Simultaneously Load Balancing for Every p-norm, With Reassignments*

Aaron Bernstein[1], Tsvi Kopelowitz[2], Seth Pettie[3], Ely Porat[4], and Clifford Stein[5]

1 **Columbia University, New York, USA**
bernstei@gmail.com
2 **University of Michigan, Ann Arbor, USA**
kopelot@gmail.com
3 **University of Michigan, Ann Arbor, USA**
pettie@umich.edu
4 **Bar Ilan University, Ramat Gan, Israel**
porately@cs.biu.ac.il
5 **Columbia University, New York, USA**
cliff@ieor.columbia.edu

Abstract

This paper investigates the task of load balancing where the objective function is to minimize the p-norm of loads, for $p \geq 1$, in both static and incremental settings. We consider two closely related load balancing problems. In the bipartite matching problem we are given a bipartite graph $G = (C \cup S, E)$ and the goal is to assign each client $c \in C$ to a server $s \in S$ so that the p-norm of assignment loads on S is minimized. In the graph orientation problem the goal is to orient (direct) the edges of a given undirected graph while minimizing the p-norm of the out-degrees. The graph orientation problem is a special case of the bipartite matching problem, but less complex, which leads to simpler algorithms.

For the graph orientation problem we show that the celebrated Chiba-Nishizeki peeling algorithm provides a simple linear time load balancing scheme whose output is an orientation that is 2-competitive, in a p-norm sense, for all $p \geq 1$. For the bipartite matching problem we first provide an offline algorithm that computes an optimal assignment. We then extend this solution to the online bipartite matching problem with reassignments, where vertices from C arrive in an online fashion together with their corresponding edges, and we are allowed to reassign an amortized $O(1)$ vertices from C each time a new vertex arrives. In this online scenario we show how to maintain a single assignment that is 8-competitive, in a p-norm sense, for all $p \geq 1$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Online Matching, Graph Orientation, Minimizing the p-norm

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.51

1 Introduction

Online algorithms for load balancing have received much attention in recent years. There are many variants of load balancing, and in this paper we consider two closely related load balancing problems – bipartite matchings and graph orientations. These problems have many natural applications in areas such as internet advertising, social networks, and routing. In the standard online paradigm, an object such as a vertex or edge arrives online

* Research supported in part by NSF grants CCF-1421161, CCF-1514383, and CCF-1637546.

and then a decision, such as orienting an edge or matching two vertices, must be made immediately. This decision is *irrevocable*. It is well understood that, for most problems, an online algorithm cannot perform as well as an offline algorithm which knows the future requests. Sometimes the gap can be quite large.

It is therefore natural to consider giving the online algorithm a small amount of additional power. This idea has taken many forms such as resource augmentation, randomization, advice, or knowledge of an input distribution. Here we study the idea of allowing the online algorithm to "redo" some of its earlier decisions. Since we are studying polynomial-time solvable problems such as matching and orientation, we have to be sure not to allow too many opportunities to redo decisions, as we would then just simulate an offline algorithm. In particular, we will allow edge reorientations or reassignments, but will focus on algorithms that allow only a small number of reassignments. Our results will, in general, show that by allowing a small number of reassignments, we can maintain a much better matching or orientation.

The p-norm

Typically one evaluates a matching by its size or total cost and one evaluates an orientation by its maximum edge out-degree, or load. In this paper, we consider a more general class of objective functions, evaluating the p-norm, for any $p \geq 1$. For a vector $X = (x_1, x_2, \ldots, c_k)$ the p-norm of X is denoted by $\|X\|_p = (\sum_{i=1}^{k} x_i^p)^{1/p}$. Often, with a load-balancing problem, there is a tension between finding a solution that minimizes the maximum load and one that fairly distributes load among various agents. Considering only the maximum load ignores fairness among the non-maximum load nodes and considering only the average ignores the possibility of a solution with one or more nodes of very high load. This tradeoff has been studied in various ways in previous work, e.g. [19, 24, 14, 25, 4]. We address this tradeoff by considering p-norms. By using a small constant $p > 1$, it is well understood that one balances these two competing objectives. In this paper, we will give algorithms that perform well on all norms simultaneously. This not only shows that the algorithm does not need to be specialized to the norm, but also shows that there exist solutions that perform well in all norms simultaneously.

1.1 Edge orientation and bipartite matching

We investigate the goal of minimizing the p-norm of loads by considering two closely related problems.

Graph orientations

In the graph orientation problem, the goal is to orient (direct) the edges of an undirected graph G while minimizing some function of the out-degrees, and in our case the p-norm. If we consider an *out-degree vector* X of length n where x_i is the out-degree of vertex v_i, then one can word this goal as minimizing $\|X\|_p$.

Bipartite matching

In the online bipartite matching with reassignments problem a bipartite graph contains two sets of vertices: there is a set of vertices S (*server* vertices) that is fixed in advance, and a set of vertices C (*client* vertices) that grows over time. In particular, at each time step, a new client c is inserted into C along with edges from c to some non-empty subset of S. We

say that a server $s \in S$ is a *neighbor* of c if edge (c, s) exists. Given any time t, we let C_t be the set of client vertices at time t. We always let C refer to the set of vertices at the *current* time t. Similarly, we let $G_t = (V_t, E_t)$ refer to the resulting bipartite graph at time t (so $V_t = C_t \bigcup S$), and G refer to the graph at the current time t.

