Space-efficient Local Computation Algorithms*

Noga Alon[†] Ronitt Rubinfeld[‡] Shai Vardi [§] Ning Xie [¶]

Abstract

Recently Rubinfeld et al. (ICS 2011, pp. 223–238) proposed a new model of sublinear algorithms called *local computation algorithms*. In this model, a computation problem F may have more than one legal solution and each of them consists of many bits. The local computation algorithm for F should answer in an online fashion, for any index i, the i^{th} bit of some legal solution of F. Further, all the answers given by the algorithm should be consistent with at least one solution of F. In this work, we continue the study of local computation algorithms. In particular, we develop a technique which under certain conditions can be applied to construct local computation algorithms that run not only in polylogarithmic time but also in polylogarithmic *space*. Moreover, these local computation algorithms are easily parallelizable and can answer all parallel queries consistently. Our main technical tools are pseudorandom numbers with bounded independence and the theory of branching processes.

1 Introduction

The classical view of algorithmic analysis, in which the algorithm reads the entire input, performs a computation and then writes out the entire output, is less applicable in the context of computations on massive data sets. To address this difficulty, several alternative models of computation have been adopted, including distributed computation as well as various sub-linear time and space models.

Local computation algorithms (LCAs) were proposed in [25] to model the scenario in which inputs to and outputs from the algorithms are large, such that writing out the entire output requires an amount of time that is unacceptable. On the other hand, only small portions of the output are required at any point in time by any specific user. LCAs support queries to the output by the user, such that after each query to a specified location i, the LCA outputs the value of the output at location i. LCAs were inspired by and intended as a generalization of several models that appear in the literature, including local algorithms, locally decodable codes and local reconstruction algorithms. LCAs whose time complexity is efficient in terms of the amount of solution requested by the user have been given for various combinatorial and coding theoretic problems.

One difficulty is that for many computations, more than one output is considered to be valid, yet the values returned by the LCA over time must be consistent. Often, the straightforward solutions ask that the LCA store intermediate values of the computations in order to maintain consistency for later computations. Though standard techniques can be useful for recomputing the values of random coin tosses in a straightforward manner, some algorithms (e.g., many greedy algorithms) choose very different solutions based on the order of input queries. Thus, though the time requirements of the LCA may be efficient for each query, it is not always clear how to *bound the storage requirements* of the LCA by a function that is sublinear in the size of the query history. It is this issue that we focus on in this paper.

1.1 Our main results

Before stating our main results, we mention two additional desirable properties of LCAs. Both of these properties are achieved in our constructions of LCAs with small storage requirements. The first is that an LCA should be *query oblivious*, that is the outputs of $\mathcal{A}$ should not depend on the order of the queries but only on the input and the random bits generated on the random tape of $\mathcal{A}$. The second is that the LCA should be *parallelizable*, i.e., that it is able to answer multiple queries simultaneously in a consistent manner.

[*] A full version of this paper is available as a technical report at http://arxiv.org/abs/1109.6178.

[†] Sackler School of Mathematics and Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel and Institute for Advanced Study, Princeton, New Jersey 08540, USA. E-mail: nogaa@tau.ac.il. Research supported in part by an ERC Advanced grant, by a USA-Israeli BSF grant and by NSF grant No. DMS-0835373.

[‡] CSAIL, MIT, Cambridge, MA 02139, USA and School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: ronitt@csail.mit.edu. Supported by NSF grants CCF-0728645 and CCF-1065125, Marie Curie Reintegration grant PIRG03-GA-2008-231077 and the Israel Science Foundation grant nos. 1147/09 and 1675/09.

[§] School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E-mail: shaivardi@gmail.com. Supported by Israel Science Foundation grant no. 1147/09.

[¶] CSAIL, MIT, Cambridge MA 02139, USA. E-mail: ningxie@csail.mit.edu. Research supported by NSF grants CCF-0728645, CCF-0729011 and CCF-1065125.

All the LCAs given in [26] suffer from one or more of the following drawbacks: the worst case space complexity is linear, the LCA is not query oblivious, and the LCA is not parallelizable. We give new techniques to construct LCAs for the problems studied in [26] which run in polylogarithmic time as well as polylogarithmic space. Moreover, all of the LCAs are query oblivious and easily parallelizable.

Theorem 1.1. (Main Theorem 1 (informal)) *There is an LCA for* Hypergraph Coloring *that runs in polylogarithmic time and space. Moreover, the LCA is query oblivious and parallelizable.*

Theorem 1.2. (Main Theorem 2 (informal)) *There is an LCA for* Maximal Independent Set *that runs in polylogarithmic time and space. Moreover, the LCA is query oblivious and parallelizable.*

We remark that following [26], analogous techniques can be applied to construct LCAs with all of the desirable properties for the radio network problem and k-CNF problems.

