Converting Online Algorithms to Local Computation Algorithms

Yishay Mansour[1] [*], Aviad Rubinstein[1] [**], Shai Vardi[1] [***], and Ning Xie[2] [†]

[1] School of Computer Science, Tel Aviv University, Israel
[2] CSAIL, MIT, Cambridge MA 02139, USA

Abstract. We give a general method for converting online algorithms to local computation algorithms,[3] by selecting a random permutation of the input, and simulating running the online algorithm. We bound the number of steps of the algorithm using a *query tree*, which models the dependencies between queries. We improve previous analyses of query trees on graphs of bounded degree, and extend the analysis to the cases where the degrees are distributed binomially, and to a special case of bipartite graphs.

Using this method, we give a local computation algorithm for maximal matching in graphs of bounded degree, which runs in time and space $O(\log^3 n)$.

We also show how to convert a large family of load balancing algorithms (related to balls and bins problems) to local computation algorithms. This gives several local load balancing algorithms which achieve the same approximation ratios as the online algorithms, but run in $O(\log n)$ time and space.

Finally, we modify existing local computation algorithms for hypergraph 2-coloring and k-CNF and use our improved analysis to obtain better time and space bounds, of $O(\log^4 n)$, removing the dependency on the maximal degree of the graph from the exponent.

1 Introduction

1.1 Background

The classical computation model has a single processor which has access to a given input, and using an internal memory, computes the output. This is essentially the von Newmann architecture, which has been the driving force since the early days of computation. The class of polynomial time algorithms is widely accepted as the definition

---

[3] For a given input x, *local computation algorithms* support queries by a user to values of specified locations y_i in a legal output $y \in F(x)$.

of *efficiently computable* problems. Over the years many interesting variations of this basic model have been studied, focusing on different issues.

Online algorithms (see, e.g., [7]) introduce limitations in the time domain. An online algorithm needs to select actions based only on the history it observed, without access to future inputs that might influence its performance. Sublinear algorithms (e.g. [11, 15]) limit the space domain, by limiting the ability of the algorithm to observe the entire input, and still strive to derive global properties of it.

Local computation algorithms (LCAs) [16] are a variant of sublinear algorithms. The LCA model considers a computation problem which might have multiple admissible solutions, each consisting of multiple bits. The LCA can return queries regarding parts of the output, in a consistent way, and in poly-logarithmic time. For example, the input for an LCA for a job scheduling problem consists of the description of n jobs and m machines. The admissible solutions might be the allocations of jobs to machines such that the makespan is at most twice the optimal makespan. The LCA is queried with a job and returns its machine. The correctness property of the LCA guarantees that different query replies will be consistent with some admissible solution.

1.2 Our results

Bounds on query trees Suppose that we have an online algorithm where the reply to a query depends on a small number of previous queries. The reply to each of those previous queries depenends on a small number of other queries and so on. These dependencies can be used to model certain problems using *query trees* - trees which model the dependency of a given query on other queries.

Bounding the size of query trees is central to the analyses of our algorithms. We show that the size of the query tree is $O(\log n)$ w.h.p., where n is the number of vertices. d, the maximal degree of the graph appears (albeit exponentially) only in the constant. This answers in the affirmative the conjecture of [2]. Previously, Alon et al. [2] show that the expected size of the query tree is constant, and $O(\log^{d+1} n)$ w.h.p.[4] Our improvement is significant in removing the dependence on d from the exponent of the logarithm. We also show that when the degrees of the graph are distributed binomially, we can achieve the same bound on the size of the query tree. In addition, we show a trivial lower bound of $\Omega(\log n / \log \log n)$.

We use these results on query trees to obtain LCAs for several online problems - maximal matching in graphs of bounded degree and several load balancing problems. We also use the results to improve the previous algorithms for hypergraph 2-coloring and k-CNF.

Hypergraph 2-coloring We modify the algorithm of [2] for an LCA for hypergraph 2-coloring, and coupled with our improved analysis of query tree size, obtain an LCA which runs in time and space $O(\log^4 n)$, improving the previous result, an LCA which runs $O(\log^{d+1} n)$ time and space.

---

[4] Notice that bounding the expected size of the query tree is not enough for our applications, since in LCAs we need to bound the probability that *any* query fails.

k-CNF Building on the similarity between hypergraph 2-coloring and k-CNF, we apply our results on hypergraph 2-coloring to give an an LCA for k-CNF which runs in time and space $O(\log^4 n)$.

We use the query tree to transform online algorithms to LCAs. We simulate online algorithms as follows: a random permutation of the items is calculated, and then when an item is queried, we simulate the online algorithm. Fortunately, because of the nature of our graphs (the fact that the degree is bounded or distributed binomially), we show that in expectation, we will only need to query a constant number of nodes, and only $O(\log n)$ nodes w.h.p. We now state our results:

Maximal matching We simulate the greedy online algorithm for maximal matching, to derive an LCA for maximal matching which runs in time and space $O(\log^3 n)$.

Load Balancing We give several LCAs to load balancing problems which run in $O(\log n)$ time and space. Our techniques include extending the analysis of the query tree size to the case where the degrees are selected from a binomial distribution with expectation d, and further extending it to bipartite graphs which exhibit the characteristics of many balls and bins problems, specifically ones where each ball chooses d bins at random. We show how to convert a large class of the "power of d choices" online algorithms (see, e.g., [3, 6, 18]) to efficient LCAs.

1.3 Related work

Nguyen and Onak [13] focus on transforming classical approximation algorithms into constant-time algorithms that approximate the size of the optimal solution of problems such as vertex cover and maximum matching. They generate a random number $r \in [0, 1]$, called the rank, for each node. These ranks are used to bound the query tree size.

Rubinfeld et al. [16] show how to construct polylogarithmic time local computation algorithms to maximal independent set computations, scheduling radio network broadcasts, hypergraph coloring and satisfying k-SAT formulas. Their proof technique uses Beck's analysis in his algorithmic approach to the Lovász Local Lemma [4], and a reduction from distributed algorithms. Alon et al. [2], building on the technique of [13], showed how to extend several of the algorithms of [16] to perform in polylogarithmic space as well as time. They further observe that we do not actually need to assign each query a rank, we only need a random permutation of the queries. Furthermore, assuming the query tree is bounded by some k, the query to any node depends on at most k queries to other nodes, and so a k-wise independent random ordering suffices. They show that how to construct a $1/n^2$-almost k-wise independent random ordering[5] from a seed of length $O(k \log^2 n)$.

Recent developments in sublinear time algorithms for sparse graph and combinatorial optimization problems have led to new constant time algorithms for approximating

---

[5] A random ordering D_r is said to be ϵ-*almost k-wise independent* if the statistical distance between D_r and some k-wise independent random ordering by at most ϵ.

the size of a minimum vertex cover, maximal matching, maximum matching, minimum dominating set, and other problems (cf. [15, 11, 13, 20]), by randomly querying a constant number of vertices. A major difference between these algorithms and LCAs is that LCAs require that w.h.p., the output will be correct on any input, while optimization problems usually require a correct output only on *most* inputs. More importantly, LCAs reuire a consistent output for each query, rather than only approximating a given global property.

There is a vast literature on the topic of balls and bins and the power of d choices. ([3, 6, 9, 18]). For a survey on the power of d choices, we refer the reader to [12].