The online algorithm must at all times maintain a *global assignment* function $A : C \to S$, which assigns each client $c \in C$ to some server $s \in S$. Whenever a new client c is inserted into C, it must immediately be *assigned* to one of its neighbors in S. However, unlike in standard online matching, we also allow the algorithm to reassign $O(1)$ vertices from C each time a client arrives. Given any $s \in S$, let $\text{LOAD}_A(s) = |\{c \in C : A(c) = s\}|$ refer to the number of client vertices assigned to s by A. Let $\overline{\text{LOAD}}_A = (\text{LOAD}_A(s_1), \text{LOAD}_A(s_2), \ldots, \text{LOAD}_A(s_k))$ where $S = \{s_1, s_2, \ldots, s_k\}$. The vector $\overline{\text{LOAD}}_A$ is is called the *load vector of* A. The goal is to maintain an assignment A that minimizes $\|\overline{\text{LOAD}}_A\|_p$.

Notice that if each client arrives with exactly two neighboring vertices in S, then the online bipartite matching problem corresponds to an incremental version of the graph orientation problem (where edges arrive online and orientations are allowed to be flipped) by considering S as the set of vertices and C as the set of edges.

1.2 Our results

Our main result is to show that for the online bipartite matching with reassignments problem we can maintain an 8-approximation to the optimal assignment, while reassigning a small number of clients. To do so, we first show how to find an optimal offline solution using an adaptation of augmenting paths as follows. Given a global assignment A we orient[1] the edges of G as follows: an edge (s, c) is oriented $s \to c$ if $A(c) = s$, and otherwise it is oriented $c \to s$. This oriented graph is denoted by G_A.

▶ **Definition 1.** For a bipartite graph $G = (C \cup S, E)$ and a global assignment A, a directed path P in G_A from server $x \in S$ to server $y \in S$ is an augmenting path if $\text{LOAD}_A(y) \leq \text{LOAD}_A(x) - 2$.

▶ **Theorem 2.** *For a bipartite graph $G = (S \cup C, E)$ and for any $1 \leq p < \infty$, a global assignment A is optimal with respect to the p-norm if and only if G_A has no augmenting paths.*

Our online algorithm will always keep track of the optimal offline solution in the background (Theorem 2 provides a simple algorithm for keeping track of OPT as new clients arrive), and use it to influence its online assignments. Note that even though the online algorithm knows OPT it does not directly use the same assignment as OPT because maintaining OPT as new vertices are inserted into C may require too many reassignments.

Let OPT_t be an optimal assignment at time t. We are now ready to state our main theorem for online assignment.

▶ **Theorem 3.** *There exists an algorithm for online assignment which only performs an amortized $O(1)$ reassignments per time step (i.e. $O(1)$ per new client vertex), and ensures that at every time t, for any $s \in S$, $\text{LOAD}_A(s) \leq 8 \cdot \text{LOAD}_{\text{OPT}_t}(s)$. Note that the resulting assignment is always 8-competitive to the optimal assignment for any p-norm where $p \geq 1$, including $p = \infty$.*

[1] Notice that this is not the same orientation as in the graph orientation problem, although the two are related.

Results for graph orientation

While Theorem 2 provides a method for finding an optimal orientation in G in polynomial time (since the graph orientation problem is a special case), the runtime (via a straightforward implementation) is $O(m^2)$ where m is the number of edges. Instead, based on the Chiba and Nishizeki [7] triangle enumeration algorithm, we provide a simple linear time algorithm for finding an orientation that is a 2-approximation to the optimal p-norm. For an orientation $\mathcal{O}$ and a vertex u Let $d^+_{\mathcal{O}}(u)$ be the out-degree of u in the graph oriented by $\mathcal{O}$.

▶ **Theorem 4.** *There exists a linear time algorithm for the graph orientation problem on a graph $G = (V, E)$ producing an orientation $\mathcal{O}$ such that for any $p \geq 1$ and any vertex $u \in V$, $d^+_{\mathcal{O}}(u) \leq 2 \cdot d^+_{\text{OPT}}(u)$ where* OPT *is an optimal orientation with respect to the p-norm.*

We emphasize that one can use Theorem 3 to solve an incremental/online version of the graph orientation problem with a constant number of edge flips to maintain an orientation with a close to optimal p-norm of the out-degrees. The nice benefit of the proof of Theorem 4 is that it introduces and investigates a new natural combinatorial measurement of graphs, called the *local degeneracy*, which may be of independent interest.

1.3 Overview and Techniques for Bipartite Matching

To prove Theorem 3 we start by proving a simpler theorem.

▶ **Theorem 5.** *Say that for some specific time t_f we know in advance an upper bound on $\text{LOAD}_{\text{OPT}_{t_f}}(s)$ for every $s \in S$: that is, we know some function $u : S \to \mathbb{N}$ such that $\text{LOAD}_{\text{OPT}_{t_f}}(s) \leq u(s)$ for every $s \in S$. Then, there is an online assignment protocol that performs an amortized $O(1)$ reassignments per time step, and guarantees that at time t_f, every server $s \in S$ has $\text{LOAD}_{t_f}(s) \leq 2u(s)$.*

The proof of Theorem 5 is based on a recent result of Gupta, Kumar, and Stein [16] that also studies the problem of online assignment with reassignments, but they solve a simpler problem of trying to minimize the *maximum* load (i.e. the L_∞ norm of the loads) while performing as few reassignments as possible. In particular, they achieve the following results (the second result is a combination of the first result and the standard guess and double technique):

▶ **Theorem 6.** *[16] Say that we know in advance that at some final time step t_f, there is a solution in which every server $s \in S$ has load at most k. Then, there is an online assignment protocol that performs an amortized $O(1)$ reassignments per time step, and achieves a maximum load of $2k$ at time t_f.*