1.2 Techniques There are two main technical obstacle in making the LCAs constructed in [26] space efficient, query oblivious and parallelizable. The first is that LCAs need to remember all the random bits used in computing previous queries. The second issue is more subtle – [26] give LCAs based on algorithms which use very little additional time resources per query as they simulate greedy algorithms. These LCAs output results that depend directly on the orders in which queries are fed into the algorithms.

We address the randomness issue first. The space inefficient LCAs constructed in [26] for the problems of concern to us are probabilistic by nature. Consistency among answers to the queries seems to demand that the algorithm keeps track of all random bits used so far, which would incur linear space complexity. A simple but very useful observation is that all the computations are local and thus involve a very small number of random bits. Therefore we may replace the truly random bits with random variables of limited independence. The construction of small sample space k-wise independent random variables of Alon et al. [3] allows us to reduce the space complexity from linear to polylogarithmic. This allows us to prove our main theorem on the LCA for the maximal independent set problem. It is also an important ingredient in constructing our LCA for *Hypergraph Coloring*. We believe such a technique will be a standard tool in future development of LCAs.

For *Hypergraph Coloring*, we need to also address the second issue raised above. The original LCA for *Hypergraph Coloring* in [26] emulates Alon's algorithm [2]. Alon's algorithm runs in three phases. During the first phase, it colors *all* vertices in an *arbitrary* order. Such an algorithm looks "global" in nature and it is therefore non-trivial to turn it into an LCA. In [26], they use the order of vertices being queried as the order of coloring in Alon's algorithm, hence the algorithm needs to store all answers to previous queries and requires linear space in computation.

We take a different approach to overcome this difficulty. Observe that there is some "local" dependency among the colors of vertices – namely, the color of any vertex depends only on the colors of at most a constant number, say D, other vertices. The colors of these vertices in turn depend on the colors of their neighboring vertices, and so on. We can model the hypergraph coloring process by a query tree: Suppose the color of vertex x is now queried. Then the root node of the query tree is x, the nodes on the first level are the vertices whose colors the color of x depends on. In general, the colors of nodes on level i depends on[1] the colors of nodes on level $i + 1$. Note that the query tree has degree bound D and moreover, the size of the query tree clearly depends on the order in which vertices are colored, since the color of a vertex depends only on vertices that are colored before it. In particular, if x is the k^{th} vertex to be colored, then the query tree contains at most k vertices.

An important fact to note is that Alon's algorithm works for *any* order, in particular, it works for a random order. Therefore we can apply the random order method of Nguyen and Onak [21]: generate a random number $r \in [0, 1]$, called the *rank*, and use these ranks to prune the original query tree into a *random query tree* $\mathcal{T}$. Specifically, $\mathcal{T}$ is defined recursively: the root of $\mathcal{T}$ is still x. A node z is in $\mathcal{T}$ if its parent node y in the original query tree is in $\mathcal{T}$ and $r(z) < r(y)$. Intuitively, a random query tree is small and indeed it is surprisingly small [21]: the expected size of $\mathcal{T}$ is $\frac{e^D-1}{D}$, a constant!

Therefore, if we color the vertices in the hypergraph in a random order, the *expected* number of vertices we need to color is only a constant. However, such an "average case" result is insufficient for our LCA purpose: what we need is a "worst case" result which tells almost surely how large a random query tree will be. In other words, we need a concentration result on the sizes of the random query trees. The previous techniques in [21, 29] do not seem to work in this setting.

Consider the worst case in which the rank of the root node x is 1. A key observation is, although there

[1] In fact, they may depend on the colors of some nodes on levels lower than i. However, as we care only about query complexity, we will focus on the worst case that the query relations form a tree.

are D child nodes of x, only the nodes whose ranks are close to 1 are important, as the child nodes with smaller ranks will die out quickly. But in expectation there will be very few important nodes! This inspires us to partition the random query tree into $D+1$ levels based on the ranks of the nodes, and analyze the sizes of trees on each level using the theory of branching processes. In particular, we apply a quantitative bound on the total number of off-springs of a Galton-Watson process [23] to show that, for any $m > 0$, with probability at least $1 - 1/m^2$ the size of a random query tree has at most $C(D) \log^{D+1} m$ vertices, where $C(D)$ is some constant depending only on D. We conjecture that the upper bound can be further reduced to $C(D) \log m$.

However, the random order approach raise another issue: how do we store the ranks of all vertices? Observe that in constructing a random query tree, the actual values of the ranks are never used – only the relative orders between vertices matter. This fact together with the fact that all computations are local enables us to replace the rank function with some pseudorandom ordering among the vertices, see Section 4 for formal definition and construction. The space complexity of the pseudorandom ordering is only polylogarithmic, thus making the total space complexity of the LCA also polylogarithmic.

