Deterministic Stateless Centralized Local Algorithms for Bounded Degree Graphs

Guy Even, Moti Medina, and Dana Ron

School of Electrical Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel
{guy,medinamo,danar}@eng.tau.ac.il

Abstract. We design centralized local algorithms for: maximal independent set, maximal matching, and graph coloring. The improvement is threefold: the algorithms are deterministic, stateless, and the number of probes is $O(\log^* n)$, where n is the number of vertices of the input graph. Our algorithms for maximal independent set and maximal matching improves over previous randomized algorithms by Alon et al. (SODA 2012) and Mansour et al. (ICALP 2012). In these previous algorithms, the number of probes and the space required for storing the state between queries are $\text{poly}(\log n)$.

We also design the first centralized local algorithm for graph coloring. Our graph coloring algorithms are deterministic and stateless. Let Δ denote the maximum degree of a graph over n vertices. Our algorithm for coloring the vertices by $\Delta + 1$ colors requires $O(\log^* n)$ probes for constant degree graphs. Surprisingly, for the case where the number of colors is $O(\Delta^2 \log \Delta)$, the number of probes of our algorithm is $O(\Delta \cdot \log^* n + \Delta^2)$, that is, the number of probes is sublinear if $\Delta = o(\sqrt{n})$, i.e., our algorithm applies for graphs with unbounded degrees.

Keywords: Centralized Local Algorithms, Sublinear Approximation Algorithms, Graph Algorithms.

1 Introduction

Local Computation Algorithms, as defined by Rubinfeld et al. [17], are algorithms that answer queries regarding (global) solutions to computational problems by performing local (sublinear time) computations on the input. The answers to all queries must be consistent with a single solution regardless of the number of possible solutions. To make this notion concrete, consider the *Maximal Independent Set* problem, which we denote by MIS. Given a graph $G = (V, E)$ as input, the local algorithm ALG gives the illusion that it "holds" a specific maximal independent set $I \subseteq V$. Namely, given any vertex v as a query, ALG answers whether v belongs to I even though ALG cannot read all of G, cannot store the solution I, and cannot even remember all the answers to previous queries. In order to answer such queries, ALG can probe the graph G by asking about the neighbors of a vertex of its choice.

A local computation algorithm may be randomized, so that the solution according to which it answers queries may depend on its internal coin flips.

A. Schulz and D. Wagner (Eds.): ESA 2014, LNCS 8737, pp. 394–405, 2014.

However, the solution should not depend on the sequence of the queries (this property is called query order obliviousness [17]). We measure the performance of a local computation algorithm by the following criteria: the maximum number of probes it makes to the input per query, the success probability over any sequence of queries, and the maximum space it uses between queries[1] . It is desired that both the probe complexity and the space complexity of the algorithm be sublinear in the size of the graph (e.g., polylog($|V|$)), and that the success probability be $1 - 1/\text{poly}(|V|)$. It is usually assumed that the maximum degree of the graph is upper-bounded by a constant, but our results are useful also for non-constant upper bounds (see also [16]). For a formal definition of local algorithms in the context of graph problems, which is the focus of this work, see Subsection 2.2.

The motivation for designing local computation algorithms is that local computation algorithms capture difficulties with very large inputs. A few examples include: (1) Reading the whole input is too costly if the input is very long. (2) In certain situations one is interested in a very small part of a complete solution. (3) Consider a setting in which different uncoordinated servers need to answer queries about a very long input stored in the cloud. The servers do not communicate with each other, do not store answers to previous queries, and want to minimize their accesses to the input.

Local computation algorithms have been designed for various graph (and hypergraph) problems, including the abovementioned MIS [17,1], hypergraph coloring [17,1], maximal matching [8] and (approximate) maximum matching [9]. Local computation algorithms also appear implicitly in works on sublinear approximation algorithms for various graph parameters, such as the size of a minimum vertex cover [14,10,19,11]. Some of these implicit results are very efficient in terms of their probe complexity (in particular, it depends on the maximum degree and not on $|V|$) but do not give the desired $1 - 1/\text{poly}(|V|)$ success probability. We compare our results to both the explicit and implicit relevant known results.

