatic edges is at most $\alpha - 1$.

5.3 Acyclic Orientation with Bounded Reachability (OBR)

The *Acyclic Orientation with Bounded Reachability* problem (OBR) [18, 17] is as follows. An instance of the problem is an undirected graph $G = (V, E)$. A solution is a directed graph $H = (V, A)$ such that the underlying undirected graph of H is G. The objective function is to minimize the *reachability of H*, where the reachability of H is the maximum number of vertices which are reachable via a directed path from any vertex.

The results that we overview here are in terms of the size of the obtained reachability rather than it relation to the optimal one.

In the CentLocal context the goal is to design an algorithm that answers queries of the form "is the edge from u to v is outgoing?".

Solving the OBR problem serves as the first step in the technique of transforming sequential greedy algorithms to a CentLocal-algorithms (see Section 3.1).

Randomized Algorithms. The randomized ranking technique for OBR was first studied by Nguyen and Onak [47] and Onak [48]. The orientation in the ranking technique is computed as follows: (1) each vertex v is assigned a uniform number in the interval $[0, 1]$ which we denote by $r(v)$, (2) an edge is oriented from u to v if the rank of $r(u) > r(v)$. Nguyen and Onak used this orientation in the context of sublinear approximation algorithms and showed that the *expected* reachability under this randomized orientation is e^{Δ}/Δ (see [48, Lemma 2.3.1]). The ranking technique was also applied by Yoshida et al. [61] and Onak et al. [49] in the context of sublinear approximation algorithms to obtain better algorithms by resorting to *pruning* the probing of the reachability set (see Section 3.5). Mansour et al. [45] and Reingold and Vardi [54] studied the orientation problem in the context of CentLocal algorithms. Reingold and Vardi [54] showed that the reachability size is $2^{O(\Delta)} \cdot \log n$ w.h.p. while using $O(\log n)$ space (see also [42, Lemma 3]). In fact, Reingold and Vardi [54, Thm. 8.4] showed that this reachability guarantee holds for a wider class of graphs which they call Δ-*light graphs* (see Section 4.1.1 for more details).

Deterministic Algorithms. Even et al. [18, 17] designed and analyzed a stateless, and deterministic CentLocal algorithm for OBR. They observed that the OBR problem can be reduced to a coloring problem, as follows. Given a c-coloring of the vertices of the graph one can simply orient the edges from a higher color to a lower one (essentially, colors serve as ranks). The orientation induced by the c-coloring is acyclic (since edges are only directed downwards). Moreover, the length of the longest directed path is c. Hence, the obtained reachability size is $O(\Delta^c)$. Now, one can apply any of the CentLocal algorithms for c-coloring from Section 5.1 to obtain a CentLocal-algorithm for obr as follows: given an edge (u, v) query the CentLocal-algorithm for obr first query whether $\{u, v\} \in E$, and then queries the colors of each vertex. Plugging in the Δ^2-Color algorithm by [18, 17] (see also Section 5.1) yields a CentLocal$\left[O(\Delta^4 \cdot \log^* n)\right]$-algorithm for obr that obtains reachability of size $\Delta^{O(\Delta^2)}$. One can obtain an even better reachability by using the $(\Delta + 1)$-Color deterministic, stateless CentLocal$\left[\Delta^{O(\sqrt{\Delta}\cdot\log^{2.5}\Delta)} \cdot \log^* n\right]$-algorithm by Fraigniaud, Heinrich, and Kosowski [23] (this algorithm uses the Δ^2-Color algorithm by [18, 17] as a "black-box", see Section 5.1) and obtain a reachability of $\Delta^{O(\Delta)}$. Although, at first this look like a too high price to pay to get this reduced reachability, we will see in Section 5.4 that this lower reachability with higher probe-complexity is beneficial.