1.4 Organization of our paper

The rest of the paper is organized as follows: Some preliminaries and notations that we use throughout the paper appear in Section 2. In Section 3 we prove the upper bound of $O(\log n)$ on the size of the query tree in the case of bounded and binomially distributed degrees. In section 4, we use this analysis to give improved algorithms for hypergraph 2-coloring and k-CNF. In Section 5 we give an LCA for finding a maximal matching in graphs of bounded degree. Section 6 expands our query tree result to a special case of bipartite graphs; we use this bound for bipartite graph to convert online algorithms for balls and bins into LCAs for the same problems. The appendices provide detailed proofs of the lemmas in the analysis of the query tree size, in-depth discussions of the hypergraph 2-coloring and analogous k-CNF LCAs, and a lower bound to the query tree size.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. We denote by $N_G(v) = \{u \in V(G) : (u, v) \in E(G)\}$ the neighbors of vertex v, and by $deg_G(v)$ we denote the degree of v. When it is clear from the context, we omit the G in the subscript. Unless stated otherwise, all logarithms in this paper are to the base 2. We use $[n]$ to denote the set $\{1, \ldots, n\}$, where $n \geq 1$ is a natural number.

We present our model of local computation algorithms (LCAs): 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* for F is to find any $y \in F(x)$.

A $(t(n), s(n), \delta(n))$-*local computation algorithm* $\mathcal{A}$ is a (randomized) algorithm which solves a search problem for F for an input x of size n. However, the LCA $\mathcal{A}$ does not output a solution $y \in F(x)$, but rather implements query access to $y \in F(x)$. $\mathcal{A}$ receives a sequence of queries $i_1, \ldots, i_q$ and for any $q > 0$ satisfies the following: (1) after each query i_j it produces an output y_{i_j}, (2) With probability at least $1 - \delta(n)$ $\mathcal{A}$ is *consistent*, that is, the outputs $y_{i_1}, \ldots, y_{i_q}$ are substrings of some $y \in F(x)$. (3) $\mathcal{A}$ 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 $O(\log^c n)$ for some constant c.

Two important properties of LCAs are as follows. 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 $\mathcal{A}$. We say an LCA $\mathcal{A}$ is *parallelizable* if $\mathcal{A}$ supports parallel queries, that is $\mathcal{A}$ is able to answer multiple queries simultaneously so that all the answers are consistent.

3 Bounding the size of a random query tree

3.1 The problem and our main results

In online algorithms, queries arrive in some unknown order, and the reply to each query depends only on previous queries (but not on any future events). The simplest way to transform online algorithms to LCAs is to process the queries in the order in which they arrive. This, however, means that we have to store the replies to all previous queries, so that even if the time to compute each query is polylogarithmic, the overall space is linear in the number of queries. Furthermore, this means that the resulting LCA is not query-oblivious. The following solution can be applied to this problem ([13] and [2]): Each query is assigned a random number, $r(v) \in [0, 1]$, called its *rank*, and the queries are performed in ascending order of rank. Then, for each query x, a query tree can be constructed, to represent the queries that x depends on. If we can show that the query tree is small, we can conclude that each query does not depend on many other queries, and therefore a small number of queries are needed to be processed in order to reply to query x. We formalize this as follows:

Let $G = (V, E)$ be an undirected graph. The vertices of the graph represent queries, and the edges represent the dependencies between the queries. A real number $r(v) \in [0, 1]$ is assigned independently and uniformly at random to every vertex $v \in V$; we call $r(v)$ the *rank* of v. This models the random permutation of the vertices. Each vertex $v \in V$ holds an input $x(v) \in R$, where the range R is some finite set. The input is the content of the query associated with v. A randomized function F is defined inductively on the vertices of G such that $F(v)$ is a (deterministic) function of $x(v)$ as well as the values of F at the neighbors w of v for which $r(w) < r(v)$. F models the output of the online algorithm. We would like to upper bound the number of queries needed to vertices in the graph in order to compute $F(v_0)$ for any vertex $v_0 \in G$, namely, the time to simulate the output of query v_0 using the online algorithm.

To upper bound the number of queries to the graph, we turn to a simpler task of bounding the size of a certain d-regular tree, which is an upper bound on the number of queries. Consider 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 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 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_0, add all the children of v_0 whose ranks are smaller than that of v_0 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. Note that $|T|$ is an upper bound on the number of queries since each node in T has at least as many neighbors as that in G and if a node is connected to some previously queried nodes, this can only decrease the number of queries. Therefore the number of queries is bounded by the size of T. Our goal is to find an upper bound on $|T|$ which holds with high probability.

We improve the upper bound on the query tree of $O(\log^{d+1} N)$ given in [2] for the case when the degrees are bounded by a constant d and extend our new bound to the case that the degrees of G are binomially distributed, independently and identically with expectation d, i.e., $deg(v) \sim B(n, d/n)$.

Our main result in this section is bounding, with high probability, the size of the query tree T as follows. The complete proof is in Appendix A.

Lemma 1. *Let G be a graph whose vertex degrees are bounded by d or distributed independently and identically from the binomial distribution: $deg(v) \sim B(n, d/n)$. Then there exists a constant $C(d)$ which depends only on d, such that*

$$\Pr[|T| > C(d) \log n] < 1/n^2,$$

where the probability is taken over all the possible permutations $\pi \in \Pi$ of the vertices of G, and T is a random query tree in G under π.

3.2 Overview of the proof

Our proof consists of two parts. Following [2], we partition the query tree into levels. The first part of the proof is an upper bound on the size of a single (sub)tree on any level. For the bounded degree case, this was already proved in [2] (the result is restated as Lemma 3). We prove this for the binomially distributed degrees case in Appendix A. In both cases the bound is that with high probability each subtree is of size at most logarithmic in the size of the input.

The second part, which is a new ingredient of our proof, inductively upper bounds the number of vertices on each level, as the levels increase. For this to hold, it crucially depends on the fact that all subtrees are generated independently and that the probability of any subtree being large is exponentially small. The main idea is to show that although each subtree, in isolation, can reach a logarithmic size, their combination is not likely to be much larger. In the proof (in Appendix A), we use the distribution of the sizes of the subtrees, in order to bound the aggregate of multiple subtrees.

4 Hypergraph 2-coloring and k-CNF

We use the bound on the size of the query tree of graphs of bounded degree to improve the analysis of [2] for hypergraph 2-coloring. We also modify their algorithm slightly to further improve the algorithm's complexity. Due to lack of space, we only state our main theorem for hypergraph 2-coloring; the proof is in Appendix C.

Theorem 1. *Let H be a k-uniform hypergraph s.t. each hyperedge intersects at most d other hyperedges. Suppose that $k \geq 16 \log d + 19$.*
Then there exists an $(O(\log^4 n), O(\log^4 n), 1/n)$-local computation algorithm which, given H and any sequence of queries to the colors of vertices $(x_1, x_2, \ldots, x_s)$, with probability at least $1 - 1/n^2$, returns a consistent coloring for all x_i's which agrees with a 2-coloring of H. Moreover, the algorithm is query oblivious and parallelizable.

Due to the similarity between hypergraph 2-coloring and k-CNF, we also have the following theorem; the proof is in Appendix D.

Theorem 2. *Let H be a k-CNF formula with $k \geq 2$. Suppose that each clause intersects no more than d other clauses, and furthermore suppose that $k \geq 16 \log d + 19$.*
Then there exists a $(O(\log^4 n), O(\log^4 n), 1/n)$-local computation algorithm which, given a formula H and any sequence of queries to the truth assignments of variables $(x_1, x_2, \ldots, x_s)$, with probability at least $1 - 1/n^2$, returns a consistent truth assignment for all x_i's which agrees with some satisfying assignment of the k-CNF formula H. Moreover, the algorithm is query oblivious and parallelizable.

5 Maximal matching

We condsider the problem of *maximal matching in a bounded-degree graph*. We are given a graph $G = (V, E)$, where the maximal degree is bounded by some constant d, and we need to find a maximal matching. A matching is, a set of edges with the property that no two edges share a common vertex. The matching is maximal if no other edge can be added to it without violating the matching property.

Assume the online scenario in which the edges arrive in some unknown order. The following greedy online algorithm can be used to calculate a maximal matching: When an edge e arrives, we check whether e is already in the matching. If it is not, we check if any of the neighboring edges are in the matching. If none of them is, we add e to the matching. Otherwise, e is not in the matching.