▶ **Theorem 7.** *[16] There exists an algorithm for online assignment which performs $O(1)$ reassignments per time step and has the following guarantee: at every time step t, if there exists an offline solution in which every $s \in S$ has a load of at most k_t, then the online protocol achieves a maximum load of at most $8k_t$.*

Although there is a trivial extension from Theorem 6 to Theorem 5, there is no trivial extension from Theorem 7 to Theorem 3. The intuition for why not knowing the optimal maximum load in advance poses a problem is as follows. Loosely speaking, the analysis of Theorem 6 (where the optimal max load k is known in advance) relies on the fact that the server vertices are only getting more and more constrained: if we know in advance that the

final load should be k, then since the load on every server $s \in S$ can only increase over time, as more clients arrive less and less capacity remains before we reach load k. But if the loads are not known in advance, then the algorithm must occasionally increase the desired load on s from k to say $2k$, which effectively makes s *less* constrained, and destroys the analysis in Theorem 6.

Theorem 7 is able to very easily overcome this obstacle because as long as we only care about the maximum load, the desired load for all $s \in S$ changes at the same time: as more vertices are inserted into C, the algorithm sometimes realizes that max load k is no longer feasible, so it increases the max allowable load to $2k$ for all vertices $s \in S$. Such global changes are easy to handle by effectively restarting the entire algorithm every time the desired max load doubles. But this approach does not work for Theorem 3 because here the algorithm must aim for a different load on every server, and these loads will change at different times. This prevents us from applying a global restart, and locally changing the allowable load on one server ends up destroying the analysis for the not-yet-changed parts of the graph.

We show, however, that it is nonetheless possible to do a reduction from Theorem 3 (loads not known in advance) to Theorem 5 (loads known in advance), but the reduction is significantly more complicated, and requires a more careful analysis of how the optimal offline assignment OPT relates to our online assignment. The main idea is to create another graph G^* by carefully duplicating servers with exponentially increasing capacities. Once the first i servers are full we begin using the $i+1$'th server. We show that maintaining an optimal solution in G^* can be used to maintain a close to optimal solution in G, and then we show how to maintain a close-to-optimal solution in G^*.

1.4 Overview and Techniques for Graph orientations: The Local Degeneracy

When considering the ∞-norm of out-degrees, a natural parameter to consider is the *arboricity* or *degeneracy* of the graph. The degeneracy of an undirected graph $G = (V, E)$ is the largest minimum degree of any subgraph of G. Formally the degeneracy of G is $\delta(G) = \max_{U \subseteq V} \min_{u \in U} d_{G_U}(u)$ where d_{G_U} is the degree of u in the subgraph of G induced by U. The degeneracy is roughly equal (up to constant factors) to other well known graph parameters such as the arboricity, strength, or thickness of a graph. Of particular interest is the arboricity $\alpha(G)$ of G since the maximum out-degree of any orientation is at least $\alpha(G) - 1$, and $\alpha(G) \leq \delta(G) \leq 2\alpha(G)$-1.

Chiba and Nishizeki [7] gave a linear time algorithm, called the *CN-peeling* algorithm, that computes a $\delta(G)$-orientation as follows. Repeatedly remove a vertex u in (the current) G with smallest (current) degree, together with the edges E_u touching u at the time u is removed. For each u, all of the edges E_u are oriented as leaving u. The out-degree of each vertex is shown to be at most $\delta(G)$ using straightforward arguments. The CN-peeling algorithm provides an acyclic orientation of G, where for each vertex u all of the edges E_u are oriented as leaving u. This acyclic orientation is used to enumerate all triangles of the graph in time $O(m \cdot \delta(G))$ with the following observation: for each triangle v_1, v_2, v_3 in G there must be at least one vertex of the three with both edges oriented outwards. Thus, it suffices to consider, for each vertex, all pairs of outgoing edges.

Local versus global parameters

While the algorithm of [7] is essentially tight for worst-case inputs (see [21, 22]), for many graphs the runtime bound of $O(m \cdot \delta(G))$ is not tight. For example, consider a graph G composed of a clique on $n^{1/3}$ vertices, and a tree on the rest of the vertices, with the tree being connected to the clique with one edge. The degeneracy of G is $\Theta(n^{1/3})$ and so $O(m \cdot \delta(G)) = O(n^{4/3})$. However, it is obvious that triangle enumeration through the CN-peeling algorithm runs in $O(n)$ time.

Why is this analysis not tight enough? Because the degeneracy is a global parameter of G which is largely due to a small part of G being dense. However, in the analysis, this global parameter should not affect the time cost of other sparser parts in G. It would therefore be more beneficial to express the runtime using tighter graph parameters which are more appropriate in determining the local cost of each vertex.

Local degeneracy

We extend the notion of degeneracy by introducing a new local sparseness measurement. The *local degeneracy* (LD) of a vertex v in G is $\ell_v = \max_{U \subseteq V, v \in U} \min_{u \in U} d_{G_U}(u)$. In words, the LD of v is the largest minimum degree of any subgraph of G that contains v. It is straightforward to show that in the orientation obtained from the CN-peeling algorithm the out-degree d_u^+ of a vertex u is at most its LD ℓ_u. The runtime of the CN-peeling algorithm for triangle enumeration is then $\sum_{v \in V} (d_u^+)^2 \leq \sum_{v \in V} (\ell_u)^2$, which is always at most $O(m \cdot \delta(G))$, but could be much less. In the example graph G above, this turns out to be $O(n)$. So, at least in the example, LD captures a locality notion that better expresses the runtime of the peeling algorithm.