5.4 Maximal Independent Set (mis) and Maximal Matching (mm)

Given a graph $G = (V, E)$ a *maximal independent set* (mis) of G is a set of vertices such that each pair of vertices does not constitute an edge, i.e., they are independent, and that adding any vertex which is not in the mis violates the independence requirement, i.e., the set is maximal w.r.t. inclusion. In the CentLocal model, an algorithm answers a query of the form "is vertex v is *the* mis ?" (note that CentLocal algorithms gives answers to queries which are consistent which a specific mis). The goal in the *maximal matching* (mm) problem is to find a subset of edges of G such that the induced subgraph w.r.t. to these edges is of degree 1, i.e, it is a matching, and that adding any edge which is not in the matching violates the matching requirement, i.e., the matching is maximal w.r.t. set inclusion. The two problems are closely related as the mm problem is simply mis on the *line graph of* G.[1] In turn, a query for a CentLocal algorithm for the mm problem is "is edge e is an mm ?". We overview the state-of-the-art deterministic and randomized algorithms for mis in the following paragraphs.

Deterministic Algorithms. There is a local-sequential (greedy) algorithm for mis: scan the vertices according to an arbitrary order and add a vertex v to the mis if

[1]In the line graph of G each edge is mapped to a vertex, and edges connect vertices which correspond to adjacent edges in G.

none of its neighbors, that appear before v in this arbitrary order, are already in the MIS. Now, by applying the Localization of Local-Sequential Algorithms technique (see Sec. 3.1) with an underlying CENTLOCAL $[p]$ algorithm for the OBR problem that achieves reachability r, we obtain a stateless, deterministic CENTLOCAL $[p \cdot r]$-algorithm. Note that we are interested in minimizing the product $p \cdot r$. Plugging in the induced OBR deterministic, stateless CENTLOCAL $\left[\Delta^{O(\sqrt{\Delta} \cdot \log^{2.5} \Delta)} \cdot \log^* n\right]$-algorithm by Fraigniaud, Heinrich, and Kosowski [23] (see Section 5.1) which obtains a reachability of $\Delta^{O(\Delta)}$ gives a CENTLOCAL $\left[\Delta^{O(\Delta)} \cdot \log^* n\right]$-algorithm for MIS.

Randomized Algorithms. Ghaffari [25] provides a randomized algorithm for MIS. The algorithm proceeds in ℓ rounds. In each round, t, each vertex has a *desired level*, denoted by $p_t(v)$ to join the MIS. Initially the desire of all the vertices to join the MIS is the same. Specifically, $p_0(v) = 1/2$ for all $v \in V$. In each round t, each vertex is *marked* with probability $p_t(v)$. Every vertex which is marked and all its neighbors are un-marked, joins the MIS. All the vertices that join the MIS and their neighbors are removed from the graph. To proceed to the next iteration the desired level of all the vertices in the graph is updated as follows. The effective degree of a vertex v at round t, is defined to be the sum of the desire of its neighbors to get into the MIS, namely, $d_t(v) = \sum_{u \in N(v)} p_t(u)$. The desire of a vertex to get into the MIS is decreased by a factor of 2 if its effective degree is greater or equal to 2 and otherwise it is increased by a factor of 2 (unless it becomes greater than $1/2$, in which case it is set to $1/2$, which is the maximum level of desire). Intuitively, if the desire of the neighbors of a vertex, v, to join the MIS is low, then v increases its desire to get into the MIS and vice versa. This makes sense since if the desire of its neighbors to get into the MIS, namely, the effective degree of v, is low, then v has a good chance of getting into the MIS. One the other hand, if v has a high effective degree, but a constant fraction of its neighbors has a low effective degree the there is a good chance that one of the neighbors of v will get into the MIS, and hence, v will be removed. Therefore, the only problem is when v has a high effective degree but the same is true for most of its neighbors. The main Lemma in [25] shows that this situation can not happen for too many rounds. Intuitively, the reason is that if most of v's neighbors have high effective degree then v's effective degree is decreasing (because desired levels of most of its neighbors are decreasing). This leads to the following theorem.

Theorem 1 ([25]). *For each node v, the probability that v has not made its decision within the first $\Omega(\log d_v + \log 1/\varepsilon)$ rounds, is at most ε. This hold even if the coin tosses outside $N_2^+(v)$ are determined adversarially.*

Building on this, with the shattering technique (see Section 3.2), the algorithm in [25] obtains an improved time and space complexity of $2^{O(\log^2 \Delta)} \log^3 n$ and $2^{O(\log^2 \Delta)} \log^2 n$, compared to $2^{O(\log^3 \Delta)} \log^3 n$ and $2^{O(\log^3 \Delta)} \log^2 n$ of [40].