We turn to the local computation variation of this problem. We would like to query, for some edge $e \in E$, whether e is part of some maximal matching. (Recall that all replies must be consistent with some maximal matching).

We use the technique of [2] to produce an almost $O(\log n)$-wise independent random ordering on the edges, using a seed length of $O(\log^3 n)$.[6] When an edge e is queried, we use a BFS (on the edges) to build a DAG rooted at e. We then use the greedy online algorithm on the edges of the DAG (examining the edges with respect to the ordering), and see whether e can be added to the matching.

From Lemma 1, we derive the following theorem.

Theorem 3. *Let $G = (V, E)$ be an undirected graph with n vertices and maximum degree d. Then there is an $(O(\log^3 n), O(\log^3 n), 1/n)$ - local computation algorithm which, on input an edge e, decides if e is in a maximal matching. Moreover, the algorithm gives a consistent maximal matching for every edge in G.*

---

[6] Since the query tree is of size $O(\log n)$ w.h.p., we don't need a complete ordering on the vertices; an almost $O(\log n)$-wise independent ordering suffices.

6 Local load balancing

We consider a general "power of d choices" online algorithm for load balancing. In this setting there are n balls that arrive in an online manner, and m bins. Each ball selects a random subset of d bins, and queries these bins. (Usually the query is simply the current load of the bin.) Given this information, the ball is assigned to one of the d bins (usually to the least loaded bin). We denote by $\mathcal{LB}$ such a generic algorithm (with a decision rule which can depend in an arbitrary way on the d bins that the ball is assigned to). Our main goal is to simulate such a generic algorithm.

The load balancing problem can be represented by a bipartite graph $G = (\{V, U\}, E)$, where the balls are represented by the vertices V and the bins by the vertices U. The random selection of a bin $u \in U$ by a ball $v \in V$ is represented by an edge. By definition, each ball $v \in V$ has degree d. Since there are random choices in the algorithm $\mathcal{LB}$ we need to specify what we mean by a simulation. For this reason we define the input to be the following: a graph $G = (\{V, U\}, E)$, where $|V| = n$, $|U| = m$, and $n = cm$ for some constant $c \geq 1$. We also allocate a rank $r(u) \in [0, 1]$ to every $u \in U$. This rank represents the ball's arrival time: if $r(v) < r(u)$ then vertex v arrived before vertex u. Furthermore, all vertices can have an input value $x(w)$. (This value represents some information about the node, e.g., the weight of a ball.) Given this input, the algorithm $\mathcal{LB}$ is deterministic, since the arrival sequence is determined by the ranks, and the random choices of the balls appear as edges in the graph. Therefore by a simulation we will mean that given the above input, we generate the same allocation as $\mathcal{LB}$.

We consider the following stochastic process: Every vertex $v \in V$ uniformly and independently at random chooses d vertices in U. Notice that from the point of view of the bins, the number of balls which chose them is distributed binomially with $X \sim B(n, d/m)$. Let X_v and X_u be the random variables for the number of neighbors of vertices $v \in V$ and $u \in U$ respectively. By definition, $X_v = d$, since all balls have d neighbors, and hence each X_u is independent of all X_v's. However, there is a dependence between the X_u's (the number of balls connected to different bins). Fortunately this is a classical example where the random variables are negatively dependent (see e.g. [9]). [7]

Recall that in Section 3, we assumed that the degrees of the vertices in the graph were independent. We would like to prove an $O(\log n)$ upper bound on the query tree T for our bipartite graph. As we cannot use the theorems of Section 3 directly, we show that the query tree is smaller than another query tree which meets the conditions of our theorems.

The query tree for the binomial graph is constructed as follows: a root $v_0 \in V$ is selected for the tree. (v_0 is the ball whose bin assignment we are interested in determining.) Label the vertices at depth j in the tree by W_j. Clearly, $W_0 = \{v_0\}$. At each depth d, we add vertices one at a time to the tree, from left to right, until the depth is "full" and then we move to the next depth. Note that at odd depths $(2j + 1)$ we add bin vertices and at even depths $(2j)$ we add ball vertices.

---

[7] We remind the reader that two random variables X_1 and X_2 are negatively dependent if $\Pr[X_1 > x | X_2 = a] < \Pr[X_1 > x | X_2 = b]$, for $a > b$ and vice-versa.

Specifically, at odd depths $(2j + 1)$ we add, for each $v \in W_{2j}$ its d neighbors $u \in N(v)$ as children, and mark each by u.[8] At even depths $(2j)$ we add for each node marked by $u \in W_{2j-1}$ all its (ball) neighbors $v \in N(u)$ such that $r(v) < r(parent(u))$, if they have not already been added to the tree. Namely, all the balls that are assigned to u by time

A leaf is a node marked by a bin u_ℓ for whom all neighboring balls $v \in N(u_\ell) - \{parent(u_\ell)\}$ have a rank larger than its parent, i.e., $r(v) > r(parent(u_\ell))$. Namely, $parent(u_\ell)$ is the first ball to be assigned to bin u_ℓ. This construction defines a stochastic process $F = \{F_t\}$, where F_t is (a random variable for) the size of T at time t. (We start at $t = 0$ and t increases by 1 for every vertex we add to the tree).

We now present our main lemma for bipartite graphs; the proof is in Appendix B.

Lemma 2. *Let $G = (\{V, U\}, E)$ be a bipartite graph, $|V| = n$ and $|U| = m$ and $n = cm$ for some constant $c \geq 1$, such that for each vertex $v \in V$ there are d edges chosen independently and at random between v and U. Then there is a constant $C(d)$ which depends only on d such that*

$$\Pr[|T| < C(d) \log n] > 1 - 1/n^2,$$

where the probability is taken over all of the possible permutations $\pi \in \Pi$ of the vertices of G, and T is a random query tree in G under π.

The following theorem states our basic simulation result.

Theorem 4. *Consider a generic online algorithm $\mathcal{LB}$ which requires constant time per query, for n balls and m bins, where $n = cm$ for some constant $c > 0$. There exists an $(O(\log n), O(\log n), 1/n)$-local computation algorithm which, on query of a (ball) vertex $v \in V$, allocates v a (bin) vertex $u \in U$, such that the resulting allocation is identical to that of $\mathcal{LB}$ with probability at least $1 - 1/n$.*

Proof. Let $K = C(d) \log |U|$ for some constant $C(d)$ depending only on d. K is the upper bound given in Lemma 2. (In the following we make no attempt to provide the exact values for $C(d)$ or K.)

We now describe our $(O(\log n), O(\log n), 1/n)$-local computation algorithm for $\mathcal{LB}$. A query to the algorithm is a (ball) vertex $v_0 \in V$ and the algorithm will chose a (bin) vertex from the d (bin) vertices connected to v_0.

We first build a query tree as follows: Let v_0 be the root of the tree. For every $u \in N(u_0)$, add to the tree the neighbors of u, $v \in V$ such that $r(v) < r(v_0)$. Continue inductively until either K nodes have been added to the random query tree or no more nodes can be added to it. If K nodes have been added to the query tree, this is a failure event, and assign to v_0 a random bin in $N(v_0)$. From Lemma 2, this happens with probability at most $1/n^2$, and so the probability that some failure event will occur is at most $1/n$. Otherwise, perform $\mathcal{LB}$ on all of the vertices in the tree, in order of addition to the tree, and output the bin to which ball v_0 is assigned to by $\mathcal{LB}$. $\square$

---

[8] A bin can appear several times in the tree. It appears as different nodes, but they are all marked so that we know it is the same bin. Recall that we assume that all nodes are unique, as this assumption can only increase the size of the tree.

A reduction from various load balancing algorithms gives us the following corollaries to Theorem 4.