We prove that LD captures a fundamental property of graph orientations stated in the following theorem.

▶ **Theorem 8.** *For an undirected graph $G = (V, E)$ and for any $1 < p < 0$, let* OPT *be any optimal orientation of the edges with respect to the p-norm. Let $d^+_{\text{OPT}}(v)$ be the out-degree of $v \in V$ when G is oriented according to* OPT*. Then $d^+_{\text{OPT}}(v) > \ell_v/2 - 1$. For the ∞-norm, the same is true for some optimal orientation.*

Theorem 8 has two important consequences. The first is that the CN-peeling algorithm gives an orientation whose out-degrees are 2-competitive to the optimal orientation with respect to the p-norm (Theorem 4). The second implication is that any algorithm that uses an acyclic orientation for enumerating all triangles in a given graph, by separately considering for each vertex all pairs of its outgoing edges, cannot be asymptotically faster than the CN-peeling algorithm.

Local degeneracy and the k-core

The *k-core* of G is the maximum subgraph of G with minimum degree at least k, and is denoted by G_k. As it turns out, if $u \in G_k - G_{k+1}$ then $\ell_u = k$ (more on this in Section 4). While the notion of a k-core is certainly not new, the focus from an algorithmic perspective has mainly been on computing the k-core of a graph for a given k. In other words, the focus has been on the k-core from a global prospective of the graph. Our focus is on a local property of vertices, which is captured by the local degeneracy of each vertex. Thus, we find the terminology of *local degeneracy* to be more appropriate for our objectives.

1.5 Related work

There is a large body of work that deals with the tradeoff between reassignment and the quality of an online solution. E.g., consider the problem of load balancing, where each arriving task has a processing time but can run only on some subset of the servers. Here, Phillips and Westbrook [28] and Westbrook [33] showed that a small number of reassignments (linear in the number of tasks) improve the quality of the solution, getting a constant competetive ratio, compares to logarithmic lower bounds in the case without reassignments [3]. There has also been work on the case of identical machines [30, 13, 29, 32, 12] and reassignments for other online problems such as Steiner Tree [18, 26, 15].

Related work on online bipartite matching

For online matching, the maximization version has been studied extensively. In this model all arriving nodes do not need to be matched, and the goal is to maximize the reward accrued by successfully matching a large number/weight of these nodes. It is well known that the greedy algorithm is 1/2-competitive for maximum matching. There are many papers on variants including when the capacity on one side is large relative to the requests, or when there is stochastic information, or when randomization is allowed. These results are orthogonal to our line of investigation, and we omit a detailed discussion.

Most relevant to our work is a paper by Gupta, Kumar and Stein [16] that studies the problem of online assignment in the same model as this paper. For online bipartite matching, where the left vertices arrive online and must be matched to the right vertices, they give an algorithm that reassigns the left vertices an (amortized) constant number of times, and maintains a constant factor to the optimal load on the right vertices. They then extend this result in several ways. For restricted machine scheduling with arbitrary sized jobs, they give an algorithm that maintains load which is $O(\log \log mn)$ times the optimum, and reassigns each job only an (amortized) constant number of times. They also give an algorithm for online flow that reroutes flow an (amortized) constant number of times while achieving constant factor approximation to the congestion. The bounds in the SODA proceedings version of [16] are correct, but there is an error in the proof. Work on this paper revealed that proof, which the authors have since corrected (the corrected verion has not yet been made public). Our work generalizes the results in that paper.

Related work on graph orientations

The task of maintaining an edge orientation with the goal of minimizing the maximum out-degree has also received a lot of attention [5, 20, 17]. Edge orientations have many algorithmic applications including "color-coding" [1], adjacency queries [8, 5, 23, 20], short-path queries [24], load balancing [6], maximal matchings [27], pattern matching [2], counting subgraphs in sparse graphs [9], prize-collecting TSPs and Steiner Trees [10], reporting all maximal independent sets [11], answering dominance queries [11], subgraph listing problems (listing triangles and 4-cliques) in planar graphs [8], and computing the girth [24].

2 Optimal Offline Bipartite Matching

Proof of Theorem 2. If A is optimal with respect to the p-norm then clearly there are no augmenting paths since flipping an augmenting path leads to a new assignment with smaller p-norm, due to convexity. So we focus on proving that if there are no augmenting paths in

G_A then A must be optimal for any p-norm with $1 < p < \infty$. For the following let p be any p in the range.

Let OPT_p be the optimal global assignment with respect to p. We partition the set of vertices in S as follows. $S^< = \{u \in S | \text{LOAD}_A(u) < \text{LOAD}_{\text{OPT}_p}(u)\}, S^> = \{u \in S | \text{LOAD}_A(u) > \text{LOAD}_{\text{OPT}_p}(u)\}$, and $S^= = S - \{S^< \cup S^>\}$. Let $E^* = \{(c,s) \in E | A(c) = s \in A \wedge \text{OPT}_p(c) \neq s\}$ be the set of edges in the symmetric difference between G_A and G_{OPT_p}. Let A^* by a global assignment of $G^* = (S \cup C, E^*)$ where each assignment of a vertex from C is made according to its assignment in A. Notice that each vertex in C has either degree 0 or 2 in G^*.