5.5 Approximate Maximum (Weighted) Matching (MCM, MWM)

Given a graph $G = (V, E)$ a *maximum cardinality matching* (MCM) is a subset of the edges of G such that each vertex in the graph is incident to at most one edge, i.e., a matching, which has the highest number of edges among the possible matchings in G. An α-approximate MCM where $0 < \alpha < 1$, is a matching which its cardinality is at least α fraction of the MCM. In the *maximum weighted matching* (MWM) problem each edge is assigned a positive weight. Now, the goal in MWM is to output matching with the highest total weight, and analogously in the α-approximate MWM the goal is to output a weighted matching which its weight is at least an α fraction from the MWM. As in the MM problem a query for a CentLocal algorithm for the MCM or MWM problems is "Does edge e belong to the matching?". In this section we overview the state-of-the-art deterministic and randomized algorithms for these problems.

Deterministic Algorithms. A $(1-\varepsilon)$-approximate MCM CentLocal$\left[(\log^* n)^{O(1/\varepsilon)} \cdot 2^{O(\Delta^{1/\varepsilon})}\right]$-algorithm is presented in [18, 17, Thm, 19]. Similarly to previous work (both in the DistLocal and CentLocal models) [47, 43, 46], this algorithm for MCM is based on the *augmenting paths framework* of Hopcroft and Karp [30]. The augmenting path framework proceeds in iterations, where in each iteration it constructs a new matching based on the matching from the previous iteration (starting from the empty matching). Each iteration computes an MIS over a subset of paths with special properties (which are a function of the matching from the previous iteration) called *augmenting paths*. A CentLocal algorithm is obtained by applying the *local improvement* technique by Nguyen and Onak [47] (see Section 3.6) together with a deterministic CentLocal algorithm for MIS from Section 5.4. Roughly speaking, in each iteration of the local improvement technique a CentLocal oracle is designed for the corresponding matching, where a query oracle for this matching is given. Then, simulating the set of augmenting paths can also be done locally followed by an application of a deterministic MIS on these paths, and updating the matching to the current iteration's matching. An $(1-\varepsilon)$-approximate MWM CentLocal$\left[(\log^* n)^{O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})} \cdot \left((w_{\min}(\varepsilon))^{-1}\right)^{O(\Delta^{1/\varepsilon})}\right]$-algorithm is presented in [17, Thm. 24], where $w_{\min}(\varepsilon)$ denote the smallest weight which is bigger than ε/n. Similarly to the $(1-\varepsilon)$-approximate MCM the MWM algorithm is obtained by applying the local improvement technique by Nguyen and Onak [47] (see Section 3.6) together with a deterministic CentLocal algorithm for (a variant of) MIS. The algorithm which this CentLocal algorithm builds on is the parallel approximation algorithm of Hougardy and Vinkemeir [31].

Fischer and Ghaffari [22] presented a deterministic

CentLocal $\left[2^{O(\log^3 \Delta)} \cdot \log^* n\right]$ algorithm for $\Theta(1)$-approximate mcm.[2] The idea is as follows. First, transform the graph to a bipartite graph by (1) considering an arbitrary orientation of the edges,(2) have two copies for each vertex which correspond to out-edges and in-edges, (3) map each edge of the original graph to an edge between the corresponding "out to in" vertices. The obtained graph is bipartite. Now, they compute a fractional approximate mcm and round it to obtain an integral approximate mcm in $O(\log^2 \Delta)$ rounds in the DistLocal model over the bipartite graph. Going back to the input graph "corrupts" the matching at hand and results with an induced subgraph of degree 2. On this they compute a mis in $O(\log^* \Delta)$ rounds [50] this incurs a factor of $\frac{1}{3}$ to the approximation. Moreover, this mis algorithm requires a Δ^2-Color coloring of the graph. Hence, the total number of probes follows from the Δ^2-Color algorithm from Sec 5.1 and by simulating the DistLocal algorithm of $O(\log^2 \Delta)$ by a CentLocal algorithm (see Section 4.1.1).