Corollary 1. *(Using [6]) Suppose we wish to allocate m balls into n bins of uniform capacity, $m \geq n$, where each ball chooses d bins independently and uniformly at random. There exists a $(\log n, \log n, 1/n)$ LCA which allocates the balls in such a way that the load of the most loaded bin is $m/n + O(\log \log n / \log d)$ w.h.p.*

Corollary 2. *(Using [19]) Suppose we wish to allocate n balls into n bins of uniform capacity, where each ball chooses d bins independently at random, one from each of d groups of almost equal size $\theta(\frac{n}{d})$. There exists a $(\log n, \log n, 1/n)$ LCA, which allocates the balls in such a way that the load of the most loaded bin is $\ln \ln n / (d-1) \ln 2 + O(1)$ w.h.p. [9]*

Corollary 3. *(Using [5]) Suppose we wish to allocate m balls into $n \leq m$ bins, where each bin i has a capacity c_i, and $\sum_i c_i = m$. Each ball chooses d bins at random with probability proportional to their capacities. There exists a $(\log n, \log n, 1/n)$ LCA which allocates the balls in such a way that the load of the most loaded bin is $2 \log \log n + O(1)$ w.h.p.*

Corollary 4. *(Using [5]) Suppose we wish to allocate m balls into $n \leq m$ bins, where each bin i has a capacity c_i, and $\sum_i c_i = m$. Assume that the size of a large bin is at least $rn \log n$, for large enough r. Suppose we have s small bins with total capacity m_s, and that $m_s = O((n \log n)^{2/3})$. There exists a $(\log n, \log n, 1/n)$ LCA which allocates the balls in such a way that the expected maximum load is less than 5.*

Corollary 5. *(Using [8]) Suppose we have n bins, each represented by one point on a circle, and n balls are to be allocated to the bins. Assume each ball needs to choose $d \geq 2$ points on the circle, and is associated with the bins closest to these points. There exists a $(\log n, \log n, 1/n)$ LCA which allocates the balls in such a way that the load of the most loaded bin is $\ln \ln n / \ln d + O(1)$ w.h.p.*

6.1 Random ordering

In the above we assume that we are given a random ranking for each ball. If we are not given such random rankings (in fact, a random permutation of the vertices in U will also suffice), we can generate a random ordering of the balls. Specifically, since w.h.p. the size of the random query is $O(\log n)$, an $O(\log n)$-*wise independent random ordering*[10] suffices for our local computation purpose. Using the construction in [2] of $1/n^2$-almost $O(\log n)$-wise independent random ordering over the vertices in U which uses space $O(\log^3 n)$, we obtain $(O(\log^3 n), O(\log^3 n), 1/n)$-local computation algorithms for balls and bins.

---

[9] In fact, in this setting the tighter bound is $\frac{\ln \ln n}{d \ln \phi_d} + O(1)$, where ϕ_d is the ratio of the d-step Fibonacci sequence, i.e. $\phi_d = \lim_{k \to \infty} \sqrt[k]{F_d(k)}$, where for $k < 0$, $F_d(k) = 0$, $F_d(1) = 1$, and for $k \geq 1$ $F_d(k) = \sum_{i=1}^{d} F_d(k-i)$

[10] See [2] for the formal definitions of k-wise independent random ordering and almost k-wise independent random ordering.

References

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

[2] N. Alon, R. Rubinfeld, S. Vardi, and N. Xie. Space-efficient local computation algorithms. In *Proc. 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 1132–1139, 2012.

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

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

[5] P. Berenbrink, A. Brinkmann, T. Friedetzky, and L. Nagel. Balls into non-uniform bins. In *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10. IEEE, 2010.

[6] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

[7] A. Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[8] John W. Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple load balancing for distributed hash tables. In *Proc. of Intl. Workshop on Peer-to-Peer Systems(IPTPS)*, pages 80–87, 2003.

[9] D. Dubhashi and D. Ranjan. Balls and bins: A study in negative dependence. *Random Structures and Algorithms*, 13:99–124, 1996.

[10] T. Harris. *The Theory of Branching Processes*. Springer, 1963.

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

[12] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: A survey of techniques and results. In *Handbook of Randomized Computing, Vol. I, edited by P. Pardalos, S. Rajasekaran, J. Reif, and J. Rolim*, pages 255–312. Norwell, MA: Kluwer Academic Publishers, 2001.

[13] 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.

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

[15] 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), 2007.

[16] 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.

[17] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms, 2011.

[18] K. Talwar and U. Wieder. Balanced allocations: the weighted case. In *Proc. 39th Annual ACM Symposium on the Theory of Computing*, pages 256–265, 2007.

[19] Berthold Vöcking. How asymmetry helps load balancing. *J. ACM*, 50:568–589, July 2003.

[20] 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.

A Bounding the subtree size (Proof of Lemma 1)

As in [2], we partition the query tree into levels and then upper bound the probability that a subtree is larger than a given threshold. Let $L > 1$ be a function of d to be determined later. First, we partition the interval [0,1] into L sub-intervals: $I_i = (1 - \frac{i}{L+1}, 1 - \frac{i-1}{L+1}]$, for $i = 1, 2, \cdots, L$ and $I_{L+1} = [0, \frac{1}{L+1}]$. We refer to interval I_i as *level* i. A vertex $v \in T$ is said to be on level i if $r(v) \in I_i$. We consider the worst case, in which $r(v_0) \in I_1$. In this case, the vertices on level 1 form a tree T_1 rooted at v_0. Denote the number of (sub)trees on level i by t_i. The vertices on level 2 will form a set of trees $\{T_2^{(1)}, \cdots, T_2^{(t_2)}\}$, where the total number of subtrees is at most the sum of the children of all the vertices in T_1 (we only have inequality because some of the children of the vertices of T_1 may be assigned to levels 3 and above.) The vertices on level $i > 1$ form a set of subtrees $\{T_i^{(1)}, \cdots T_i^{(t_i)}\}$. Note that all these subtrees $\{T_i^{(j)}\}$ are generated independently by the same stochastic process, as the ranks of all nodes in $\mathcal{T}$ are i.i.d. random variables. In the following analysis, we will set $L = 3d$.

For the bounded degree case, bounding the size of the subtree follows from [2], and is based on the following lemma.

Lemma 3 ([2]). *For $L = d + 1$, for any $1 \leq i \leq L$ and any $1 \leq j \leq t_i$, $\Pr[|T_i^{(j)}| \geq n] \leq \sum_{i=n}^{\infty} 2^{-ci} \leq 2^{-\Omega(n)}$, for all $n \geq \beta$, where β is some constant.*

This immediately establishes the following lemma, since the worse case is $L = d + 1$.

Lemma 4. *Let $L \geq d + 1$ be a fixed integer and let $\mathcal{T}$ be the d-regular infinite query tree. Then for any $1 \leq i \leq L$ and $1 \leq j \leq t_i$, $\Pr[|T_i^{(j)}| \geq n] \leq \sum_{i=n}^{\infty} 2^{-ci} \leq 2^{-\Omega(n)}$, for all $n \geq \beta$, where β is some constant. In particular, there is an absolute constant c_0 depending on d only such that for all $n \geq 1$,*

$$\Pr[|T_i^{(j)}| \geq n] \leq e^{-c_0 n}.$$

We are interested in bounding the subtree size also in the case that the degrees are not a constant d, but rather selected independently and identically from a binomial distribution with mean d.

Lemma 5. *Let $\mathcal{T}$ be a tree with vertex degree distributed i.i.d. binomially with $\deg(v) \sim B(n, d/n)$. For any $1 \leq i \leq L$ and any $1 \leq j \leq t_i$, $\Pr[|T_i^{(j)}| \geq n] \leq \sum_{i=n}^{\infty} 2^{-ci} \leq 2^{-\Omega(n)}$, for $n \geq \beta$, for some constant $\beta > 0$.*

Proof. The proof of Lemma 5 is similar to that of Lemma 3 in [2]; we employ the theory of Galton-Watson processes. For a good introduction to Galton-Watson branching processes see e.g. [10].