Let C be a directed cycle in $G^*_{A^*}$. Notice that if we were to flip all of the edges of C in G_A, then the out-degree of each vertex would not change due to the flip, and so the p-norm of the corresponding assignment would remain the same. However, after flipping the edges in C these edges are oriented in the same direction in both G_A and G_{OPT_p}. So we may assume without loss of generality that $G^*_{A^*}$ contains no cycles, since otherwise we iteratively pick a cycle and flip it until no cycles are left.

Given that $G^*_{A^*}$ does not contain any directed cycles, then either E^* is empty, in which case we are done, or $G^*_{A^*}$ is a directed acyclic graph with some edges. We say a directed path from $u \in S$ to $v \in S$ in $G^*_{A^*}$ with length at least 1 is a maximal path if there is no edge entering u and no edge leaving v. Notice that a maximal path cannot start in S^- since every vertex in S^- has edges leaving it, and it cannot start in $S^=$ since every vertex in $S^=$ has the same number of incoming and outgoing edges. Similarly, a maximal path cannot end in S^+ or $S^=$. Thus, all maximal paths must begin in S^+ and end in S^-.

Consider a maximal path P from $u \in S^+$ to $v \in S^-$. We will prove that $\text{LOAD}_A(u) = \text{LOAD}_A(v) + 1$. From this it will follow that we can flip P in G_A thereby swapping the loads of u and v in A. So the p-norm of the loads of this new assignment is the same as the $\|\text{LOAD}_A\|_p$. We can then iteratively flip maximal paths until we obtain the assignment OPT_p, implying that $\|\text{LOAD}_A\|_p = \|\text{LOAD}_{\text{OPT}_p}\|_p$ as required.

If $\text{LOAD}_A(u) > \text{LOAD}_A(v) + 1$ then P is an augmenting path in G_A, which contradicts the assumption. If $\text{LOAD}_A(u) < \text{LOAD}_A(v) + 1$, then since $u \in S^+$ and $v \in S^-$ we have that $\text{LOAD}_{\text{OPT}_p}(u) \leq \text{LOAD}_A(u) + 1 \leq \text{LOAD}_A(v) + 1 \leq \text{LOAD}_{\text{OPT}_p}(v) + 2$. This implies that the reverse of P, which is a path in G_{OPT_p}, is an augmenting path, contradicting OPT_p being an optimal orientation with respect to the p-norm. ◀

By Theorem 2 an orientation that is optimal with respect to some $1 < p < \infty$ is optimal with respect to any such p, and so we denote any such orientation by OPT. We also call OPT the optimal orientation for G.

▶ **Corollary 9.** *If an orientation has no augmenting paths, then it is optimal for the ∞-norm.*

3 Online Matching with Reassignments: Preliminaries and Main Theorem Statement

Proof of 5. The proof shows how to reduce this problem to the one in Theorem 6. In particular, recall that $G = (V, E)$ always corresponds to the graph induced by $V = C \cup S$ at the current time t (so G changes as new clients are inserted). Instead of directly maintaining an online assignment in G, we create a slightly different graph $G' = (V', E')$ which is essentially equivalent to G, but where we only need to concern ourselves with the ∞-norm. We define $V' = S' \cup C$, where the set S' contains $u(s)$ copies of every vertex $s \in S$, labeled $s_1, s_2, ..., s_{u(s)}$. Then, for every edge $(c, s) \in E$, we add edges $(c, s_1), (c, s_2), ..., (c, s_{u(s)})$ to E'.

It is clear that since $\text{LOAD}_{\text{OPT}_{t_f}}(s) \leq u(s)$, at time t_f there is an assignment in G' in which every vertex $s_i \in S$ has load at most 1. But this means via applying Theorem 6 that there is an assignment protocol for G' that only requires an amortized $O(1)$ reassignments per time step, and that maintains an assignment in G' where at time t_f, every vertex s_i has load at most 2. This assignment for G' then translates directly to the desired assignment in G: assigning a client c to some copy $s_i \in V'$, corresponds to assigning c to $s \in V$. We only perform a reassignment in G when we perform one in G' (but not necessarily vice versa: a reassignment between two copies s_i and s_j in G' does not lead to a reassignment in G), so the number of reassignments in G will also be at most amortized $O(1)$ per time step. Since every copy $s_i \in V'$ has at most two client vertices assigned to it, and there are $u(s)$ copies of s in V', we will end up with $\text{LOAD}_{t_f}(s) \leq 2u(s)$, as desired. ◀

Defining a Capacity Function

Unlike Theorem 5, Theorem 3 does not specify a particular finish time t_f, so the online assignment must be a good approximation to OPT at *all* time steps. Thus, in order to rely on Theorem 5, we need to define a *fixed* capacity function $u : S \to \mathbb{N}$ such that at *all* times t we have that for every $s \in S$, $\text{LOAD}_{\text{OPT}_t}(s) \leq u(s)$.

This seems contradictory – how can u be an upper bound on $\text{LOAD}_{\text{OPT}_t}$ if the first is fixed and the latter increases with t – but the idea is to capture changes in $\text{LOAD}_{\text{OPT}_t}(s)$ by creating many copies of each $s \in S$, each with different capacities $u(s)$. (Note: these copies are unrelated to the copies used in the proof of Theorem 5.)

Recall that $G = (V, E)$ refers to the current version of the graph. We define another graph $G^* = (V^*, E^*)$ that also changes as new clients are inserted. We start by defining a new set of servers S^*, which contains an infinite number of copies for each vertex $s \in S$, labeled: $s_0, s_1, s_2, s_3...$, where copy s_i will have capacity $u(s_i) = 2^{i+1}$. Our algorithm will not have to actually handle an infinite number of copies because originally all the copies s_i of a vertex $s \in S$ are *closed*: there are no edges from C to a closed copy s_i, so s_i will never have any client vertices assigned to it and can be entirely ignored.