As can be gleaned from the definition in [17], local computation algorithms are closely related to *Local Distributed Algorithms* [14]. This connection is discussed in Section 2.3 (see also [4]).

In what follows we denote the aforementioned local computation model by CentLocal (where the "Cent" stands for "centralized") and the distributed (local) model by DistLocal (for a formal definition of the latter, see Subsection 2.3). We denote the number of vertices in the input graph by n and the maximum degree by Δ.

1.1 The Ranking Technique

The starting point for our results in the CentLocal model is the *ranking* technique [10,19,1,8,9]. To exemplify this, consider, once again, the MIS problem.

[1] In our algorithms the running time per query in the RAM model is at most $\text{poly}(q) \cdot \log\log n$, where q is the maximum number of probes per query and $n = |V|$.

A very simple (global "greedy") algorithm for this problem works by selecting an arbitrary ranking of the vertices and initializing I to be empty. The algorithm then considers the vertices one after the other according to their ranks and adds a vertex to I if and only if it does not neighbor any vertex already in I. Such an algorithm can be "localized" as follows. For a fixed ranking of the vertices (say, according to their IDs), given a query on a vertex v, the local algorithm performs a *restricted* DFS starting from v. The restriction is that the search continues only on paths with monotonically decreasing ranks. The local algorithm then simulates the global one on the subgraph induced by this restricted DFS.

The main problem with the above local algorithm is that the number of probes it performs when running the DFS may be very large. Indeed, for some rankings (and queried vertices), the number of probes is linear in n. In order to circumvent this problem, *random* rankings were studied [10]. This brings up two questions, which were studied in previous works, both for the MIS algorithm described above and for other ranking-based algorithms [10,19,1,8,9]. The first is to bound the number of probes needed to answer a query with high probability. The second is how to efficiently store a random ranking between queries.

1.2 Our Contributions

Orientations with bounded reachability. Our first conceptual contribution is a simple but very useful observation. Rather than considering vertex rankings, we suggest to consider *acyclic orientations* of the edges in the graph. Such orientations induce partial orders over the vertices, and partial orders suffice for our purposes. The probe complexity induced by a given orientation translates into a combinatorial measure, which we refer to as the *reachability* of the orientation. Reachability of an acyclic orientation is the maximum number of vertices that can be reached from any start vertex by directed paths (induced by the orientation). This leads us to the quest for a CentLocal algorithm that computes an orientation with bounded reachability.

Orientations and colorings. Our second conceptual contribution is that an orientation algorithm with bounded reachability can be based on a CentLocal *coloring* algorithm. Indeed, every vertex-coloring with k colors induces an orientation with reachability $O(\Delta^k)$. Towards this end, we design a CentLocal coloring algorithm that applies techniques from DistLocal colorings algorithms [3,5,7,13]. Our CentLocal algorithm is deterministic, does not use any space between queries, performs $O(\Delta \cdot \log^* n + \Delta^2)$ probes per query, and computes a coloring with $O(\Delta^2 \log \Delta)$ colors. (We refer to the problem of coloring a graph by $\text{poly}(\Delta)$ colors as $\text{poly}(\Delta)$-Color.) Our coloring algorithm yields an orientation whose reachability is $\Delta^{O(\Delta^2 \log \Delta)}$. For constant degree graphs, this implies $O(\log^* n)$ probes to obtain an orientation with constant reachability. As an application of this orientation algorithm, we also design a CentLocal algorithm for $(\Delta + 1)$-coloring.

Centralized local simulations of sequential algorithms. We apply a general transformation (similarly to what was shown in [1]) from global algorithms with certain properties to local algorithms. The transformation is based on our CentLocal orientation with bounded reachability algorithm. As a result we get deterministic CentLocal algorithms for MIS and maximal matching (MLM), which significantly improve over previous work [17,1,8], and the first CentLocal algorithm for coloring with $(\Delta+1)$ colors (We refer to the problem of coloring a graph by $\Delta+1$ colors as $(\Delta+1)$-Color). Compared to previous work, for MIS and MLM the dependence on n in the probe complexity is reduced from polylog(n) to $\log^*(n)$ and the space needed to store the state between queries is reduced from polylog(n) to zero.