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 offsprings in the ith generation. Clearly $Z_0 = 1$ and $\{Z_i : i = 0, 1, \ldots\}$ form a Markov chain. Let $m := \mathbb{E}[Z_1] = \sum_k k p_k$ be the expected number of children of any individual. Let

$Z = Z_0 + Z_1 + \cdots$ be the sum of all offsprings in all generations of the Galton-Watson process. The following result of Otter is useful in bounding the probability that Z is large.

Theorem 5 ([14]). *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*

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

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

We prove Lemma 5 for the case of tree T_1 – the proof actually applies to all subtrees $T_i^{(j)}$. Recall that T_1 is constructed inductively as follows: for $v \in N(v_0)$ in $\mathcal{T}$, we add v to T_1 if $r(v) < r(v_0)$ and $r(v) \in I_1$. Then for each v in T_1, we add the neighbors $w \in N(v)$ 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 vertex 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 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 vertex of v in $\mathcal{T}$, then we add w to T_1' as long as $r(w) \in I_1$. We 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 vertex in T is independently and uniformly distributed in $[0, 1]$. We take vertex v to be the parent node. Since $|I_1| = 1/L$, then for any vertex $u \in V(G)$, $u \neq v$, the probability that u is a child node of v in T_1' is

$$
d/n \cdot 1/L = d/nL,
$$

as the random process that connects w to v and the random process that generates the rank of w are independent (each edge is chosen with probability d/n, and the probability that $r(w)$ is in $I_v = 1/L$). It follows that we have a binomial distribution for the number of child nodes of v in T_1':

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

where $q := d/nL$ is the probability that a child vertex in T appears in T_1' when its parent vertex is in T_1'. Note that the expected number of children of a vertex in T_1' is

$nq = d/L < 1$, so from the classical result on the extinction probability of Galton-Watson processes (see e.g. [10]), the tree T_1' is finite with probability one.

The generating function of **p** is

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

as the probability function $\{p_k\}$ obeys the binomial distribution $p_k = \Pr[X = k]$ where $X \sim B(n, q)$. In addition, the convergence radius of f is $\rho = \infty$ since $\{p_k\}$ has only a finite number of non-zero terms.

$$f'(s) = nq(1 - q + qs)^{n-1}$$

Solving the equation $af'(a) = f(a)$ yields $anq(1 - q + qa)^{n-1} = (1 - q + qa)^n$ and hence $anq = 1 - q + qa$. Consequently, solving for a gives

$$a = \frac{1 - q}{q(n - 1)}$$
$$= \frac{1 - d/nL}{d(n - 1)/nL}$$
$$= \frac{nL - d}{d(n - 1)}$$
$$= \frac{3n - 1}{n - 1}.$$

We can lower bound α as

$$\alpha = \frac{a}{f(a)} = \frac{1}{f'(a)}$$
$$= \frac{L}{d(1 - q + q\frac{3n-1}{n-1})^{n-1}}$$
$$= \frac{3}{\left(1 + \frac{2}{3(n-1)}\right)^{n-1}}$$
$$\geq \frac{3}{e^{2/3}} > 1.$$

Finally we calculate $f''(a)$:

$$f''(a) = q^2 n(n - 1)(1 - q + qa)^{n-2}$$
$$= \frac{d^2 n(n - 1)}{n^2 L^2} \left(1 - \frac{d}{nL} + \frac{d}{nL} \cdot \frac{3n - 1}{n - 1}\right)^{n-2}$$
$$= \frac{n - 1}{9n} \left(1 + \frac{2}{3(n - 1)}\right)^{n-2}$$
$$= \Theta(1),$$

therefore $\frac{a}{\alpha f''(a)}$ is a bounded constant.

Now applying Theorem 5 to the Galton-Watson process which generates T'_1 (note that $t = 1$ in our case) gives that, there exists a constant n_0 such that for $n > n_0$, $\Pr[|T'_1| = n] \leq 2^{-cn}$ for some constant $c > 0$. It follows that $\Pr[|T'_1| \geq n] \leq \sum_{i=n}^{\infty} 2^{-ci} \leq 2^{-\Omega(n)}$ for all $n > n_0$. Hence for all large enough n, with probability at least $1 - 1/n^3$, $|T_1| \leq |T'_1| = O(\log n)$. $\qquad\square$

The following corollary stems directly from Lemmas 4 and 5:

Corollary 6. *Let $\mathcal{T}$ be any infinite d-regular query tree or tree with vertex degree distributed i.i.d. binomially with $deg(v) \sim B(n, d/n)$. For any $1 \leq i \leq L$ and any $1 \leq j \leq t_i$, with probability at least $1 - 1/n^3$, $|T_i^{(j)}| = O(\log n)$.*

A.1 Bounding the increase in number of vertices as we increase the level

From Corollary 6 we know that the size of any subtree, in particular $|T_1|$, is bounded by $O(\log n)$ with probability at least $1 - 1/n^3$ in both the degree d and the binomial degree cases. Our next step in proving Lemma 1 is to show that, as we increase the levels, the size of the tree does not increase by more than a constant factor for each level. That is, there exists an absolute constant η depending on d only such that if the number of vertices on level k is at most $|T_k|$, then the number of vertices on level $k+1$, $|T_{k+1}|$ satisfies $|T_{k+1}| \leq \eta \sum_{i=1}^{k} |T_i| + O(\log n) \leq 2\eta|T_k| + O(\log n)$. Since there are L levels in total, this implies that the number of vertices on all L levels is at most $O((2\eta)^L \log n) = O(\log n)$.

The following lemma establishes our inductive step.

Lemma 6. *For any infinite query tree $\mathcal{T}$ with constant bounded degree d (or degrees i.i.d. $\sim B(n, d/n)$), for any $1 \leq i < L$, there exist constants $\eta_1 > 0$ and $\eta_2 > 0$ s.t. if $\sum_{j=1}^{t_i} |T_i^{(j)}| \leq \eta_1 \log n$ then $\Pr[\sum_{j=1}^{t_{i+1}} |T_{i+1}^{(j)}| \geq \eta_1 \eta_2 \log n] < 1/n^2$ for all $n > \beta$, for some $\beta > 0$.*

Proof. Denote the number of vertices on level k by Z_k and let $Y_k = \sum_{i=1}^{k} Z_i$. Assume that each vertex i on level $\leq k$ is the root of a tree of size z_i on level $k + 1$. Notice that $Z_{k+1} = \sum_{i=1}^{Y_k} z_i$.

By Lemma 4 (or Lemma 5), there are absolute constants c_0 and β depending on d only such that for any subtree $T_k^{(i)}$ on level k and any $n > \beta$, $\Pr[|T_k^{(i)}| = n] \leq e^{-c_0 n}$. Therefore, given $(z_1, \ldots, z_{Y_k})$, the probability of the forest on level $k + 1$ consisting of exactly trees of size $(z_1, \ldots, z_{Y_k})$ is at most $\prod_{i=1}^{Y_k} e^{-c_0(z_i - \beta)} = e^{-c_0(Z_{k+1} - \beta Y_k)}$.

Notice that, given Y_k (the number of nodes up to level k), there are at most $\binom{Z_{k+1} + Y_k - 1}{Y_k - 1} < \binom{Z_{k+1} + Y_k}{Y_k}$ vectors $(z_1, \ldots, z_{Y_k})$ that can realize Z_{k+1}.[11]

---

[11] This can be viewed as Y_{k-1} separators of $Z_{k=1}$ items.