Recall that as new clients are inserted into the graph, we are always keeping track of an optimal offline solution OPT, and that by the proof of Theorem 2 $\text{LOAD}_{\text{OPT}}(s)$ never decreases for any $s \in S$. Let OPT_t be an optimal solution at time t. When a new vertex $c \in C$ arrives at time $t(c)$, we do the following to the graph G^*:

- If for some vertex $s \in S$, we have $\text{LOAD}_{\text{OPT}t(c)-1}(s) = 2^i - 1$ and $\text{LOAD}_{\text{OPT}t(c)}(s) = 2^i$, then we open copy $s_i \in S^*$.
- For every edge (c, s) in G, we add an edge (c, s_i) to E^* for every *open* copy s_i.

Note that our end goal is to apply Theorem 5 to G^*, so it is crucial that G^* evolves according to the definition of an online assignment problem given in Section 3: at each time step, a client arrives with all of its incoming edges, and no new edges are ever added or removed. In particular, when a new copy s_i opens up at time t, we do NOT add edges from (c, s_i) to G^* for client vertices $c \in C$ that arrived before time t. However, the lack of edges from opened server copies in S^* to older client vertices in C leads to the problem that G^* ends up being quite different from the main graph G. In particular, we can in theory imagine a situation where OPT assigns many client vertices $c_1, c_2, ..., c_q$ to some vertex $s \in S$, and yet in G^* we cannot assign all these c_i to copies of s in S^*, because even though there are many open copies of s (because $\text{LOAD}_{\text{OPT}}(s) = q$ is large), it might be the case that most of the copies s_i were created *after* the clients $c_1, ..., c_q$ arrived, so there no edges from these c_i to the copies of s in S^*. The following lemma shows that G^* is nonetheless a good

approximation to G: we first apply the Lemma to Theorem 3, and then proceed to prove the lemma.

▶ **Lemma 10.** *At all times t, there exists an assignment in G_t^* of vertices $c \in C$ to vertices $s_i \in S^*$ such that $\text{LOAD}(s_i) \leq 2^{i+1} = u(s_i)$.*

Proof of Theorem 3. The proof relies on using Theorem 5 to maintain an online assignment in G^*. Note that we do not need to specify a specific time t_f as in Theorem 5, because we know from Lemma 10 that at ALL times t there is an assignment from C to S^* in which every $s_i \in S^*$ has load at most $u(s_i) = 2^{i+1}$. Thus, by Theorem 5 we can maintain an assignment from C to S^* that only uses an amortized $O(1)$ reassignments per new client, and in which for all times t we have $\text{LOAD}(s_i) \leq 2u(s_i) = 2^{i+2}$.

The assignment from C to S^* in G^* then translates directly into an assignment from C to S in G: for every assignment $c \to s_i$, if $s_i \in S^*$ is a copy of $s \in S$, then we assign $c \to s$. The number of reassignments in G is also amortized $O(1)$ per new client, since every reassignment in G corresponds directly to one in G^* (the opposite is not necessarily true: a reassignment in G^* between two copies s_i and s_j does not translate to a reassignment in G). All we have left to show is that for each $s \in S$, and for all times t, we always have $\text{LOAD}_t(s) \leq 8\text{LOAD}_{\text{OPT}_t}(s)$. To see this, note that $\text{LOAD}_t(s) = \sum_i \text{LOAD}_t(s_i)$, where we know that $\text{LOAD}_t(s_i) \leq 2^{i+2}$ if copy i is open, and $\text{LOAD}_t(s_i) = 0$ if copy i is closed. But we only open copy s_i if $\text{LOAD}_{\text{OPT}_t}(s) \geq 2^i$, so letting j be an index for which $2^j \leq \text{LOAD}_{\text{OPT}_t}(s) < 2^{j+1}$, we have that

$$\text{LOAD}_t(s) = \sum_i \text{LOAD}_t(s_i) = \sum_{i \leq j} \text{LOAD}_t(s_i) \leq \sum_{i \leq j} 2^{i+2} < 2^{j+3} \leq 8\text{LOAD}_{\text{OPT}_t}(s). \qquad \blacktriangleleft$$

We now turn to the proof of Lemma 10, which requires a more careful analysis of the offline solution OPT in G.

▶ **Definition 11.** If at some time t OPT assigns $c \in C$ to $s \in S$ – whether because c just arrived, or because OPT chooses to reassign c at time t – we say that the assignment $c \to s$ has *level* k if $\text{LOAD}_{\text{OPT}_t}(s) = k$.

Note that if we look at all the different assignments of some $c \in C$, their level is monotonically increasing over time. This is because OPT always makes the lowest level assignment possible and LOAD_{OPT} is monotonically increasing. So if at time t_1 there was an assignment $c \to s_1$ of level k_1, and at time $t_2 > t_1$ there was an assignment $c \to s_2$ of level $k_2 < k_1$, then OPT would have instead assigned c to s_2 at time t_1.

▶ **Definition 12.** We say that a vertex $c \in C$ has *initial level* k if the assignment $c \to s$ performed by OPT when c first arrives is a level k assignment. Finally, we say that a vertex $s \in S$ is the *final level k server* of some $c \in C$ if the assignment $c \to s$ is the last level k assignment OPT performs on c.