Mansour, Patt-Shamir, and Vardi [44, Sec. 5] showed how to reduce the problem of computing an approximate mwm to computing an approximate mcm as follows: given a CentLocal $[t(n)]$-algorithm for α-approximate mcm, then there is a CentLocal $[\Delta \cdot t(n)]$-algorithm for $\frac{\alpha}{8}$-approximate mwm. The idea is to partition the edges of the graph into disjoint parts of edges, which have the same weight up to a multiplicative constant factor, and to compute an α-mcm on each of them. Now, an edge is in the output weighted matching if it is in the matching of its part, and if its weight is the maximum among the adjacent edges which are in the matchings of the other parts.

Randomized Algorithms. Yoshida, Yamamoto, and Ito [61] studied the $(1-\epsilon)-$ mcm problem in the context of sublinear approximation algorithms. Building on their result, Levi, Rubinfeld and Yodpinyanee [40] provide a $(1-\varepsilon)$-approximation CentLocal algorithm for mcm. The complexities of their algorithm are polynomial in Δ, exponential in $1/\varepsilon$ and poly-logarithmic in n. To obtain their algorithm they use amplification as described in Section 3.4.

5.6 Minimum Vertex Cover (vc)

In the *Minimum Vertex Cover* problem (vc) the goal is to compute a minimum cardinality subset of the vertices such that each edge is incident to a vertex in the cover. Analogously, in the weighted variant, there is a weight function that assigns nonnegative weights for each vertex. Now, the goal is to compute a minimum

[2]The actual result of [22, Thm. 1.1] is a $(2+\varepsilon)$-approximate mcm. To obtain this stronger approximation, an additional factor of $O(\log \frac{1}{\varepsilon})$ is introduced to the number of rounds of the DistLocal model.

weight subset of the vertices such that it covers the edges. A query to a CentLocal algorithm for vc is "is vertex v is in a vc?".

Vertex cover approximation algorithms in the DistLocal model are abundant. For a summary of results see [6, 4]. Current state-of-the-are for $(2+\varepsilon)$-approximation weighted vc in the DistLocal model is by Bar-Yehuda, Censor-Hillel, and Schwartzman [6] which obtained a deterministic DistLocal algorithm with $O\left(\frac{\log\Delta}{\varepsilon\log\log\Delta}\right)$ rounds. In turn, simulating this algorithm in the CentLocal model (see Sec. 4.1.1) results with a probe complexity which is superpolynomial in the maximum degree Δ.

Feige, Mansour, and Schapire [20, Sec. 6] showed that in the case where the input graph is bipartite (where each vertex knows in which "side" it is) a $(1+\varepsilon)$-approximation vc can be computed in the CentLocal model by reducing it to the $(1-\varepsilon)$-mcm problem. The reduction is randomized and the obtained approximation is in expectation. More formally, given an $(1-\varepsilon)$-mcm there is a CentLocal$\left[O\left(\Delta^{1/\varepsilon}\right)\right]$ randomized algorithm. Given a query v the vertex cover CentLocal algorithm probes the $B_{1/\varepsilon}(v)$, and invokes the $(1-\varepsilon)$-mcm CentLocal algorithm on each of the edges within this ball. Actually, the radius of the probed ball is a random variable that attains odd values in $[1, 1/\varepsilon]$. Feige, Mansour, and Schapire [20, App. D] also obtained the same approximation in he weighted setting.

5.6.1 Lower Bounds

Trevisan proved that any algorithm that outputs, with constant probability, a $(2-\gamma,\varepsilon)$-estimate of the size of the Minimum Vertex Cover, for constant γ and ε, requires $\Omega(\sqrt{n})$ probes to the graph (see proof sketch in [51]). Parnas and Ron showed that for approximation with multiplicative factor α and additive factor (constant) ε the query complexity of the problem grows at least linearly with respect to the average degree as long as it is bounded by $O(n/\alpha)$. These lower bounds can be adapted to show corresponding lower bounds in the CentLocal model.

Recently, Feige, Patt-Shamir and Vardi [21] showed the following lower bound for a variety of approximation ratios. Their lower bound applies even with respect to *strong probes*, namely, when probing a vertex v, the list of all the neighbors of v is returned.