1.3 Other related work

Locally decodable codes [10] which given an encoding of a message, provide quick access to the requested bits of the original message, can be viewed as LCAs. Known constructions of LDCs are efficient and use small space [28]. LCAs generalize the *reconstruction* models described in [1, 7, 27, 8]. These models describe scenarios where an input string that has a certain property, such as monotonicity, is assumed to be corrupted at a relatively small number of locations. The reconstruction algorithm gives fast query access to an uncorrupted version of the string that is close to the original input. Most of the works mentioned are also efficient in terms of space.

In [25], it is noted that the model of LCAs is related to *local algorithms*, studied in the context of distributed computing [20, 18, 13, 14, 15, 12, 11]. This is due to a reduction given by Parnas Ron [24] which allows one to construct (sequential) LCAs based on constant round distributed algorithms. Note that this relationship does not immediately yield space-efficient local algorithms, nor does it yield sub-linear time LCAs when used with parallel or distributed algorithms whose round complexity is $O(\log n)$.

Recent exciting developments in sublinear time algorithms for sparse graph and combinatorial optimization problems have led to new constant time algorithms for approximating the size of a minimum vertex cover, maximal matching, maximum matching, minimum dominating set, minimum set cover, packing and covering problems (cf. [24, 17, 21, 29]). For example, for *Maximal Independent Set*, these algorithms construct a constant-time oracle which for most, but not all, vertices outputs whether or not the vertex is part of the independent set. For the above approximation algorithms, it is not necessary to get the correct answer for each vertex, but for LCAs, which must work for any sequence of online inputs, the requirements are more stringent, thus the techniques are not applicable without modification.

1.4 Organization

The rest of the paper is organized as follows. Some preliminaries and notations that we use throughout the paper appear in Section 2. We then prove our main technical result, namely the bound on the sizes of random query trees in Section 3. In Section 4 we construct pseudorandom orderings with small space. Once these technical tools are developed, applying them to designing local computation algorithms are relatively easy – we omit the details from this conference version and refer interested readers to [4].

2 Preliminaries

Unless stated otherwise, all logarithms in this paper are to the base 2. Let $n \geq 1$ be a natural number. We use $[n]$ to denote the set $\{1, \ldots, n\}$.

All graphs in this paper are undirected graphs. Let $G = (V, E)$ be a graph. The *distance* between two vertices u and v in $V(G)$, denoted by $d_G(u, v)$, is the length of a shortest path between the two vertices. We write $N_G(v) = \{u \in V(G) : (u, v) \in E(G)\}$ to denote the neighboring vertices of v. Furthermore, let $N_G^+(v) = N(v) \cup \{v\}$. Let $d_G(v)$ denote the degree of a vertex v.

2.1 Local computation algorithms

We present our model of local computation algorithms: Let F be a computational problem and x be an input to F. Let $F(x) = \{y \mid y \text{ is a valid solution for input } x\}$. The *search problem* is to find any $y \in F(x)$.

Definition 2.1. ((t, s, δ)-local algorithms [26]) *Let x and $F(x)$ be defined as above. A $(t(n), s(n), \delta(n))$-local computation algorithm $\mathcal{A}$ is a (randomized) algorithm which implements query access to an arbitrary $y \in F(x)$ and satisfies the following: $\mathcal{A}$ gets a sequence of queries $i_1, \ldots, i_q$ for any $q > 0$ and after each query i_j it must produce an output y_{i_j} satisfying that the outputs $y_{i_1}, \ldots, y_{i_q}$ are substrings of some $y \in F(x)$. The probability of success over all q queries must be at least $1 - \delta(n)$. $\mathcal{A}$ has access to a random tape and local* *computation memory on which it can perform current computations as well as store and retrieve information from previous computations. We assume that the input x, the local computation tape and any random bits used are all presented in the RAM word model, i.e., $\mathcal{A}$ is given the ability to access a word of any of these in one step. The running time of $\mathcal{A}$ on any query is at most $t(n)$, which is sublinear in n, and the size of the local computation memory of $\mathcal{A}$ is at most $s(n)$. Unless stated otherwise, we always assume that the error parameter $\delta(n)$ is at most some constant, say,* 1/3. *We say that $\mathcal{A}$ is a* strongly local computation algorithm *if both $t(n)$ and $s(n)$ are upper bounded by* $\log^c n$ *for some constant c.*

Two important properties of LCAs are as follows:

Definition 2.2. (Query oblivious[26]) *We say an LCA $\mathcal{A}$ is* query order oblivious *(*query oblivious *for short) if the outputs of $\mathcal{A}$ do not depend on the order of the queries but depend only on the input and the random bits generated on the random tape of $\mathcal{A}$.*

Definition 2.3. (Parallelizable[26]) *We say an LCA $\mathcal{A}$ is* parallelizable *if $\mathcal{A}$ supports parallel queries, that is the LCA is able to answer multiple queries simultaneously so that all the answers are consistent.*

2.2 k-wise independent random variables

Let $1 \le k \le n$ be an integer. A distribution $D : \{0,1\}^n \to \mathbb{R}^{\ge 0}$ is *k-wise independent* if restricting D to any index subset $S \subset [n]$ of size at most k gives rise to a uniform distribution. A random variable is said to be k-wise independent if its distribution function is k-wise independent. Recall that the support of a distribution D, denoted $\text{supp}(D)$, is the set of points at which $D(x) > 0$. We say a discrete distribution D is *symmetric* if $D(x) = 1/|\text{supp}(D)|$ for every $x \in \text{supp}(D)$. If a distribution $D : \{0,1\}^n \to \mathbb{R}^{\ge 0}$ is symmetric with $|\text{supp}(D)| \le 2^m$ for some $m \le n$, then we may index the elements in the support of D by $\{0,1\}^m$ and call m the *seed length* of the random variable whose distribution is D. We will need the following construction of k-wise independent random variables over $\{0,1\}^n$ with small symmetric sample space.

Theorem 2.1. ([3]) *For every $1 \le k \le n$, there exists a symmetric distribution $D : \{0,1\}^n \to \mathbb{R}^{\ge 0}$ of support size at most $n^{\lfloor \frac{k}{2} \rfloor}$ and is k-wise independent. That is, there is a k-wise independent random variable $x = (x_1, \ldots, x_n)$ whose seed length is at most $O(k \log n)$. Moreover, for any $1 \le i \le n$, x_i can be computed in space $O(k \log n)$.*

3 Bounding the size of a random query tree

3.1 The problem and our main result

Consider the following scenario which was first studied by [21] in the context of constant-time approximation algorithms for maximal matching and some other problems. We are given a graph $G = (V, E)$ of bounded degree D. A real number $r(v) \in [0,1]$ is assigned independently and uniformly at random to every vertex v in the graph. We call this random number the *rank* of v. Each vertex in the graph G holds an input $x(v) \in R$, where the range R is some finite set. A randomized Boolean function F is defined inductively on the vertices in the graph such that $F(v)$ is a function of the input $x(v)$ at v as well as the values of F at the neighbors w of v for which $r(w) < r(v)$. The main question is, in order to compute $F(v_0)$ for any vertex v_0 in G, how many queries to the inputs of the vertices in the graph are needed?

Here, for the purpose of upper bounding the query complexity, we may assume for simplicity that the graph G is D-regular and furthermore, G is an infinite D-regular tree rooted at v_0. It is easy to see that making such modifications to G can never decrease the query complexity of computing $F(v_0)$.

Consider the following question. We are given an infinite D-regular tree $\mathcal{T}$ rooted at v_0. Each node w in $\mathcal{T}$ is assigned independently and uniformly at random a real number $r(w) \in [0,1]$. For every node w other than v_0 in $\mathcal{T}$, let $\text{parent}(w)$ denote the parent node of w. We grow a (possibly infinite) subtree T of $\mathcal{T}$ rooted at v as follows: a node w is in the subtree T if and only if $\text{parent}(w)$ is in T and $r(w) < r(\text{parent}(w))$ (for simplicity we assume all the ranks are distinct real numbers). That is, we start from the root v, add all the children of v whose ranks are smaller than that of v to T. We keep growing T in this manner where a node $w' \in T$ is a leaf node in T if the ranks of its D children are all larger than $r(w')$. We call the random tree T constructed in this way a *query tree* and we denote by $|T|$ the random variable that corresponds to the size of T. We would like to know what are the typical values of $|T|$.

Following [21, 22], we have that, for any node w that is at distance t from the root v_0, $\Pr[w \in T]$=1/(t+1)! as such an event happens if and only if the ranks of the $t+1$ nodes along the shortest path from v_0 to w is in monotone decreasing order. It follows from linearity of expectation that the expected value of $|T|$ is given by the elegant formula

$$\mathbb{E}[|T|] = \sum_{t=0}^{\infty} \frac{D^t}{(t+1)!} = \frac{e^D - 1}{D},$$

which is a constant depending only on the degree bound D.