1.3 Comparison to Previous Work

Comparison to previous (explicit) CentLocal *algorithms.* A comparison of our results with previous CentLocal algorithms is summarized in Table 1. The dependence on Δ of previous algorithms is not explicit; the dependency in Table 1 is based on our understanding of these results.

Table 1. A comparison between CentLocal algorithms under the assumption that $\Delta = O(1)$. Our algorithms are deterministic and stateless (i.e., the space needed to store the state between queries is zero). MLM denotes a maximal matching, MM denotes maximum matching.

Problem	Previous work			Here (Deterministic, 0-Space)
	Space	# Probes	success prob.	# Probes
MIS	$\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log^3 n$	$\Delta^{O(\Delta \cdot \log \Delta)} \cdot \log^2 n$	$1 - \frac{1}{\text{poly}(n)}$ [1]	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$ [Coro. 6]
MLM	$\Delta^{O(\Delta)} \cdot \log^3 n$	$\Delta^{O(\Delta)} \cdot \log^3 n$	$1 - \frac{1}{\text{poly}(n)}$ [8]	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$ [Coro. 6]
poly(Δ)-Color	none	none	none	$O(\Delta \cdot \log^* n + \Delta^2)$ [Thm. 3]
$(\Delta+1)$-Color	none	none	none	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$ [Coro. 6]

Comparison to previous CentLocal *oracles in sublinear approximation algorithms.* 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 is required to output an approximation of the graph parameter with high (constant) success probability. Many such algorithms work by designing an *oracle* that answers queries (e.g., a query can ask: does a given vertex belong to a fixed small vertex cover?). The sublinear approximation algorithm estimates the graph parameter by performing (a small number of) queries to the oracle. The oracles are essentially CentLocal algorithms but they tend to have constant error probability, and it is not clear how to reduce this error probability without significantly increasing their probe complexity. Furthermore, the question of bounded space needed to store the state between queries was not an issue in the design of these oracles, since only few queries are performed by the sublinear approximation algorithm. Hence, they are not usually considered to be "bona fide"

Table 2. A comparison between CentLocal oracles in sub-linear approximation algorithms and our CentLocal (deterministic) algorithms. The former algorithms were designed to work with constant success probability and a bound was given on their expected probe complexity. When presenting them as CentLocal algorithms we introduce a failure probability parameter, δ, and bound their probe complexity in terms of δ. Furthermore, the approximation ratios of the sublinear approximation algorithms were stated in additive terms, and we translate the results so as to get a multiplicative approximation.

Problem	Previous work			Here	
	# Probes	success prob.	apx. ratio	# Probes	apx. ratio
MIS	$O(\Delta^4) \cdot \text{poly}(\frac{1}{\delta}, \frac{1}{\varepsilon})$	$1-\delta$	$1-\varepsilon$ [19]	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$	1
MLM	$O(\Delta^4) \cdot \text{poly}(\frac{1}{\delta}, \frac{1}{\varepsilon})$	$1-\delta$	$1-\varepsilon$ [19]	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$	1
poly(Δ)-Color	none	none	-	$O(\Delta \cdot \log^* n + \Delta^2)$	-
$(\Delta+1)$-Color	none	none	-	$\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$	-

CentLocal algorithms. A comparison of our results and these oracles appears in Table 2.

2 Preliminaries

2.1 Notations

Let $G = (V, E)$ denote an undirected graph. Let n denote the number of vertices and m denote the number of edges. We denote the degree of v by $deg(v)$. Let Δ denote the maximum degree, i.e., $\Delta \triangleq \max_{v \in V}\{deg(v)\}$. Let $\Gamma(v)$ denote the set of neighbors of $v \in V$. The length of a path equals the number of edges along the path. We denote the length of a path p by $|p|$. For $u, v \in V$ let $dist(u, v)$ denote the length of the shortest path between u and v. The ball of radius r centered at v 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. Note that $\log^{(0)} n \triangleq n$ and if $\log^{(i)} n = 0$, we define $\log^{(j)} n = 0$, for every $j > i$. For $n \geq 1$, define $\log^* n \triangleq \min\{i : \log^{(i)} n \leq 1\}$.