Theorem 2 ([21]). *For any $\varepsilon < 1/2$, any randomized* CentLocal*-algorithm that computes a vertex cover whose size is a $(\frac{1}{2}n^{1-2\varepsilon})$-approximation to the size of the minimum vertex cover requires at least $\varepsilon n^{\varepsilon}$ strong probes.*

Roughly speaking, to prove their lower bound they construct a family of graphs in which for a large subset of vertices, all the vertices in the subset have a similar view, and therefore are indistinguishable, but only a single vertex in the subset needs to be added to the vertex cover. They obtain this property by identifying vertices with low degree polynomials, which ensure that vertices do not share too many neighbors, as the intersection of the neighbors corresponds to the roots of a low degree polynomial.

5.7 Local Sparse Spanning (Sub-)Graphs (LSSG)

When dealing with large graphs, it is often important to work with a sparse subgraph that maintains essential properties of the original input graph. In a series of papers [38, 35, 39, 33], the problem of locally constructing (ultra-)sparse subgraph while maintaining connectivity was considered (see also survey by Rubinfeld [56]). Namely, given an edge of the original graph, the algorithm quickly determines whether this edge belongs to the sparse spanning subgraph without computing the entire sparsification.

Definition 3. *An algorithm $\mathcal{A}$ is a* Local Sparse Spanning Graph (LSSG) algorithm *if, given $n, d \geq 1$, a parameter $\epsilon \geq 0$, and query access to the incidence-lists representation of a connected graph $G = (V, E)$ over n vertices and degree at most d, it provides oracle access to a subgraph $G' = (V, E')$ of G such that:*

1. *G' is connected.*
2. *$|E'| \leq (1 + \epsilon) \cdot n$ with probability at least $1 - 1/\Omega(n)$, where E' is determined by G and the internal randomness of $\mathcal{A}$.*

By "providing oracle access to G'" we mean that on input $(u, v) \in E$, $\mathcal{A}$ returns whether $(u, v) \in E'$, and for any sequence of edges, $\mathcal{A}$ answers consistently with the same G'.

An algorithm $\mathcal{A}$ is an LSSG algorithm for a family of graphs *C if the above conditions hold, provided that the input graph G belongs to C.*

Note that Item 2 implies that the answers of an LSSG algorithm to queries cannot depend on previously asked queries.

Levi et al. [38] show a lower bound of $\Omega(\sqrt{|V|})$ probes per user query for this problem. With respect to the upper bound they made progress for specific families of graphs, but determining the complexity of the problem in general remains open. For families of graphs with very high expansion, they provide an almost tight upper bound [38]. For families of graphs which are, roughly speaking, non-expanding everywhere, and in particular for hyper-finite graphs, Levi et al. [35] provide an upper bound with probe complexity which is independent of $|V|$. On the negative side, they provide a construction of a family of graphs with relatively weak expansion properties in which the complexity of the problem grows with

n. For the family of graphs with an excluded fixed minor (which in particular includes planar graphs), [39] provide an algorithm whose probe and time complexities are polynomial in $1/\epsilon$ and in d. Their algorithm is also useful in the distributed message-passing setting, in fact, it can be easily adapted to yield an efficient algorithm in this setting as well. Moreover, subgraph that their algorithm provides access to is also a good spanner, i.e., distances are approximately preserved. Both algorithms in [38] and [39] uses random centers (see Section 3.3.2) as an initial step to partition the graph.

Recently, Lenzen and Levi [33] provided an LSSG algorithm for general bounded degree graphs with sublinear probe complexity.

Theorem 3 ([33])**.** *There exists an* LSSG *algorithm such that for any graph G over n vertices of maximum degree at most Δ and $\varepsilon > 0$, its query complexity, space complexity (length of the random seed), and running time are* $\tilde{O}(n^{2/3}) \cdot \mathrm{poly}(1/\varepsilon, \Delta)$.

Moreover, the algorithm has the additional property that it outputs a spanner. Specifically, with high probability, for each deleted edge there is a path of $O(\log n \cdot (\Delta + \log n)/\varepsilon)$ hops in the spanning subgraph that connects its endpoints.

Loosely speaking, the idea in [33] is to run two different algorithms for expanding parts and for non-expanding parts of the graph. For the non-expanding parts, one can borrow techniques from distributed algorithms, since in this case it is possible to gather all the information on the local neighborhood (specifically, the $O(\log n)$ hop neighborhood) of every vertex. the main focus is thus in designing an algorithm for the expanding parts. The algorithm in [33] is quite different from the algorithm in [38], as the expansion properties are much weaker than required by the latter algorithm.