Our main result in this section can be regarded as showing that in fact $|T|$ is highly concentrated around its mean:

THEOREM 3.1. *For any degree bound $D \geq 2$, there is a constant $C(D)$ which depends on D only such that for all large enough N,*

$$\Pr[|T| > C(D) \log^{D+1} N] < 1/N^2.$$

3.2 Breaking the query tree into levels A key idea in the proof is to break the query tree into levels and then upper bound the sizes of the subtrees on each level separately. First partition the interval $[0, 1]$ into $D + 1$ sub-intervals: $I_i := (1 - \frac{i}{D+1}, 1 - \frac{i-1}{D+1}]$ for $i = 1, 2, \ldots, D$ and $I_{D+1} = [0, \frac{1}{D+1}]$. We then decompose the query tree T into $D + 1$ levels such that a node $v \in T$ is said to be on level i if $r(v) \in I_i$. For ease of exposition, in the following we consider the worst case that $r(v_0) \in I_1$. Then the vertices of T on level 1 form a tree which we call $T_1 = T_1^{(1)}$ rooted at v_0. The vertices of T on level 2 will in general form a set of trees $\{T_2^{(1)}, \ldots, T_2^{(m_2)}\}$, where the total number of such trees m_2 is at most D times the number of nodes in T_1 (we have only inequality here because some of the child nodes in $\mathcal{T}$ of the nodes in T_1 may fall into levels 2, 3, etc). Finally the nodes on level $D+1$ form a forest $\{T_{D+1}^{(1)}, \ldots, T_{D+1}^{(m_{D+1})}\}$. Note that all these trees $\{T_i^{(j)}\}$ are generated by the same stochastic process, as the ranks of all nodes in $\mathcal{T}$ are i.i.d. random variables. The next lemma shows that each of the subtrees on any level is of size $O(\log N)$ with probability at least $1 - 1/N^3$,

LEMMA 3.1. *For any $1 \leq i \leq D + 1$ and any $1 \leq j \leq m_i$, with probability at least $1 - 1/N^3$, $|T_i^{(j)}| = O(\log N)$.*

One can see that Theorem 3.1 follows directly from Lemma 3.1: Once again we consider the worst case that $r(v_0) \in I_1$. By Lemma 3.1, the size of T_1 is at most $O(\log N)$ with probability at least $1 - 1/N^3$. In what follows, we always condition our argument upon that this event happens. Notice that the root of any tree on level 2 must have some node in T_1 as its parent node; it follows that m_2, the number of trees on level 2, is at most D times the size of T_1, hence $m_2 = O(\log N)$. Now applying Lemma 3.1 to each of the m_2 trees on level 2 and assume that the high probability event claimed in Lemma 3.1 happens in each of the subtree cases, we get that the total number of nodes at level 2 is at most $O(\log^2 N)$. Once again, any tree on level 3 must have some node in either level 1 or level 2 as its parent node, so the total number of trees on level 3 is also at most $D(O(\log N) + O(\log^2 N)) = O(\log^2 N)$. Applying this argument inductively, we get that $m_i = O(\log^{i-1} N)$ for $i = 2, 3, \ldots, D + 1$. Consequently, the total number of nodes at all $D + 1$ levels is at most $O(\log N) + O(\log^2 N) + \cdots + O(\log^{D+1} N) = O(\log^{D+1} N)$, assuming the high probability event in Lemma 3.1 holds for all the subtrees in all the levels. By the union bound, this happens with probability at least $1 - O(\log^{D+1} N)/N^3 > 1 - 1/N^2$, thus proving Theorem 3.1.

The proof of Lemma 3.1 requires results in branching processes, in particular the Galton-Watson processes.

3.3 Galton-Watson processes Consider a Galton-Watson process defined by the probability function $\mathbf{p} := \{p_k; k = 0, 1, 2, \ldots\}$, with $p_k \geq 0$ and $\sum_k p_k = 1$. Let $f(s) = \sum_{k=0}^{\infty} p_k s^k$ be the generating function of $\mathbf{p}$. For $i = 0, 1, \ldots,$ let Z_i be the number of off-springs in the i^{th} generation. Clearly $Z_0 = 1$ and $\{Z_i : i = 0, 1, \ldots\}$ form a Markov chain. Let $m := \mathbb{E}[Z_1] = \sum_k kp_k$ be the expected number of children of any individual. The classical result of the Galton-Watson processes is that the *survival probability* (namely $\lim_{n\to\infty} \Pr[Z_n > 0]$) is zero if and only if $m \leq 1$. Let $Z = Z_0 + Z_1 + \cdots$ be the sum of all off-springs in all generations of the Galton-Watson process. The following result of Otter is useful in bounding the probability that Z is large.

