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 propose 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 this improved 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

⋆ For the full version of the paper, please see $www.tau.ac.il/\sim shaivar1/OnlineLCA.pdf$

⋆⋆ mansour@post.tau.ac.il. Supported in part by the Google Inter-university center for Electronic Markets and Auctions, by a grant from the Israel Science Foundation, by a grant from United States-Israel Binational Science Foundation (BSF), by a grant from Israeli Centers of Research Excellence (ICORE), and by a grant from the Israeli Ministry of Science (MoS).

⋆⋆⋆ aviadrub@mail.tau.ac.il.

† shaivar1@post.tau.ac.il. Supported in part by the Google Inter-university center for Electronic Markets and Auctions

‡ ningxie@csail.mit.edu. Supported by NSF grants CCF-0728645, CCF-0729011 and CCF-1065125.

[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 $y \in F(x)$.

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 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., [6]) 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. [9, 12]) limit the space domain, by limiting the ability of an algorithm to observe the entire input, and still strive to derive global properties of it.

Local computation algorithms (LCAs) [13] 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. On any query of a job, the LCA answers quickly the job's machine. The correctness property of the LCA guarantees that different query replies will be consistent with some admissible solution.

1.2 Our results

1.2.1 Bounds on query trees Suppose that we have an online algorithm where the reply to a query depends on the replies to a small number of previous queries. The reply to each of those previous queries depends on the replies to 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 the replies to a given query on the replies to other queries.

Bounding the size of a query tree 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 degree bound of the dependency graph, appears only in the constant. [4] This answers in the affirmative the conjecture of [1]. Previously, Alon et al. [1] show that the expected size of the query tree is constant, and $O(\log^{d+1} n)$ w.h.p.[5] 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, in the full version of this paper, 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.

[4] Note that, however, the hidden constant is exponentially dependent on d. Whether or not this bound can be improved to have a polynomial dependency on d is an interesting open question.

[5] 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.

1.2.2 Hypergraph 2-coloring We modify the algorithm of [1] 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.

1.2.3 *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: first a random permutation of the items is generated on the fly. Then, for each query, we simulate the online algorithm on a stream of input items arriving according to the order of the random permutation. 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:

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

1.2.5 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., [2, 5, 14]) to efficient LCAs.

1.3 Related work

Nguyen and Onak [11] 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. [13] 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 [3], and a reduction from distributed algorithms. Alon et al. [1], building on the technique of [11], show how to extend several of the algorithms of [13] 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 how to construct a $1/n^2$-almost k-wise independent random ordering[6] 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 the size of a minimum vertex cover, maximal matching, maximum matching, minimum dominating set, and other problems (cf. [12, 9, 11, 16]), 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. (e.g. [2, 5, 8, 14]). For a survey on the power of d choices, we refer the reader to [10].

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

[6] 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.

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 ([11] and [1]): Each query v 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 on which x depends. 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 need 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 to vertices in the graph needed 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 [1] 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.

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 of Lemma 1 consists of two parts. Following [1], we partition the query tree into $L = 3d$ 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 [1]. We extend their proof to the binomial case; that is, we prove the following, where $T_i^{(j)}$ is the j-th subtree on level i of the tree.

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

The proof can be found in the full version of this paper.

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. We use the distribution of the sizes of the subtrees, in order to bound the aggregate of multiple subtrees.

3.3 Bounding the increase in subtree size as we go up levels

From [1] and Proposition 1 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 bounded degree and the binomial degree cases (see the full version for a more complete discussion). 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 Proposition establishes our inductive step.

Proposition 2. *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 $Z_{k+1} = \sum_{i=1}^{Y_k} z_i$.

From [1] and Proposition 1, 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}.

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:

$$\begin{aligned}
\Pr[|T_{k+1}| = Z_{k+1}] &< \binom{Z_{k+1} + Y_k}{Y_k} e^{-c_0(Z_{k+1} - \beta Y_k)} \\
&< \left(\frac{e \cdot (Z_{k+1} + Y_k)}{Y_k} \right)^{Y_k} e^{-c_0(Z_{k+1} - \beta Y_k)} \\
&= (e(1+\eta))^{Y_k} e^{-c_0(\eta - \beta) Y_k} \\
&= e^{Y_k(-c_0(\eta-\beta) + \ln(\eta+1) + 1)} \\
&\leq e^{-c_0 \eta Y_k / 2},
\end{aligned}$$

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 $O(\log n)$. □

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 [1] for hypergraph 2-coloring. We also modify their algorithm slightly to further improve the algorithm's complexity. Due to space limitations, we only state our main theorems for hypergraph 2-coloring and k-CNF; the proofs can be found in the full version of this paper.

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.

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 consider 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 [1] to produce an almost $O(\log n)$-wise independent random ordering on the edges, using a seed length of $O(\log^3 n)$.[7] 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.

As the query tree is an upper-bound on the size of the DAG, we derive the following theorem from Lemma 1.

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.*

[7] 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 The bipartite case and 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. [8]). [8]

6.1 The bipartite case

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 j, we add vertices one at a time to the tree, from left to right, until the depth is

[8] 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.

"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.

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.[9] 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 can be found in the full version of the paper.

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.

6.2 Local load balancing

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

[9] 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.

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 $\mathcal{LB}$. □

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

Corollary 1. *(Using [5]) 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 [15]) 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.* [10]

Corollary 3. *(Using [4]) 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 [4]) 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 [7]) 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.*

6.3 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

[10] 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 $k \geq 1$ $F_d(k) = \sum_{i=1}^{d} F_d(k-i)$

size of the random query is $O(\log n)$, an $O(\log n)$-*wise independent random ordering*[11] suffices for our local computation purpose. Using the construction in [1] 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.

References

[1] 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.

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

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

[4] 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.

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

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

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[11] See [1] for the formal definitions of k-wise independent random ordering and almost k-wise independent random ordering.