In [38] there is also an algorithm for the weighted case, namely, in the case where the spanning subgraph the algorithm outputs also has low weight. The algorithm is designed for graphs with an excluded fixed minor and proceeds by contracting edges (see Section 3.3.1).

5.8 Approximation to the Maximum (Weight) Spanning Tree

Mansour et al. [44] study the problem of approximated Maximum Spanning Tree. They provide the following result.

Theorem 4 ([44])**.** *There exists a deterministic* CentLocal *algorithm, that for any graph G whose degree is bounded by Δ and every $\varepsilon > 0$, computes a forest whose weight is a $(1 - \varepsilon)$-approximation to the maximum spanning tree of G in time complexity $\Delta^{O(1/\varepsilon)}/\varepsilon$ and probe complexity $\Delta^{O(1/\varepsilon)}$.*

Their algorithm proceeds in $1/\varepsilon$ iterations, where in each iteration all the vertices, in parallel, add another edge to their growing component. Specifically, the heaviest edge in the respective edge cut is added. An Adaptation of this parallel algorithm to the CentLocal model gives the complexities as stated in Theorem 4.

5.9 CentLocal Mechanism Design

In the context of CentLocal mechanism design we consider two problems: (1) *stable matching*, and (2) *machine scheduling*.

In the stable matching problem there are two sets of agents, men and women, each of cardinality n. Each man and woman have an ordered preference list of women and men which they would like to be matched. The lists are ordered from the most preferred match to the least preferred match. The goal is to compute a matching such that there is no man or woman that prefer each other over their partners in the computed matching, e.g., the matching is stable. Hassidim, Mansour, and Vardi [29, Sec. 3] considered the case in which the lists of the men are bounded by Δ, and that each woman is picked uniformly at random. Hence, the input graph is a Δ-light graph. For this case they simulate the Gale-Shapley algorithm [24] for $2\Delta/\varepsilon$ rounds and obtained a randomized CentLocal-algorithm that performs $O(\Delta)^{2\Delta/\varepsilon} \cdot \log n$ probes (see Section 4.1.1 for simulation of DistLocal by CentLocal algorithms where the input graph a is Δ-light graph) and uses a seed of size $O(\log n)$ w.h.p. Moreover, the obtained matching has at most $2n/\Delta + \varepsilon n$ unmatched men.

In the machine scheduling problem there are n uniform jobs, and m machines. Each machine has a positive integral constant (i.e., does not depend on n or m) *speed*. The *load* of a machine is the number of jobs that are allocated to the machine over its speed. The *utility* of a machine is the payment it received from the mechanism minus its load. The machines are the players and their private information is their speeds. Each machine declares its *bid*, which is supposedly its speed. The first part of the *mechanism design* is to compute a payment scheme that encourages the players to be truthful about their speeds. These "honest" bids enable the mechanism to compute an allocation that minimizes the maximum load, which is its goal. Hassidim, Mansour, and Vardi [29, Sec. 5] considered the case where $m = \Theta(n)$ and in which each job selects uniformly at random (w.r.t. the bids) Δ machines and commit to one of them. Hassidim, Mansour, and Vardi [29] considered two variants of the machine scheduling problem where in each of them the set of allowable machines for each job is defined differently. For each case they simulated an online algorithm (e.g., [5, 10, 60]) to obtain a CentLocal allocation algorithm (see Section 3.1 for simulation of (online) Local-Sequential Algorithms in the CentLocal model) and obtained a probe complexity of $2^{O(\Delta)} \cdot \log n$, and a seed length of $O(\log n)$ w.h.p. They also showed how to implement a CentLocal algorithm for the payment scheme in [29, Lemma 5.6, Lemma 5.18]. In the two studied variants of the problem, Hassidim, Mansour, and Vardi [29, Thm. 5.10, Thm. 5.20] obtained a truthful mechanism which is a $O(\log \log n)$-approximation.

Note that in both problems the underlying graph is a bipartite graph $G = (L \cup R, E)$. On one side there are selfish agents L each of which picks uniformly at random Δ items (or other set of selfish agents) from the other side R. In turn, the degrees of the vertices in R are distributed binomially with expected degree of Δ. This input graph is in the family of graphs called Δ-light graphs [54], and admit fast simulation of local-sequential algorithms as well as DistLocal algorithms in the CentLocal model (see Sections 3.1, 4.1.1).