▶ **Claim 13.** *Say that vertex $s \in S$ is the final level k server of vertex $c \in C$. Then, if at time t OPT reassigns c from s to s', we must have that $\text{LOAD}_{\text{OPT}_t}(s) > k$.*

Proof. OPT only has to reassign c from s to s' if it assigned some other c' to s. Now, we know that the assignment $c \to s'$ could not have been of level less than k because earlier c had a level k assignment to s, and the level of assignments of c is monotonically increasing. We also know that the reassignment $c \to s'$ cannot be of level k since s was the *final* level k server of c. Thus, the assignment $c \to s'$ is of level at least $k+1$, so by definition $\text{LOAD}_{\text{OPT}_t}(s') \geq k+1$. But this means that $\text{LOAD}_{\text{OPT}_t}(s) \geq k+1$, because otherwise OPT would not have reassigned c from s to s'. $\blacktriangleleft$

▶ **Claim 14.** *Let $s \in S$ be the final level k server for vertices $c_1, c_2, c_3, ..., c_q$; then $q \leq k$.*

Proof. Say, for contradiction, that $q \geq k+1$. Let t_i, for $1 \leq i \leq q$, be the time at which s became the final level k sever for c_i. Without loss of generality, let $t_1 < t_2 < ... < t_q$. We want to argue that at the end of time t_q we have $\text{LOAD}_{\text{OPT}t_q}(s) \geq k+1$, which contradicts s being the final k server for c_q. There are two cases to consider. The first case is that OPT still assigns all of $c_1, c_2, ..., c_{q-1}$ to s at the end of t_q; then, since OPT also assigns c_q to s at time t_q, we have that $\text{LOAD}_{\text{OPT}t_q}(s) \geq q \geq k+1$. The second case is that by time t_q some c_i is no longer assigned to s in OPT, in which case by Claim 13, we must have that $\text{LOAD}_{\text{OPT}t_q}(s) \geq k+1$. ◀

Proof of Lemma 10. We say that a vertex $c \in C$ has *priority* i if the initial level of c is in $(2^i, 2^{i+1}]$. We now argue the following: if vertex $c \in C$ has priority i, then for every edge $(c, s) \in E$, we have that E^* must contain edges $(c, s_0), (c, s_1), ...(c, s_i)$ (E^* might also contain edges from c to later copies of s). Let us say for the sake of contradiction that edge $(c, s_i) \notin E^*$. This can only happen if s_i was not yet open at time $t(c)$, where $t(c)$ is the arrival time of c, and so in particular if $\text{LOAD}_{\text{OPT}t(c)}(s) < 2^i$. Say that at time $t(c)$, c was assigned to $s' \in S$. Since c has priority i, we know that at the end of $t(c)$ we have $\text{LOAD}_{\text{OPT}t(c)}(s') > 2^i$. This inequality yields the desired contradiction, since OPT is not optimal, as it would have been better off assigning c to s instead of s' at time $t(c)$.

Let $C_t^i \subseteq C_t$ contain all vertices in C_t that have priority exactly i. We first observe that there is an assignment in G from all vertices $c \in C_t^i$ to vertices in S in which every $s \in S$ has load at most 2^{i+1}. In particular, we can simply assign each $c \in C_t^i$ to its final level 2^{i+1} server, and by claim 14, each $s \in S$ will end up with at most 2^{i+1} client vertices assigned to it. This implies that at all times t, there exists an assignment in G^* that assigns each vertex $c \in C_t^i$ to some ith copy $s_i \in S^*$, while maintaining $\text{LOAD}(s_i) \leq 2^{i+1} = u(s_i)$ (and $\text{LOAD}(s_j) = 0$ for all $j \neq i$.) The reason is that we can take the above assignment from C_t^i to S in the main graph G, and for every assignment $c \to s$ in G, we can simply assign $c \to s_i$ in G^*; we know from the above paragraph that edge (c, s_i) necessarily exists in G^* because c has priority i. This completes the proof of Lemma 10, since our final assignment in G^* from C_t to S^* is simply the union among all priorities i of all the assignments from C_t^i to the ith copies $s_i \in S^*$. ◀

4 Graph Orientation

Orienting with local degeneracy

We use Chiba and Nishizeki's peeling algorithm. Let $(v_1, \ldots, v_n)$ be the order of vertices as encountered by the peeling algorithm. Recall that E_u is the set of edges touching u at the time u is peeled.

▶ **Lemma 15.** $|E_{v_i}| \leq \ell_{v_i}$.

Proof. The subgraph of G at the point where v_i is peeled has a minimum degree of $|E_{v_i}|$ so ℓ_{v_i} must be at least $|E_{v_i}|$. ◀

Consider the k-core of G, denoted by G_k. Clearly, every vertex in G_k must have local degeneracy at least k. Seidman in [31] showed that by recursively deleting (peeling) vertices with degree less than k one obtains G_k.

▶ **Lemma 16.** *For v_i let $j \leq i$ be the smallest integer such that $|E_{v_j}| = \max_{1 \leq k \leq i}\{|E_{v_k}|\}$. Then $\ell_{v_i} = |E_{v_j}|$.*

Proof. When v_j is peeled, all vertices must have degree at least $|E_{v_j}|$ and so $\ell_{v_i} \geq |E_{v_j}|$. Furthermore, by the correctness of the process described in Seidman in [31], right before v_j is peeled the remaining graph must be a $|E_{v_j}|$-core of G, and if $\ell_{v_i} > |E_{v_j}|$ then the algorithm must encounter a vertex $v_{j'}$ where $j < j' \leq i$ such that $\ell_{v_i} = |E_{v_{j'}}| > |E_{v_j}|$ contradicting the definition of j. ◀

Proof of Theorem 4. The whole proof relies on the following simple claim: for $v, w \in V$, if there is a directed path in G oriented by OPT from w to v, then $d^+_{\mathrm{OPT}}(w) \leq d^+_{\mathrm{OPT}}(v) + 1$. This is because otherwise, OPT could be converted to a better solution by flipping the path from w to v.