2.2 The CentLocal Model

The model of centralized local computations was defined in [17]. In this section we describe this model for problems over labeled graphs.

Labeled graphs. An undirected graph $G = (V, E)$ is labeled if: (1) The vertices have unique names. 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. For simplicity, we assume that the answer also contains the port number j such that v is the jth neighbor of u. (This assumption reduces the number of probes by at most a factor of Δ.)

Online algorithms in the CentLocal *model.* An online deterministic algorithm ALG for a problem Π over labeled graphs in the CentLocal model is defined as follows. The input for the algorithm 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 $f \in sol(G, \Pi)$. The algorithm is online because it must output an evaluation of $f(q_i)$ without any knowledge of subsequent queries.

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

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

Consider, for example, the problem of computing a $(\Delta + 1)$ vertex coloring. Consistency in this example means the following. The online algorithm is input a sequence of queries, each of which is a vertex. The algorithm must output the color of each queried vertex. If a vertex is queried twice, then the algorithm must return the same color. Moreover, queried vertices that are neighbors must be colored by different colors. Thus, if all vertices are queried, then the answers constitute a legal vertex coloring that uses $(\Delta + 1)$ colors. We now describe two measures of performance that are used in the CentLocal model.

Performance measures. In the CentLocal model, two computational resources are considered: state-space and number of probes. The *state* of algorithm ALG is the information that ALG saves between queries. The *state-space* of algorithm ALG is the maximum number of bits required to encode the state of ALG. The state is used to ensure consistency. We note that the running time used to answer a query is not counted.

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 state can be encoded by s bits.*

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

Space vs. state-space. In [17] no distinction was made between the space needed to answer a query and the space needed to store the state between queries. Because the space needed to answer a query is freed after the query is answered, we only count the space needed to store the state between queries.

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

Parallelizability and query order obliviousness. In [1,8,9] two requirements are introduced: *parallelizability* and *query order obliviousness*. These requirements are fully captured by the definition of a consistent, online, deterministic algorithm with zero state-space. That is, every online algorithm that is consistent, zero-state-space, and deterministic is both parallelizable and query order oblivious.

2.3 The DistLocal Model

The model of local distributed computation is a classical model (see [7,15,18]).

The distributed computation takes place in an undirected labeled graph $G = (V, E)$. Each vertex models a processor, and communication is possible between neighboring processors. All processors execute the same algorithm. Initially, every $v \in V$ is input a local input. The computation is done in $r \in \mathbb{N}$ synchronous rounds as follows. In every round: (1) every processor receives a message from each neighbor, (2) every processor performs a computation based on its local input and the messages received from its neighbors, (3) every processor sends a message to each neighbor. We assume that a message sent in the end of round i is received in the beginning of round $i + 1$. After the rth round, every processor computes a local output.

The following assumptions are made in the DistLocal model: (1) The local input to each vertex v includes the ID of v, the degree of the vertex v, the maximum degree Δ, the number of vertices n, and the ports of v to its neighbors. (2) The IDs are distinct. For simplicity, we assume that the IDs are in the set $\{1, \ldots, n\}$. (3) The length of the messages sent in each round is not bounded.

We say that a distributed algorithm is a DistLocal$[r]$-*algorithm* if the number of communication rounds is r. Strictly speaking, a distributed algorithm is

considered *local* if r is bounded by a constant. We say that a DistLocal$[r]$-algorithm is *almost local* if $r = O(\log^* n)$. When it is obvious from the context we refer to an almost DistLocal algorithm simply by a DistLocal algorithm.

We remark that in the DistLocal model, efficiency of the algorithm executed locally by the processors is not important. Namely, one does not bound the running time required to complete each round.

Simulation of DistLocal *by* CentLocal *[14]:* Every deterministic DistLocal$[r]$-algorithm, can be simulated by a deterministic, stateless CentLocal$[O(\Delta^r)]$-algorithm. The simulation proceeds simply by probing all vertices in the ball of radius r centered at the query.