THEOREM 3.2. ([23]) *Suppose $p_0 > 0$ and that there is a point $a > 0$ within the circle of convergence of f for which $af'(a) = f(a)$. Let $\alpha = a/f(a)$. Let $t = \gcd\{r : p_r > 0\}$, where* gcd *stands for greatest common divisor. Then*

(3.1)

$$\Pr[Z = n] = \begin{cases} t\left(\frac{a}{2\pi\alpha f''(a)}\right)^{1/2} \alpha^{-n} n^{-3/2} + O(\alpha^{-n} n^{-5/2}), & \textit{if } n \equiv 1 \pmod{t}; \\ 0, & \textit{if } n \not\equiv 1 \pmod{t}. \end{cases}$$

In particular, if the process is *non-arithmetic*, i.e. $\gcd\{r : p_r > 0\} = 1$, and $\frac{a}{\alpha f''(a)}$ is finite, then

$$\Pr[Z = n] = O(\alpha^{-n} n^{-3/2}),$$

and consequently $\Pr[Z \geq n] = O(\alpha^{-n})$.

3.4 Proof of Lemma 3.1 To simplify exposition, we prove Lemma 3.1 for the case of tree T_1. Recall that T_1 is constructed recursively as follows: for every child node v of v_0 in $\mathcal{T}$, we add v to T_1 if $r(v) < r(v_0)$ and $r(v) \in I_1$. Then for every child node v of v_0 in T_1, we add the child node w of v in $\mathcal{T}$ to T_1 if $r(w) < r(v)$ and $r(w) \in I_1$. We repeat this process until there is no node that can be added to T_1.

Once again, we work with the worst case that $r(v_0) = 1$. To upper bound the size of T_1, we consider a related random process which also grows a subtree of $\mathcal{T}$ rooted at v_0, and denote it by T_1'. The process that grows T_1' is the same as that of T_1 except for the following difference: if $v \in T_1'$ and w is a child node of v in $\mathcal{T}$, then we add w to T_1' as long as $r(w) \in I_1$, but give up the requirement that $r(w) < r(v)$. Clearly, we always have $T_1 \subseteq T_1'$ and hence $|T_1'| \geq |T_1|$.

Note that the random process that generates T_1' is in fact a Galton-Watson process, as the rank of each node in $\mathcal{T}$ is independently and uniformly distributed in $[0, 1]$. Since $|I_1| = 1/(D + 1)$, the probability function is

$$\mathbf{p} = \{(1-q)^D, \binom{D}{1} q(1-q)^{D-1}, \binom{D}{2} q^2(1-q)^{D-2}, \\ \ldots, \binom{D}{D-1} q^{D-1}(1-q), q^D\},$$

where $q := 1/(D+1)$ is the probability that a child node in $\mathcal{T}$ appears in T_1' when its parent node is in T_1'. Note that the expected number of children of a node in T_1' is $Dq = D/(D+1) < 1$, so the tree T_1' is a finite tree with probability one.

The generating function of $\mathbf{p}$ is

$$f(s) = (1 - q + qs)^D,$$

as the probability function $\{p_k\}$ obeys the binomial distribution $p_k = b(k, D, q)$. In addition, the convergence radius of f is $\rho = \infty$ since $\{p_k\}$ has only a finite number of non-zero terms.

Solving the equation $af'(a) = f(a)$ yields $a = \frac{1-q}{q(D-1)} = \frac{D}{D-1}$. It follows that (since $D \geq 2$)

$$f''(a) = q^2 D(D-1)\left(1 - q + \frac{1-q}{D-1}\right)^{D-2} > 0,$$

hence the coefficient in (3.1) is non-singular.

Let $\alpha(D) := a/f(a) = 1/f'(a)$, then

$$\begin{aligned} 1/\alpha(D) &= f'(a) \\ &= \frac{D}{D+1}\left(\frac{D^2}{D^2-1}\right)^{D-1} \\ &= \left(1 + \frac{1}{D^2-1}\right)^{(D^2-1)/(D+1)} \frac{D}{D+1} \\ &< e^{1/(D+1)} \frac{D}{D+1} \\ &< \left((1 + \frac{1}{D})^{D+1}\right)^{1/(D+1)} \frac{D}{D+1} \\ &= 1, \end{aligned}$$

where in the third and the fourth steps we use the inequality (see e.g. [19]) that $(1 + \frac{1}{t})^t < e < (1 + \frac{1}{t})^{t+1}$ for any positive integer t. This shows that $\alpha(D)$ is a constant greater than 1.

Now applying Theorem 3.2 to the Galton-Watson process which generates T_1' (note that $t = 1$ in our case) gives that, for all large enough n, $\Pr[|T_1'| = n] \leq 2^{-cn}$ for some constant c. It follows that $\Pr[|T_1'| \geq n] \leq \sum_{i=n}^{\infty} 2^{-ci} \leq 2^{-\Omega(n)}$ for all large enough n. Hence for all large enough N, with probability at least $1 - 1/N^3$, $|T_1| \leq |T_1'| = O(\log N)$. This completes the proof of Lemma 3.1.