We want to bound the probability that $Z_{k+1} = \eta Y_k$ for some (large enough) constant $\eta > 0$. We can bound this as follows:

$$\Pr[|T_{k+1}| = Z_{k+1}] < \binom{Z_{k+1} + Y_k}{Y_k} e^{-c_0(Z_{k+1} - \beta Y_k)}$$

$$< \left(\frac{e \cdot (Z_{k+1} + Y_k)}{Y_k} \right)^{Y_k} e^{-c_0(Z_{k+1} - \beta Y_k)}$$

$$= (e(1+\eta))^{Y_k} e^{-c_0(\eta - \beta) Y_k}$$

$$= e^{Y_k(-c_0(\eta - \beta) + \ln(\eta+1) + 1)}$$

$$\leq e^{-c_0 \eta Y_k / 2},$$

for all large enough η.

It follows that there is some absolute constant c' which depends on d only such that $\Pr[|T_{k+1}| \geq \eta Y_k] \leq e^{-c' \eta Y_k}$. That is, if $\eta Y_k = \Omega(\log n)$, the probability that $|T_{k+1}| \geq \eta Y_k$ is at most $1/n^3$. Adding the vertices on all L levels and applying the union bound, we conclude that with probability at most $1/n^2$, the size of T is at most $O(\log n)$. $\square$

B Bounding the size of the query tree in the bipartite case (Proof of Lemma 2)

To prove Lemma 2, we look at another stochastic process F', which constructs a tree T': we start with a root v_0'. Label the vertices at depth j in the tree by W_j'. Assign every vertex y that is added to the tree a rank $r(y) \in [0, 1]$ independently and uniformly at random. Similarly to T, $W_0' = \{v_0'\}$. At odd depths $(2j + 1)$ we add to each $v' \in W_{2j}'$, D children (from left to right). At even depths $(2j)$ we add to each node $u' \in W_{2j-1}'$, $X_{u'}'$ children, where $X_{u'}' \sim B(n, 2D/m)$ and the $X_{u'}'$ of different nodes are i.i.d. Of the nodes added in this level, we remove all those vertices y' for which $r(y') > r(parent(parent(y')))$.

Importantly, the neighbor distributions of the vertices in the tree are independent of each other. If at any point T' has "more than half the bins", i.e., the sum of nodes on odd levels is at least $m/2$, we add $n + m$ bin children of rank 0 to some even-level node in the tree.

Given a tree T we define $squash(T)$ to be the tree T with the odd levels deleted, and a node v in level $2j$ is connected to node v' in level $2j + 2$ if $v = parent(parent(v'))$.

Lemma 7. *There is a constant $C(d)$ which depends only on D such that for all large enough n, $\Pr[|squash(T')| > C(d) \log n] < 1/n^2$.*

Because $D \cdot |squash(T')| \geq |T'| \geq |squash(T')|$, we immediately get the following corollary:

Corollary 7. *There is a constant $C(d)$ which depends only on d such that for all large enough n, $\Pr[|T'| > C(d) \log n] < 1/n^2$.*

We first make the following claim:

Claim. $squash(T')$ has vertex degree distributed i.i.d. binomially with $deg(v) \sim B(Dn, 2d/m)$.

Proof. Each $v \in W'_{2j}$ has d children, each with degree distributed binomially $\sim B(n, 2d/m)$. For any independent r.v.'s $Y_1, Y_2, \cdots$ where $\forall i > 0$, $Y_i \sim B(n, p)$, we know that $\sum_{i=1}^{q} Y_i \sim B(qn, p)$. The Claim follows. $\qquad\square$

We can now turn to the proof of Lemma 7:

Proof. As long as $|squash(T')| < m/2$, the proof of the lemma follows the proof of Lemma 1 with slight modifications to constants and will therefore be omitted.

We notice that $|squash(T')| < m/2$ w.p. at least $1 - 1/n^2$: the proof of Lemma 1 is inductive - we show that at level I_1, the size of the subtree is at most $O(\log n)$, and then bound the increase in tree size as we move to the next level. By the level I_L, $|squash(T')| = O(\log n)$ w.p. at least $1 - 1/n^2$. Therefore it follows that $|squash(T')| < m/2$ w.p. at least $1 - 1/n^2$. $\qquad\square$

Before we can complete the proof of Lemma 2, we need to define the notion of first order stochastic dominance:

Definition 1 (First order stochastic dominance). *We say a random variable X first order stochastically dominates (dominates for short) a random variable Y if $\Pr[X \geq a] \geq \Pr[Y \geq a]$ for all a and $\Pr[X \geq a] > \Pr[Y \geq a]$ for some a. If X dominates Y, then we write $X \geq Y$.*

Lemma 8. *For every t, F'_t first-order stochastically dominates F_t.*

Proof. Assume we add a (bin) vertex $u \in U$ to T at time t, the random variable for the number of u's neighbors is negatively dependent on all other X_w, $w \in T_t \cap U$. We label this variable $\bar{X} = X_u | \{X_w\}$, $w \in T_t$.

We first show that $F'_t \geq F_t$ when T has less than $m/2$ bins, and then show that $F'_t \geq F_t$ when T' has more than $m/2$ bins. (It is easy to see why this is enough).
Assume $|T_t \cap U| \leq m/2$. X_u is dependent on at most $m/2$ other random variables, X_w. Because the dependency is negative, X_u is maximized when $\forall w, X_w = 0$. Therefore, in the worst case, X_u is dependent on $m/2$ bins with 0 children. If $m/2$ bins have 0 children, all edges in G must be distributed between the remaining bins. Therefore $\bar{X} \leq X'_u$, where $X'_u \sim B(n, 2d/m)$.
When T' has more than $m/2$ bins, by the construction of F'_t, it has more than $m + n$ vertices, and so F'_t trivially dominates F_t. $\qquad\square$

Combining Corollary 7 and Lemma 8 completes the proof of Lemma 2.

C Hypergraph two-coloring

Recall that a *hypergraph* H is a pair $H = (V, E)$ where V is a finite set whose elements are called *nodes* or *vertices*, and E is a family of non-empty subsets of V, called *hyperedges*. A hypergraph is called k-*uniform* if each of its hyperedges contains precisely k vertices. A *two-coloring* of a hypergraph H is a mapping $\mathbf{c} : V \to \{\text{red, blue}\}$ such that no hyperedge in E is monochromatic. If such a coloring exists, then we say H is *two-colorable*. We assume that each hyperedge in H intersects at most d other hyperedges. Let n be the number of hyperedges in H. Here we think of k and d as fixed constants and all asymptotic forms are with respect to n. By the Lovász Local Lemma, when $e(d+1) \le 2^{k-1}$, the hypergraph H is two-colorable (e.g. [1]).

Following [17], we let m be the total number of vertices in H. Note that $m \le kn$, so $m = O(n)$. For any vertex $x \in V$, we use $\mathcal{E}(x)$ to denote the set of hyperedges x belongs to. For any hypergraph $H = (V, E)$, we define a *vertex-hyperedge incidence matrix* $\mathcal{M} \in \{0, 1\}^{m \times n}$ so that, for every vertex x and every hyperedge e, $\mathcal{M}_{x,e} = 1$ if and only if $e \in \mathcal{E}(x)$. Because we assume both k and d are constants, the incidence matrix $\mathcal{M}$ is necessarily very sparse. Therefore, we further assume that the matrix $\mathcal{M}$ is implemented via linked lists for each row (that is, vertex x) and each column (that is, hyperedge e).

Let G be the *dependency graph* of the hyperedges in H. That is, the vertices of the undirected graph G are the n hyperedges of H and a hyperedge E_i is connected to another hyperedge E_j in G if $E_i \cap E_j \ne \emptyset$. It is easy to see that if the input hypergraph is given in the above described representation, then we can find all the neighbors of any hyperedge E_i in the dependency graph G (there are at most d of them) in constant time (which depends on k and d).

A natural question to ask is: Given a two-colorable hypergraph H, and a vertex v, can we *quickly* compute the coloring of v? Alon et al. gave ([2]) a $\text{polylog}(n)$-time and space LCA based on Alon's 3-phase parallel hypergraph coloring algorithm ([1]), where the exponent of the logarithm depends on d. We get rid of the dependence on d (in the exponent of the logarithm) using the improved analysis of the query tree in section 3, together with a modified 4-phase coloring algorithm.