If $\Delta = 2$, then balls are simple paths (or cycles) and hence simulation of a DistLocal$[r]$-algorithm is possible by a CentLocal$[2r]$-algorithm.

3 Acyclic Orientation with Bounded Reachability

In this section we introduce the problem of *Acyclic Orientation with Bounded Reachability* (OBR). We then design a CentLocal algorithm for OBR.

Notations. Let $H = (V, A)$ denote a directed graph, where V is the set of vertices and $A \subseteq V \times V$. The *reachability set* of $v \in V$ is the set of vertices R such that there is a path from v to every vertex in R. We denote the reachability set of $v \in V$ in digraph H by $R_H(v)$. Let $r_H(v) \triangleq |R_H(v)|$ and $r_H^{\max} \triangleq \max_{v \in V} r_H(v)$. We simply write $R(v), r(v), r^{\max}$ when the digraph H is obvious from the context. We say that a digraph $H = (V, A)$ is an *orientation* of an undirected graph $G = (V, E)$ if G is an underlying graph of H.

In the problem of acyclic orientation with bounded reachability (OBR), the input is an undirected graph. The output is an orientation H of G that is acyclic. The goal is to minimize $r_H^{\max}$.

Previous works obtain an acyclic orientation by random vertex ranking [10,19,1,8,9]. We propose to obtain an acyclic orientation by vertex coloring.

Proposition 1 (Orientation via Coloring). *Every coloring by c colors induces an acyclic orientation with*

$$r^{\max} \leq \Delta \cdot \sum_{i=0}^{c-2} (\Delta - i)^i \leq \begin{cases} 2\Delta \cdot (\Delta - 1)^{c-2}, & \text{if } \Delta \geq 3, \\ 2c, & \text{if } \Delta = 2\,. \end{cases}$$

Proof. Direct each edge from a high color to a low color. By monotonicity the orientation is acyclic. Every directed path has at most c vertices, and hence the reachability is bounded as required.

3.1 A CentLocal Algorithm for OBR

In Theorem 3, we present a deterministic, stateless CentLocal$[O(\Delta \cdot \log^* n + \Delta^2)]$-algorithm that computes a vertex coloring that uses $c = O(\Delta^2 \log \Delta)$ colors. Orientation by this coloring yields an acyclic orientation with $r^{\max} \leq \Delta^c$.

Acyclic orientation can be also obtained by simulating DistLocal vertex coloring algorithms. Consider, for example, the $(\Delta+1)$ coloring using $r_1 = O(\Delta)+\frac{1}{2}\cdot\log^* n$ rounds of [2] or the $O(\Delta^2)$ coloring using $r_2 = O(\log^* n)$ rounds of [7]. CentLocal simulations of these algorithms require $O(\Delta^{r_i})$ probes. Thus, in our algorithm, the number of probes grows (slightly) slower as a function of n and is polynomial in Δ.

Our algorithm relies on techniques from two previous DistLocal coloring algorithms.

Theorem 1 ([7, Corollary 4.1]). *A $5\Delta^2 \log c$ coloring can be computed from a c coloring by a* DistLocal[1]*-algorithm.*

Theorem 2 ([12, Section 4]). *A $(\Delta+1)$ coloring can be computed by a* DistLocal[$O(\Delta^2+\log^* n)$]*-algorithm.*

Theorem 3. *An $O(\Delta^2 \log \Delta)$ coloring can be computed by a deterministic, stateless* CentLocal[$O(\Delta\cdot\log^* n+\Delta^2)$]*-algorithm.*

Proof. We begin by describing a two phased DistLocal[$O(\log^* n)$]-algorithm D that uses $O(\Delta^2\cdot\log\Delta)$ colors. Algorithm D is especially designed so that it admits an "efficient" simulation by a CentLocal-algorithm.