4 Construction of almost k-wise independent random orderings

An important observation that enables us to make some of our local algorithms run in polylogarithmic space is the following. In the construction of a random query tree $\mathcal{T}$, we do not need to generate a random real number $r(v) \in [0, 1]$ independently for each vertex $v \in \mathcal{T}$; instead only the *relative orderings* among the vertices in $\mathcal{T}$ matter. Indeed, when generating a random query tree, we only compare the ranks between a child node w and its parent node v to see if $r(w) < r(v)$; the absolute values of $r(w)$ and $r(v)$ are irrelevant and are used only to facilitate our analysis in Section 3. Moreover, since (almost surely) all our computations in the local algorithms involve only a very small number of, say at most k, vertices, so instead of requiring a random source that generates total independent random ordering among all nodes in the graph, any pseudorandom generator that produces *k-wise independent random ordering* suffices for our purpose. We now give the formal definition of such orderings.

Let $m \geq 1$ be an integer. Let $\mathcal{D}$ be any set with m elements. For simplicity and without loss of generality, we may assume that $\mathcal{D} = [m]$. Let $\mathcal{R}$ be a totally ordered set. An *ordering* of $[m]$ is an injective function $r : [m] \to \mathcal{R}$. Note that we can *project* r to an element in the symmetric permutation group $\mathcal{S}_m$ in a natural way: arrange the elements $\{r(1), \ldots, r(m)\}$ in $\mathcal{R}$ in the monotone increasing order and call the permutation of $[m]$ corresponding to this ordering the *projection of* r onto $\mathcal{S}_m$ and denote it by $P_{\mathcal{S}_m} r$. In general the projection $P_{\mathcal{S}_m}$ is not injective. Let $\mathbf{r} = \{r_i\}_{i \in I}$ be any family of orderings indexed by I. The *random ordering* $D_{\mathbf{r}}$ of $[m]$ is a distribution over a family of orderings $\mathbf{r}$. For any integer $2 \leq k \leq m$, we say a random ordering $D_{\mathbf{r}}$ is *k-wise independent* if for any subset $S \subseteq [m]$ of size k, the restriction of the projection onto $\mathcal{S}_m$ of $D_{\mathbf{r}}$ over S is uniform over all the k! possible orderings among the k elements in S. A random ordering $D_{\mathbf{r}}$ is said to *ϵ-almost k-wise independent* if the statistical distance between $D_{\mathbf{r}}$ is at most ϵ from some k-wise independent random ordering. Note that our definitions of k-wise independent random ordering and almost k-wise independent random ordering are different from that of k-wise independent permutation and almost k-wise independent permutation (see e.g. [9]), where the latter requires that the function to be a *permutation* (i.e., the domain and the range of the function are the same set). In this section we give a construction of $\frac{1}{m^2}$-almost k-wise independent random ordering whose seed length is $O(k \log^2 m)$. In our later applications $k = \text{polylog} m$ so the seed length of the almost k-wise independent random ordering is also polylogarithmic.

THEOREM 4.1. *Let $m \geq 2$ be an integer and let $2 \leq k \leq m$. Then there is a construction of $\frac{1}{m^2}$-almost k-wise independent random ordering over $[m]$ whose seed length is $O(k \log^2 m)$.*

Proof. For simplicity we assume that m is a power of 2. Let $s = 4 \log m$. We generate s *independent* copies of k-wise independent random variables $Z_1, \ldots, Z_s$ with each Z_ℓ, $1 \leq \ell \leq s$, in $\{0,1\}^m$. By Theorem 2.1, the seed length of each random variable Z_ℓ is $O(k \log m)$ and therefore the total space needed to store these random seeds is $O(k \log^2 m)$. Let these k-wise independent m-bit random variables be

$$\begin{aligned} Z_1 &= z_{1,1}, \ldots, z_{1,m}; \\ Z_2 &= z_{2,1}, \ldots, z_{2,m}; \\ &\cdots\cdots \\ Z_s &= z_{s,1}, \ldots, z_{s,m}. \end{aligned}$$

Now for every $1 \leq i \leq m$, we view each $r(i) \overset{\text{def}}{=} z_{1,i} z_{2,i} \cdots z_{s,i}$ as an integer in $\{0, 1, \ldots, 2^s - 1\}$ written in the s-bit binary representation and use $r : [m] \to \{0, 1, \ldots, 2^s - 1\}$ as the ranking function to order the m elements in the set. We next show that, with probability at least $1 - 1/m^2$, $r(1), \ldots, r(m)$ are distinct m integers.