6 Local Generator for Evolving Graphs

Random-graph generators are ubiquitous and their importance for simulations and extrapolations is well established. For many applications we might want to access just a small portion of an enormous random graph, in which case it would be preferable to avoid constructing it entirely. In the standard approach, the whole graph is chosen randomly according to the distribution of the model before answering user queries to the adjacency lists of the graph. In a recent work, Even et al. [16] proposed to answer queries by generating the graphs on-the-fly while respecting the probability space of the random graph model. They call this the problem of *local generation* of random graphs. Their focus is on evolving random graph models and in particular they considered the Barabási-Albert Preferential Attachment model (BA-graphs) [7] and the random recursive tree (RR-trees) model [53]. They provide a randomized graph generator, that generates answers to adjacency list queries of BA-graphs (and RR-trees) in time complexity which is only poly-logarithmic in the size of the graph. Such a generator outputs answers to adjacency list queries as if it first picked the graph at random (according to the respective distribution) and only then answers the queries.

Many evolving random graph models have RR-trees as an underlying structure, therefore, it could be interesting to extend the results of [16] for a large family of evolving graphs. As an even broader objective, one can consider designing random access generators for other Markov-Chain and stochastic processes.

References

[1] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *SODA*, pages 1132–1139, 2012.

[2] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, V. Mirrokni, and S.-H. Teng. Local computation of pagerank contributions. *Internet Mathematics*, 5(1-2):23–45, 2008.

[3] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 475–486. IEEE, 2006.

[4] M. Åstrand and J. Suomela. Fast distributed approximation algorithms for vertex cover and set cover in anonymous networks. In *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece, June 13-15, 2010*, pages 294–302, 2010.

[5] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM journal on computing*, 29(1):180–200, 1999.

[6] R. Bar-Yehuda, K. Censor-Hillel, and G. Schwartzman. A distributed $(2+\epsilon)$-approximation for vertex cover in o(logδ/ϵ log log δ) rounds. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 3–8, 2016.

[7] A. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[8] L. Barenboim, M. Elkin, and F. Kuhn. Distributed $(\Delta+1)$-coloring in linear (in Δ) time. *SIAM Journal on Computing*, 43(1):72–95, 2014.

[9] J. Beck. An algorithmic approach to the lovász local lemma. I. *Random Struct. Algorithms*, 2(4):343–366, 1991.

[10] P. Berenbrink, A. Brinkmann, T. Friedetzky, and L. Nagel. Balls into non-uniform bins. *Journal of Parallel and Distributed Computing*, 74(2):2065–2076, 2014.

[11] P. Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[12] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[13] A. Czumaj, Y. Mansour, and S. Vardi. Sublinear graph augmentation for fast query implementation. Technical report, 2017.

[14] A. Czygrinow, M. Hańćkowiak, and W. Wawrzyniak. Fast distributed approximations in planar graphs. In *Distributed Computing*, pages 78–92. Springer, 2008.

[15] A. Edelman, A. Hassidim, H. N. Nguyen, and K. Onak. An efficient partitioning oracle for bounded-treewidth graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 530–541, 2011.

[16] G. Even, R. Levi, M. Medina, and A. Rosén. Sublinear random access generators for preferential attachment graphs. *CoRR*, abs/1602.06159, 2016.

[17] G. Even, M. Medina, and D. Ron. Best of two local models: Local centralized and local distributed algorithms. *CoRR*, abs/1402.3796, 2014.

[18] G. Even, M. Medina, and D. Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 394–405, 2014.

[19] G. Even, M. Medina, and D. Ron. Distributed maximum matching in bounded degree graphs. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 18:1–18:10, 2015.

[20] U. Feige, Y. Mansour, and R. E. Schapire. Learning and inference in the presence of corrupted inputs. In *Proceedings of The 28th Conference on Learning Theory, COLT 2015*, pages 637–657, 2015.

[21] U. Feige, B. Patt-Shamir, and S. Vardi. On the probe complexity of local computation algorithms. *arXiv preprint arXiv:1703.07734*, 2017.

[22] M. Fischer and M. Ghaffari. Deterministic distributed matching: Simpler, faster, better. *arXiv preprint arXiv:1703.00900*, 2017.