Consider a graph $G=(V,E)$ with a maximum degree Δ. In the first phase, the edges are partitioned into Δ^2 parts, so that the maximum degree in each part is at most 2. Let $p_i(u)$ denote the neighbor of vertex u pointed to by the ith port of u. Following Kuhn [6] we partition the edge set E as follows. Let $E_{\{i,j\}} \subseteq E$ be defined by

$$E_{\{i,j\}} \triangleq \{\{u,v\} \mid p_i(u)=v, p_j(v)=u\}.$$

Each edge belongs to exactly one part $E_{\{i,j\}}$. For each part $E_{\{i,j\}}$ and vertex u, at most two edges in $E_{\{i,j\}}$ are incident to u. Hence, the maximum degree in each part is at most 2. Each vertex can determine in a single round how the edges incident to it are partitioned among the parts. Let $G_{\{i,j\}}$ denote the undirected graph over V with edge set $E_{\{i,j\}}$.

By Theorem 2, we 3-color each graph $G_{\{i,j\}}$ in $O(\log^* n)$ rounds. This induces a vector of Δ^2 colors per vertex, hence a $3^{(\Delta^2)}$ vertex coloring of G.

In the second phase, Algorithm D applies Theorem 1 twice to reduce the number of colors to $O(\Delta^2\log\Delta)$.

We now present an efficient simulation of algorithm D by a CentLocal-algorithm C. Given a query for the color of vertex v, Algorithm C simulates the first phase of D in which a 3-coloring algorithm is executed in each part $E_{\{i,j\}}$. Since the maximum degree of each $G_{\{i,j\}}$ is two, a ball of radius r in $G_{\{i,j\}}$ contains at most $2r$ edges. In fact, this ball can be recovered by at most $2r$ probes. It follows that a CentLocal simulation of the 3-coloring of $G_{\{i,j\}}$ requires only $O(\log^* n)$ probes. Observe that if vertex v is isolated in $G_{\{i,j\}}$, then it may be colored arbitrarily (say, by the first color). A vertex v is not isolated in at most Δ parts. It follows that the simulation of the first phase requires $O(\Delta\cdot\log^* n)$ probes.

The second phase of algorithm D requires an additional Δ^2 probes, and the theorem follows.

Corollary 4. *There is a deterministic, stateless* CentLocal$[O(\Delta \cdot \log^* n + \Delta^2)]$*-algorithm for* OBR *that achieves* $r^{\max} \leq \Delta^{O(\Delta^2 \log \Delta)}$.

Proof. The CentLocal$[\Delta \cdot \log^* n + \Delta^2]$-algorithm for OBR is given a query (v, i). The algorithm answers whether the edge $(v, p_i(v))$ is an incoming edge or an outgoing edge in the orientation. The algorithm proceeds by querying the colors of v and $p_i(v)$. The orientation of the edge $(v, p_i(v))$ is determined by comparing the colors of v and $p_i(v)$.

4 Deterministic Localization of Sequential Algorithms and Applications

A common theme in online algorithms and "greedy" algorithms is that the elements are scanned in query order or in an arbitrary order, and a decision is made for each element based on the decisions of the previous elements. Classical examples of such algorithms include the greedy algorithms for maximal matchings, $(\Delta+1)$ vertex coloring, and maximal independent set. We present a compact and axiomatic CentLocal deterministic simulation of this family of algorithms, for which a randomized simulation appeared in [8]. Our deterministic simulation is based on an acyclic orientation that induces a partial order.

For simplicity, consider a graph problem Π, the solution of which is a function $g(v)$ defined over the vertices of the input graph. For example, $g(v)$ can be the color of v or a bit indicating if v belongs to a maximal independent set. (One can easily extend the definition to problems in which the solution is a function over the edges, e.g., maximal matching.)

We refer to an algorithm as a *sequential algorithm* if it fits the scheme listed as Algorithm 1. The algorithm ALG(G, σ) is input a graph $G = (V, E)$ and a bijection $\sigma : \{1, \ldots, n\} \to V$ of the vertices. Note that an element i in the domain of σ is a rank of a vertex. Hence, $\sigma(i)$ is the vertex whose rank is i, and $\sigma^{-1}(v)$ is the rank of v. The algorithm scans the vertices in the order induced by σ. It determines the value of $g(\sigma(i))$ based on the values of its neighbors whose value has already been determined. This decision is captured by the function f in Line 2. For example, in vertex coloring, f returns the smallest color that does not appear in a given a subset of colors.