Assume by contradiction that for some vertex $v \in V$, $d^+_{\mathrm{OPT}}(v) \leq \ell_v - 1$. For the rest of the proof, we ignore all vertices that are not in the ℓ_v-core. Now, clearly in G_{ℓ_v} as well we have that $d^+_{\mathrm{OPT}}(v) \leq \ell_v - 1$. On the other hand, we have that every vertex in the unoriented G_{ℓ_v} has degree at least ℓ_v.

Define S to be the set of all vertices that can reach v in G_{ℓ_v} by some directed path of oriented edges. Note that by the simple claim above, for any vertex $u \in S$ we have $d^+_{\mathrm{OPT}}(u) \leq \ell_v - 1 + 1 = \ell_v$. while $d^+_{\mathrm{OPT}}(u)$ itself is strictly less than $\ell_v/2$, so the average load in S is strictly less than $\ell_v/2$.

We will now prove a contradictory claim: S must have some edges directed into S from outside of S. This is contradictory because we defined S to be maximal. To yield the contradiction, let E_S be all edges incident in an undirected sense to S, and let E^*_S be all edges that are oriented outwards from vertices in S, possibly to some other vertex in S. We want to show that $E_S > E^*_S$, which implies that some edge is incoming into S. We know that E^*_S is the sum of the out-degrees of vertices in S, so given the upper bound on the average load in S, we have that $E^*_S < |S| \cdot \ell_v/2$. On the other hand, $E_S \geq |S| \cdot \ell_v/2$, because every vertex in S has degree at least ℓ_v. ◀

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B. Riva Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *Accepted to International Symposium on Algorithms and Computation (ISAAC)*, 2016.

3 Yossi Azar, Joseph Naor, and Raphael Rom. The competitiveness of on-line assignments. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1992.

4 Glencora Borradaile, Jennifer Iglesias, Theresa Migler, Antonio Ochoa, Gordon T. Wilfong, and Lisa Zhang. Egalitarian graph orientations. *CoRR*, abs/1212.2178, 2012.

5 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *Algorithms and Data Structures, 6th International Workshop, WADS*, pages 342–351, 1999. doi:10.1007/3-540-48447-7_34.

6 Julie Anne Cain, Peter Sanders, and Nick Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 469–476. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283433.

7 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

8 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86(2):243–266, 1991. doi:10.1016/0304-3975(91)90020-3.

9 Zdenek Dvorak and Vojtech Tuma. A dynamic data structure for counting subgraphs in sparse graphs. In *Algorithms and Data Structures - 13th International Symposium, WADS*, pages 304–315, 2013. doi:10.1007/978-3-642-40104-6_27.

10 David Eisenstat, Philip N. Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for steiner forest in planar graphs. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 626–638, 2012. URL: http://dl.acm.org/citation.cfm?id=2095116.2095169.

11 David Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. *ACM Transactions on Algorithms*, 5(4), 2009. doi:10.1145/1597036.1597042.

12 Leah Epstein and Asaf Levin. Robust algorithms for preemptive scheduling. In *ESA*, volume 6942 of *Lecture Notes in Comput. Sci.*, pages 567–578. Springer, Heidelberg, 2011. doi:10.1007/978-3-642-23719-5_48.

13 Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Proceedings of the 8th Annual European Symposium on Algorithms*, ESA '00, pages 202–210, London, UK, UK, 2000. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=647910.740462.

14 Ashish Goel, Adam Meyerson, and Serge A. Plotkin. Combining fairness with throughput: online routing with multiple objectives. In *STOC*, pages 670–679, 2000.

15 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. In *STOC*, pages 525–534, 2013.

16 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 468–479, 2014.

17 Meng He, Ganggui Tang, and Norbert Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *ISAAC*, pages 128–140, 2014.

18 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

19 Jon M. Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. *J. Comput. Syst. Sci.*, 63(1):2–20, 2001.

20 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *ICALP (2)*, pages 532–543, 2014.

21 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Dynamic set intersection. In *Proceedings 14th Int'l Symposium on Algorithms and Data Structures (WADS)*, pages 470–481, 2015.

22 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *SODA*, pages 1272–1287, 2016.

23 Lukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, 2007. doi:10.1016/j.ipl.2006.12.006.

24 Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms*, 2(3):335–363, 2006. doi:10.1145/1159892.1159895.

25 R. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *ACM EC*, 2004.

26 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. In *ICALP (1)*, pages 689–700, 2012.

27 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th ACM Symposium on Theory of Computing, STOC*, pages 745–754, 2013. doi:10.1145/2488608.2488703.

28 S. Phillips and J. Westbrook. On-line load balancing and network flow. *Algorithmica*, 21(3):245–261, 1998. doi:10.1007/PL00009214.

29 John F. Rudin, III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM J. Comput.*, 32(3):717–735, March 2003. doi:10.1137/S0097539702403438.

30 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.

31 Stephen. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

32 Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In *ESA (I)*, volume 6346 of *LNCS*, pages 36–47. Springer, Berlin, 2010. doi:10.1007/978-3-642-15775-2_4.

33 Jeffery Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000. doi:10.1006/jagm.2000.1074.