Our main result in this section is, given a two-colorable hypergraph H whose two-coloring scheme is guaranteed by the Lovász Local Lemma (with slightly weaker parameters), we give a $(O(\log^4 n), O(\log^4 n), 1/n)$ - local computation algorithm. We restate our main theorem:

Theorem 6. *Let H be a k-uniform hypergraph s.t. each hyperedge intersects at most d other hyperedges. Suppose that $k \ge 16 \log d + 19$.*
Then there exists an $(O(\log^4 n), O(\log^4 n), 1/n)$-local computation algorithm which, given H and any sequence of queries to the colors of vertices $(x_1, x_2, \ldots, x_s)$, with probability at least $1 - 1/n^2$, returns a consistent coloring for all x_i's which agrees with a 2-coloring of H. Moreover, the algorithm is query oblivious and parallelizable.

In fact, we only need:

$$k \ge 3\lceil \log 16d(d-1)^3(d+1) \rceil + \lceil \log 2e(d+1) \rceil$$

Throughout the following analysis, we set: $k_1 = k$, and

$$k_i = k_{i-1} - \lceil \log 16d(d-1)^3(d+1) \rceil$$

Notice that the theorem's premise simply implies that $2^{k_4-1} \geq e(d+1)$, as required by the Lovász Local Lemma.

C.1 The general phase - random coloring

In each phase we begin with subsets V_i and E_i of V and E, such that each edge contains at least k_i vertices. We sequentially assign colors at random to the vertices, as long as every monochromatic edge has at least than k_{i+1} uncolored vertices. Once the phase is over we do not change this assignment.

If an edge has all of its vertices besides k_{i+1} colored in one color, it is labeled *dangerous*. All the uncolored vertices in a dangerous edge are labeled *saved* and we do not color them in this phase. We proceed until all vertices in V_i are either red, blue, or saved. Let the *survived* hyperedges be all the edges that do not contain both red and blue vertices. Each survived edge contains some vertices colored in one color, and at least k_{i+1} saved vertices.

Let S_i be the set of survived edges after a random coloring in Phase i, and consider $G|_{S_i}$, the restriction of G to S_i The probability that $G|_{S_i}$ contains a connected component of size d^3u at most $|V_i|2^{-u}$ ([1]). In particular, after repeating the random coloring procedure t_i times, there is no connected component of size greater than d^3u with probability

$$\left(|V_i|2^{-u} \right)^{t_i}$$

If vertex x has been assigned a color, we can simply return it. Otherwise, if it is a saved vertex we let $C_i(x)$ be the connected component containing x in $G|_{S_i}$. Finally, since the coloring of $C_i(x)$ is independent of all other uncolored vertices, we can restrict ourselves to $E_{i+1} = C_i(x)$ in the next phase.

C.2 Phase 1: partial random coloring

In the first phase we begin with the whole hypergraph, i.e. $V_1 = V$, $E_1 = E$, and $k_1 = k$. Thus, we cannot even assign a random coloring to all the vertices in sublinear complexity. Instead, similarly to the previous sections, we randomly order the vertices of the hypergraph and use a query tree to randomly assign colors to all the vertices that arrive before x and may influence it. Note that this means that we can randomly assign the colors only once.

If x is a saved vertex, we must compute $E_2 = C_1(x)$, the connected component containing x in $G|_{S_1}$. Notice that the size $C_1(x)$ is bounded w.h.p.

$$\Pr \left[|C_1(x)| > 4d^3 \log n \right] < n2^{-4\log n} = n^{-3}$$

In order to compute $C_1(x)$, we run BFS on $G|_{S_1}$. Whenever we reach a new node, we must first randomly assign colors to the vertices in its query tree, like we did for x's query tree. Since (w.h.p.) there are at most $O(\log n)$ edges in $C_1(x)$, we query for trees
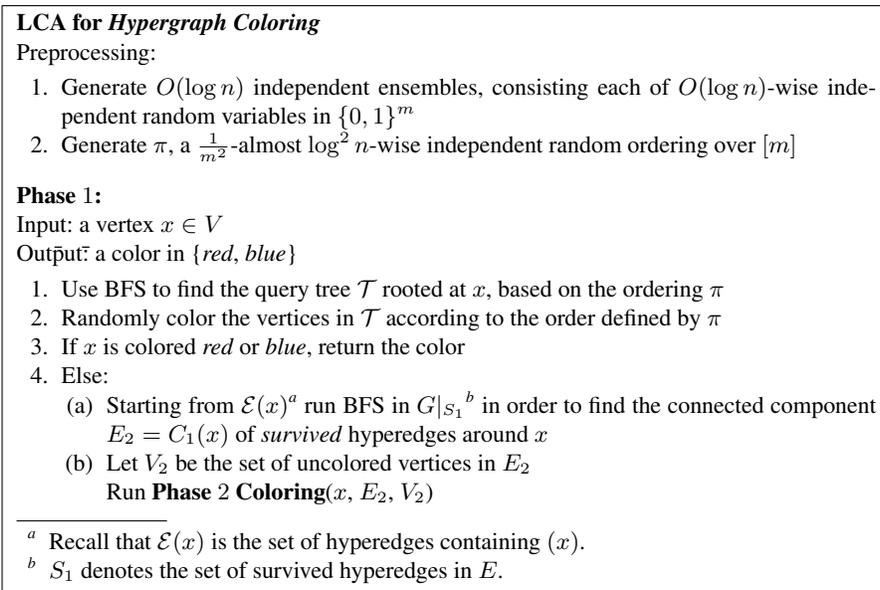

LCA for *Hypergraph Coloring*

Preprocessing:

1. Generate $O(\log n)$ independent ensembles, consisting each of $O(\log n)$-wise independent random variables in $\{0,1\}^m$
2. Generate π, a $\frac{1}{m^2}$-almost $\log^2 n$-wise independent random ordering over $[m]$

Phase 1**:**

Input: a vertex $x \in V$

Output: a color in $\{red, blue\}$

1. Use BFS to find the query tree $\mathcal{T}$ rooted at x, based on the ordering π
2. Randomly color the vertices in $\mathcal{T}$ according to the order defined by π
3. If x is colored *red* or *blue*, return the color
4. Else:
   (a) Starting from $\mathcal{E}(x)^a$ run BFS in $G|_{S_1}{}^b$ in order to find the connected component $E_2 = C_1(x)$ of *survived* hyperedges around x
   (b) Let V_2 be the set of uncolored vertices in E_2
   Run **Phase** 2 **Coloring**(x, E_2, V_2)

---
^{a} Recall that $\mathcal{E}(x)$ is the set of hyperedges containing (x).
^{b} S_1 denotes the set of survived hyperedges in E.

Fig. 1. Local computation algorithm for *Hypergraph Coloring*

of vertices in at most $O(\log n)$ edges. Therefore in total we color at most $O\left(\log^2(n)\right)$ vertices.