Lemma 1. *Let $G = (V, E)$ be a graph, let $H = (V, A)$ be an acyclic orientation of G and let $P_> \subseteq V \times V$ denote the partial order defined by the transitive closure of H. Namely, $(u, v) \in P_>$ if and only if there exists a directed path from u to v in H. Let* ALG *denote a sequential algorithm. For every bijection $\sigma : \{1, \ldots, n\} \to V$ that is a linear extension of $P_>$ (i.e, for every $(u, v) \in P_>$ we have that $\sigma^{-1}(u) > \sigma^{-1}(v)$), the output of* ALG$(G, \sigma)$ *is the same.*

Proof. Consider two linear extensions σ and τ of $P_>$. Let g_σ denote the output of ALG(G, σ) and define g_τ analogously.

Algorithm 1. The sequential algorithm scheme

Input: A graph $G = (V, E)$ and a bijection $\sigma : \{1, \ldots, n\} \to V$.
1: **for** $i = 1$ to n **do**
2: $g(\sigma(i)) \leftarrow f\left(\left\{g(v) : v \in \Gamma(\sigma(i)) \ \& \ \sigma^{-1}(v) < i\right\}\right)$ ▷ (Decide based on "previous" neighbors)
3: **end for**
4: Output: g.

Let

$$A_\sigma(u) \triangleq \{v \in \Gamma(u) \mid \sigma^{-1}(v) < \sigma^{-1}(u)\} .$$

We claim that $A_\sigma(u) = A_\tau(u)$ for every u. If v is a neighbor of u in G, then $(u, v) \in A$ or $(v, u) \in A$. We first consider the case $(u, v) \in A$. If $(u, v) \in A$, then $(u, v) \in P_>$. Hence $\sigma^{-1}(u) > \sigma^{-1}(v)$ and $\tau^{-1}(u) > \tau^{-1}(v)$ because σ and τ are linear extensions of $P_>$. We conclude that $v \in A_\sigma(u) \cap A_\tau(u)$. Similarly, if $(v, u) \in A$, then $\sigma^{-1}(u) < \sigma^{-1}(v)$ and $\tau^{-1}(u) < \tau^{-1}(v)$. This implies that $v \notin A_\sigma(u) \cup A_\tau(u)$, and hence $A_\sigma(u) = A_\tau(u)$, as required.

We prove, by induction on i, that $g_\sigma(\sigma(i)) = g_\tau(\sigma(i))$. The induction basis, for $i = 1$, holds because $\sigma(1)$ is a minimal element according to $P_>$ (a sink in H). Hence, $A_\sigma(u) = A_\tau(u) = \emptyset$. This implies that $g_\sigma(\sigma(1)) = f(\emptyset) = g_\tau(\sigma(1))$. Turning to the induction step, we prove that the claim holds for $i > 1$, assuming it holds for every $1 \leq i' < i$. Let $u = \sigma(i)$.

Hence:

$$\begin{aligned} g_\sigma(u) &= f\left(\{g_\sigma(v)\}_{v \in A_\sigma(u)}\right) \\ &= f\left(\{g_\tau(v)\}_{v \in A_\sigma(u)}\right) \\ &= f\left(\{g_\tau(v)\}_{v \in A_\tau(u)}\right) = g_\tau(u) , \end{aligned}$$

where the second equality follows from the induction hypothesis. The third equality follows since $A_\sigma(u) = A_\tau(u)$ for every u.