[23] P. Fraigniaud, M. Heinrich, and A. Kosowski. Local conflict coloring. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 625–634. IEEE, 2016.

[24] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[25] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277, 2016.

[26] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[27] M. Göös, J. Hirvonen, R. Levi, M. Medina, and J. Suomela. Non-local probes do not help with many graph problems. In *International Symposium on Distributed Computing*, pages 201–214. Springer Berlin Heidelberg, 2016. Full version is available on arXiv.

[28] A. Hassidim, J. A. Kelner, H. N. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 22–31, 2009.

[29] A. Hassidim, Y. Mansour, and S. Vardi. Local computation mechanism design. *ACM Trans. Economics and Comput.*, 4(4):21, 2016.

[30] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[31] S. Hougardy and D. E. Vinkemeier. Approximating weighted matchings in parallel. *Inf. Process. Lett.*, 99(3):119–123, 2006.

[32] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003.

[33] C. Lenzen and R. Levi. A local algorithm for the sparse spanning graph problem. *CoRR*, abs/1703.05418, 2017.

[34] C. Lenzen and R. Wattenhofer. Leveraging Linial's locality limit. In *Distributed Computing*, pages 394–407. Springer, 2008.

[35] R. Levi, G. Moshkovitz, D. Ron, R. Rubinfeld, and A. Shapira. Constructing near spanning trees with few local inspections. *Random Structures & Algorithms*, pages n/a–n/a, 2016.

[36] R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Trans. Algorithms*, 11(3):24:1–24:13, 2015.

[37] R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. *CoRR*, abs/1402.3609, 2014.

[38] R. Levi, D. Ron, and R. Rubinfeld. Local algorithms for sparse spanning graphs. In *Proc. APPROX/RANDOM*, pages 826–842. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014.

[39] R. Levi, D. Ron, and R. Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 38:1–38:15, 2016.

[40] R. Levi, R. Rubinfeld, and A. Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, pages 1–24, 2016.

[41] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[42] P. London, N. Chen, S. Vardi, and A. Wierman. Distributed optimization via local computation algorithms. 2017.

[43] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. *J. ACM*, 62(5):38:1–38:17, 2015.

[44] Y. Mansour, B. Patt-Shamir, and S. Vardi. Constant-time local computation algorithms. In *Approximation and Online Algorithms - 13th International Workshop, WAOA 2015, Patras, Greece, September 17-18, 2015. Revised Selected Papers*, pages 110–121, 2015.

[45] Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages, and Programming*, pages 653–664. Springer, 2012.

[46] Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *APPROX-RANDOM*, pages 260–273, 2013.

[47] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 327–336. IEEE, 2008.

[48] K. Onak. New sublinear methods in the struggle against classical problems, September 2010.

[49] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1123–1131, 2012.

[50] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

[51] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1):183–196, 2007.

[52] D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[53] B. Pittel. Note on the heights of random recursive trees and random m-ary search trees. *Random Struct. Algorithms*, 5(2):337–348, 1994.

[54] O. Reingold and S. Vardi. New techniques and tighter bounds for local computation algorithms. *J. Comput. Syst. Sci.*, 82(7):1180–1200, 2016.

[55] R. Rubinfeld. Sublinear time algorithms. In *Proc. International Congress of Mathematicians*, volume 3, pages 1095–1111, 2006.

[56] R. Rubinfeld. Can we locally compute sparse connected subgraphs? In *International Computer Science Symposium in Russia*, pages 38–47. Springer, Cham, 2017.

[57] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *ICS*, pages 223–238, 2011.

[58] T. Sarlós, A. A. Benczúr, K. Csalogány, D. Fogaras, and B. Rácz. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *Proceedings of the 15th international conference on World Wide Web*, pages 297–306. ACM, 2006.

[59] J. Suomela. Survey of local algorithms. *ACM Comput. Surv.*, 45(2):24:1–24:40, 2013.

[60] U. Wieder. Balanced allocations with heterogenous bins. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 188–193. ACM, 2007.

[61] Y. Yoshida, M. Yamamoto, and H. Ito. Improved constant-time approximation algorithms for maximum matchings and other optimization problems. *SIAM J. Comput.*, 41(4):1074–1093, 2012.