Finally, since we are only interested in $O\left(\log^2(n)\right)$ vertices, we may consider a coloring which is only $O\left(\log^2(n)\right)$-wise independent, and a random ordering which is only n^{-3}-almost $O\left(\log^2(n)\right)$-independent. Given the construction in [2], this can be done in space and time complexity $O\left(\log^4(n)\right)$

C.3 Phase 2 and 3: gradually decreasing the component size

Phase 2 and 3 are simply iterations of the general phase with parameters as described below. With high probability we have that $|E_2| \leq 4d^3 \log n$, and each edge has k_2 uncolored vertices. After at most $t_2 = \log n$ repetitions of the random coloring procedure, we reach an assignment that leaves a size $2d^3 \log \log n$-connected component of survived edges with probability

$$\left(\left(4d^3 k_2 \log n\right) 2^{-2 \log \log n}\right)^{\log n} < n^{-3}$$

Similarly, in the third phase we begin with $|E_3| < 2d^3 \log \log n$, and after $t_3 = \log n$ repetitions we reach an assignment that leaves a size $\frac{\log \log n}{k_4}$-connected component of survived edges with probability

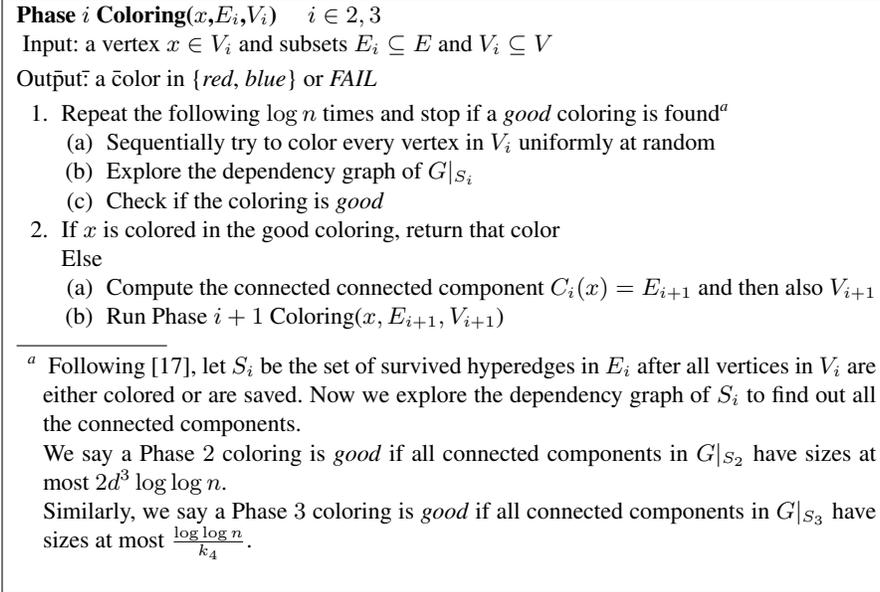$$\left(\left(2d^3 k_3 \log \log n\right) 2^{-\frac{\log \log n}{d^3 k_4}}\right)^{\log n} < n^{-3}$$

Phase i **Coloring**(x,E_i,V_i) $i \in 2, 3$

 Input: a vertex $x \in V_i$ and subsets $E_i \subseteq E$ and $V_i \subseteq V$

Output: a color in {*red*, *blue*} or *FAIL*

1. Repeat the following $\log n$ times and stop if a *good* coloring is found[a]
   - (a) Sequentially try to color every vertex in V_i uniformly at random
   - (b) Explore the dependency graph of $G|_{S_i}$
   - (c) Check if the coloring is *good*
2. If x is colored in the good coloring, return that color
   Else
   - (a) Compute the connected connected component $C_i(x) = E_{i+1}$ and then also V_{i+1}
   - (b) Run Phase $i + 1$ Coloring(x, E_{i+1}, V_{i+1})

---

[a] Following [17], let S_i be the set of survived hyperedges in E_i after all vertices in V_i are either colored or are saved. Now we explore the dependency graph of S_i to find out all the connected components.
We say a Phase 2 coloring is *good* if all connected components in $G|_{S_2}$ have sizes at most $2d^3 \log \log n$.
Similarly, we say a Phase 3 coloring is *good* if all connected components in $G|_{S_3}$ have sizes at most $\frac{\log \log n}{k_4}$.

Fig. 2. Local computation algorithm for *Hypergraph Coloring*: Phase 2 and Phase 3

Phase 4 **Coloring**(x, E_4, V_4)

Input: a vertex $x \in V_4$ and subsets $E_4 \subseteq E$ and $V_4 \subseteq V$

Output: a color in {*red*, *blue*}

1. Go over all possible colorings of the connected component V_4 and color it using a feasible coloring.
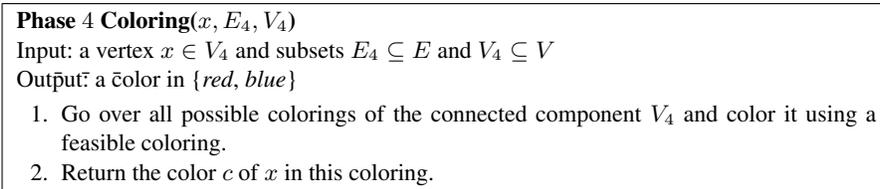2. Return the color c of x in this coloring.

Fig. 3. Local computation algorithm for *Hypergraph Coloring*: Phase 4

C.4 Phase 4: brute force

Finally, we are left with a connected component of $|E_4| < \frac{\log \log n}{k_4}$, and each edge has k_4 uncolored vertices. By the Lovasz Local Lemma, there must exists a coloring (see e.g. Theorem 5.2.1 in [1]). We can easily find this coloring via brute force search in time $O(\log n)$.

D k-CNF

As another application, our hypergraph coloring algorithm can be easily modified to compute a satisfying assignment of a k-CNF formula, provided that the latter satisfies some specific properties.

Let H be a k-CNF formula on m Boolean variables $x_1, \ldots, x_m$. Suppose H has n clauses $H = A_1 \wedge \cdots \wedge A_n$ and each clause consists of exactly k distinct literals.[12] We say two clauses A_i and A_j *intersect* with each other if they share some variable (or the negation of that variable). As in the case for hypergraph coloring, k and d are fixed constants and all asymptotics are with respect to the number of clauses n (and hence m, since $m \leq kn$). Our main result is the following.

Theorem 7. *Let H be a k-CNF formula with $k \geq 2$. Suppose that each clause intersects no more than d other clauses, and furthermore suppose that $k \geq 16 \log d + 19$. Then there exists a $(O(\log^4 n), O(\log^4 n), 1/n)$-local computation algorithm which, given a formula H and any sequence of queries to the truth assignments of variables $(x_1, x_2, \ldots, x_s)$, with probability at least $1 - 1/n^2$, returns a consistent truth assignment for all x_i's which agrees with some satisfying assignment of the k-CNF formula H. Moreover, the algorithm is query oblivious and parallelizable.*

Proof [Sketch]:   We follow a 4-phase algorithm similar to that of hypergraph two-coloring as presented in appendix C. In every phase, we sequentially assign random values to a subset of the remaining variables, maintaining a threshold of k_i unassigned variables in each unsatisfied clause. Since the same (in fact, slightly stronger) bounds that hold for the connected components in the hyperedges dependency graph also hold for the clauses dependency graph ([17]), we can return an answer which is consistent with a satisfying assignment with probability at least $1 - 1/n^2$. □

E   Lower bound on the size of the query tree

We prove a lower bound on the size of the query tree.

Theorem 8. *Let G be a random graph whose vertex degree is bounded by $d \geq 2$ or distributed independently and identically from the binomial distribution: $deg(v) \sim B(n, d/n)$ $(d \geq 2)$. Then*

$$\Pr[|T| > \log n / \log \log n] > 1/n,$$

---

[12]   Our algorithm works for the case that each clause has at least k literals; for simplicity, we assume that all clauses have uniform size.

where the probability is taken over all random permutations $\pi \in \Pi$ of the vertices, and T is the largest query tree in G (under π).

Proof. For both the bounded degree and the binomial distribution cases, there exists a path of length at least $k = \log n / \log \log n$ in the graph w.h.p. Label the vertices on the path $v_1, v_2, \ldots, v_k$. There are $k!$ possible permutations of the weights of the vertices on the path. The probability of choosing the permutation in which $w(v_1) < w(v_2) < \ldots < (v_k)$ is $1/k!$.

$$k! = (\log n / \log \log n)!$$
$$< (\log n / \log \log n)^{\log n / \log \log n}$$
$$< n.$$

Therefore, $1/k! > 1/n$ and so the probability of the query tree having size $\log n / \log \log n$ is at least $1/n$. □