Theorem 5. *For every sequential algorithm* ALG*, there exists a deterministic, stateless* CentLocal$[\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n]$*-algorithm* ALG_c *for which the following holds. For every graph G, there exists a bijection σ, such that* $\text{ALG}_c(G)$ *simulates* $\text{ALG}(G, \sigma)$*. That is, for every vertex v in G, the answer of* $\text{ALG}_c(G)$ *on query v is $g_\sigma(v)$, where g_σ denotes the output of* $\text{ALG}(G, \sigma)$*.*

Proof. Consider the acyclic orientation H of G induced by the CentLocal$[\Delta \cdot \log^* n + \Delta^2]$-algorithm for OBR presented in Corollary 4. Let $P_>$ denote the partial order that is induced by H, and let σ be any linear extension of $P_>$ (as defined in Lemma 1). On query $v \in V$ the value $g_\sigma(v)$ is computed by performing a (directed) DFS on H that traverses the subgraph of H induced by $R_H(v)$. The DFS uses the CentLocal algorithm for OBR to determine the orientation of each incident edge and continues only along outward-directed edges[2]. The

[2] Given that the CentLocal algorithm for OBR works by running a CentLocal coloring algorithm, one can actually use the latter algorithm directly.

value of $g_\sigma(v)$ is determined when the DFS backtracks from v. Since $r_H^{\max} = \Delta^{O(\Delta^2 \cdot \log \Delta)}$, by multiplying the number of probes of the OBR algorithm and $r_H^{\max}$, we obtain that $\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n$ probes suffice.

Corollary 6. *There are deterministic, stateless* CentLocal$[\Delta^{O(\Delta^2 \log \Delta)} \cdot \log^* n]$ *algorithms for* $(\Delta + 1)$*-vertex coloring, maximal independent set, and maximal matching.*

References

1. Alon, N., Rubinfeld, R., Vardi, S., Xie, N.: Space-efficient local computation algorithms. In: SODA, pp. 1132–1139 (2012)
2. Barenboim, L., Elkin, M.: Distributed (Δ+1)-coloring in linear (in Δ) time. In: STOC, pp. 111–120 (2009)
3. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. and Cont. 70(1), 32–53 (1986)
4. Even, G., Medina, M., Ron, D.: Best of two local models: Local centralized and local distributed algorithms. CoRR, abs/1402.3796 (2014)
5. Goldberg, A.V., Plotkin, S.A., Shannon, G.E.: Parallel symmetry-breaking in sparse graphs. SIDMA 1(4), 434–446 (1988)
6. Kuhn, F.: Weak graph colorings: distributed algorithms and applications. In: SPAA, pp. 138–144. ACM (2009)
7. Linial, N.: Locality in distributed graph algorithms. SICOMP 21(1), 193–201 (1992)
8. Mansour, Y., Rubinstein, A., Vardi, S., Xie, N.: Converting online algorithms to local computation algorithms. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 653–664. Springer, Heidelberg (2012)
9. Mansour, Y., Vardi, S.: A local computation approximation scheme to maximum matching. In: Raghavendra, P., Raskhodnikova, S., Jansen, K., Rolim, J.D.P. (eds.) RANDOM 2013 and APPROX 2013. LNCS, vol. 8096, pp. 260–273. Springer, Heidelberg (2013)
10. H.: N Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In: FOCS, pp. 327–336 (2008)
11. Onak, K., Ron, D., Rosen, M., Rubinfeld, R.: A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In: SODA, pp. 1123–1131 (2012)
12. Panconesi, A., Rizzi, R.: Some simple distributed algorithms for sparse networks. Dist. Comp. 14(2), 97–100 (2001)
13. Panconesi, A., Sozio, M.: Fast primal-dual distributed algorithms for scheduling and matching problems. Dist. Comp. 22(4), 269–283 (2010)
14. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theo. Comp. Sci. 381(1), 183–196 (2007)
15. Peleg, D.: Distributed computing: a locality-sensitive approach, vol. 5. SIAM (2000)
16. Reingold, O., Vardi, S.: New techniques and tighter bounds for local computation algorithms. CoRR, abs/1404.5398 (2014)
17. Rubinfeld, R., Tamir, G., Vardi, S., Xie, N.: Fast local computation algorithms. In: ICS, pp. 223–238 (2011)
18. Suomela, J.: Survey of local algorithms. ACM Comput. Surv. 45(2), 24:1–24:40 (2013)
19. Yoshida, Y., Yamamoto, M., Ito, H.: Improved constant-time approximation algorithms for maximum matchings and other optimization problems. SICOMP 41(4), 1074–1093 (2012)