Let $1 \leq i < j \leq m$ be any two distinct indices. For every $1 \leq \ell \leq s$, since $z_{\ell,1}, \ldots, z_{\ell,m}$ are k-wise independent and thus also pair-wise independent, it follows that $\Pr[z_{\ell,i} = z_{\ell,j}] = 1/2$. Moreover, as all $Z_1, \ldots, Z_s$ are independent, we therefore have

$$\begin{aligned} \Pr[r(i) = r(j)] &= \Pr[z_{\ell,i} = z_{\ell,j} \text{ for every } 1 \leq \ell \leq s] \\ &= \prod_{\ell=1}^{s} \Pr[z_{\ell,i} = z_{\ell,j}] \\ &= (1/2)^s \\ &= 1/m^4. \end{aligned}$$

Applying a union bound argument over all $\binom{m}{2}$ distinct pairs of indices gives that with probability at least $1 - 1/m^2$, all these m numbers are distinct.

Since each Z_ℓ, $1 \leq \ell \leq s$, is a k-wise independent random variable in $\{0,1\}^m$, therefore for any subset $\{i_1, \ldots, i_k\}$ of k indices, $(r(i_1), \ldots, r(i_k))$ is distributed uniformly over all 2^{ks} tuples. By symmetry, conditioned on that $r(i_1), \ldots, r(i_k)$ are all distinct, the restriction of the ordering induced by the ranking function r to $\{i_1, \ldots, i_k\}$ is completely independent. Finally, since the probability that $r(1), \ldots, r(m)$ are not distinct is at most $1/m^2$, it follows that the random ordering induced by r is $\frac{1}{m^2}$-almost k-wise independent.

Acknowledgments

We would like to thank Tali Kaufman and Krzysztof Onak for enlightening discussions.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.

[2] N. Alon. A parallel algorithmic version of the Local Lemma. *Random Structures and Algorithms*, 2:367–378, 1991.

[3] N. Alon, L. Babai, and A. Itai. A fast and simple randomized algorithm for the maximal independent set problem. *Journal of Algorithms*, 7:567–583, 1986.

[4] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. Technical report, September 2011. `http://arxiv.org/abs/1109.6178`.

[5] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley and Sons, second edition, 2000.

[6] J. Beck. An algorithmic approach to the Lovász Local Lemma. *Random Structures and Algorithms*, 2:343–365, 1991.

[7] B. Chazelle and C. Seshadhri. Online geometric reconstruction. In *SoCG*, pages 386 – 394, 2006.

[8] M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *Proc. 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011.

[9] E. Kaplan, M. Naor, and O. Reingold. Derandomized constructions of k-wise (almost) independent permutations. *Algorithmica*, 55(1):113–133, 2009.

[10] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. 32nd Annual ACM Symposium on the Theory of Computing*, pages 80–86, 2000.

[11] F. Kuhn. Local multicoloring algorithms: Computing a nearly-optimal tdma schedule in constant time. In *STACS*, pages 613–624, 2009.

[12] F. Kuhn and T. Moscibroda. Distributed approximation of capacitated dominating sets. In *SPAA*, pages 161–170, 2007.

[13] F. Kuhn, T. Moscibroda, T. Nieberg, and R. Wattenhofer. Fast deterministic distributed maximal independent set computation on growth-bounded graphs. In *DISC*, pages 273–287, 2005.

[14] F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 980–989, 2006.

[15] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. 25th ACM Symposium on Principles of Distributed Computing*, pages 7–15, 2006.

[16] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986. Earlier version in STOC'85.

[17] S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. In *APPROX-RANDOM'06*, pages 475–486, 2006.

[18] A. Mayer, S. Naor, and L. Stockmeyer. Local computations on static and dynamic graphs. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems (ISTCS)*, 1995.

[19] D. S. Mitrinović. *Analytic inequalities.* Springer-Verlag, 1970.

[20] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24(6):1259–1277, 1995.

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

[22] K. Onak. *New Sublinear Methods in the Struggle Against Classical Problems.* PhD thesis, MIT, 2010.

[23] R. Otter. The multiplicative process. *Annals of mathematical statistics*, 20(2):206–224, 1949.

[24] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1–3):183–196, 2007.

[25] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science*, pages 223–238, 2011.

[26] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. Technical report, April 2011. `http://arxiv.org/abs/1104.1377`.

[27] M. E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, 39(7):2897–2926, 2010.

[28] S. Yekhanin. Locally decodable codes. In *6th International Computer Science Symposium in Russia*, pages 289–290, 2011.

[29] Y. Yoshida, Y. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proc. 41st Annual ACM Symposium on the Theory of Computing*, pages 225–234, 2009.