Near-Perfect Fractional Matching via Balls-and-Bins

Rajeev Motwani *
Rina Panigrahy †
Ying Xu‡

April 7, 2005

Abstract

In this paper we relate the problem of finding structures related to perfect matchings in bipartite graphs to a stochastic process similar to throwing balls into bins. Given a bipartite graph with n nodes on each side, we view each node on the left as having balls that it can throw into bins on the right to which it is adjacent. If each node on the left throws exactly one ball and each bin on the right gets exactly one ball, then the edges represented by the ball-placement forms a perfect matching. Further, if each thrower is allowed to throw a large but equal number of balls, and each bin on the right receives an equal number of balls, then the set of ball-placements corresponds to a perfect *fractional* matching. We show that several simple algorithms based on throwing balls into bins deliver a near-perfect fractional matching. For example, we show that by iteratively picking a random node on the left and throwing a ball into its least-loaded neighbor, the distribution of balls obtained is no worse than randomly throwing kn balls into n bins. Another algorithm is based on the d-choice load-balancing of balls and bins. By picking a constant number of nodes on the left and appropriately inserting a ball into the least-loaded of their neighbors, we achieve a smoother load distribution on both sides – maximum load is at most $\log\log n/\log d + O(1)$. We also present an algorithm that in linear time finds a subgraph with n edges where the degree of every node is at most 2; moreover, when each vertex on the left throws k balls, we obtain an algorithm that achieves a load within $k \pm 1$ on the right vertices.

1 Introduction

We study the problem of finding perfect (fractional) matchings in unweighted bipartite graphs using algorithms based on throwing balls into bins. While the problem of finding matchings in graphs is well-studied [16], as is the balls-and-bins formulation [20], the connection between the two problems is novel to the best of our knowledge.

A *perfect matching* is the subgraph where every node has degree exactly 1. A *perfect fractional matching* is a weighted subgraph with nonnegative weights on edges so that the total weight incident at each node is 1. There are a variety of algorithms for computing maximum matchings in bipartite graphs and the related assignment problem [12, 15, 7, 5]. Most such algorithms are based on finding augmenting paths by reduction to maximum flow [8, 1]. The fastest algorithm for finding a perfect matching in a bipartite graph with m edges and n nodes runs in $O(m\sqrt{n})$ time [12] (see also [11]). The problem of finding matchings in bipartite graphs and its generalizations have also been studied in the online setting where nodes and edges arrive online; there are algorithms that achieve a competitive ratio of $1 - 1/e$ [14, 13, 17]. Fractional matchings were studied by Azar and Litichevskey [3] and used to model the switch throughput problem.

The stochastic process of throwing balls into random bins is also a well-studied problem (see, for example, [19, 2, 4, 22, 23]. The typical goal is to study the distribution obtained by randomly assigning m

*Stanford University. E-mail: `rajeev@cs.stanford.edu`.
†Stanford University. E-mail: `rinap@cs.stanford.edu`.
‡Stanford University. Email:`xuying@cs.stanford.edu`.

indistinguishable balls into n bins, or variants of this process. If n balls are inserted into n bins, then with high probability the bin with maximum load contains $(1+o(1))\log n/\log\log n$ balls; when $m > n\log n$ the maximum load is $m/n + O(\sqrt{m\log n/n})$ [22]. Azar, Broder, Karlin, and Upfal [2] showed that instead of choosing one bin, if $d \geq 2$ bins are chosen at random and the ball is inserted into the least-loaded bin, the maximum load reduces dramatically to $\log\log n/\log d + O(1)$. Berenbrink et al [4] extended this result to the case when the number of balls m is greater than the number of bins n, showing that the maximum load is at most $\log\log n/\log d + O(1)$ above the average.

In this paper we relate the problem of finding structures related to perfect matchings in bipartite graphs to a variant of the balls-and-bins process. Given a bipartite graph with n nodes on each side, view each node on the left as having balls that it can throw into bins on the right to which it is adjacent. The balls are distinguishable; each thrower has a different color. Each ball thrown into a bin represents an edge between the thrower on the left and the bin on the right. The *load* of a bin on the right is the number of balls it receives and the load of a thrower on the left is the number of balls it throws. If each node on the left throws exactly one ball and each bin on the right gets exactly one ball, then the edges represented by the balls form a perfect matching. On the other hand, if each thrower is allowed to throw a large but equal number of balls, and each bin on the right receives an equal number of balls, then the set of ball placements corresponds to a perfect fractional matching. For example, if each node on the left throws exactly k balls and each bin on the right receives exactly k bins, then the set of ball placements corresponds to a subgraph (possibly with repeated edges) where each node has a degree exactly k; then, by assigning a weight $1/k$ to each instance of an edge in the subgraph, we obtain a perfect fractional matching. In this context, for a weighted subgraph, we define the *load* of a vertex to be its weighted degree.

Assuming the graph has a perfect matching, we show how to efficiently compute a *near-perfect fractional matching*, where the load of any left vertex is 1, and load of any right vertex is at most $1+\epsilon$. While the Hopcroft-Karp Algorithm [12] can be used to find a matching of size $n(1-\epsilon)$, it is not clear that it can be used to find a fractional matching where every node is matched and has degree at most $1+\epsilon$. We show that simple algorithms based on throwing balls into bins find a near-perfect fractional matching. For example, consider the process where we iteratively pick a random node on the left and throw a ball into its least-loaded neighbor. We show that the distribution obtained from this algorithm is no worse than randomly throwing kn balls into n bins, implying that the maximum load on any node is at most $k+O(\sqrt{k\log n}+\log n)$. This gives us an algorithm for finding a fractional matching in time $O(m\log n/\epsilon^2)$, where every node on the left has degree 1 and every node on the right has degree at most $1+\epsilon$. This can also be viewed as an online algorithm for the problem of assigning jobs represented by nodes on the left to machines on the right, under the assumption of random job arrivals. When the graph does not have a perfect matching, we can obtain a near-optimal fractional matching, where an optimal fractional matching corresponds to to a subgraph in which all vertices on the left have load 1 and the maximum load on a right vertex is minimized.

Another algorithm is based on the d-choice load-balancing of balls and bins. By picking a constant number of nodes on the left and inserting a ball into the least-loaded of their neighbors, we achieve a better distribution of load amongst the bins on the right — the maximum load is at most $\log\log n/\log d + O(1)$. However, this process ignores the load of vertices on the left which is the number of balls of each color. By appropriately choosing a lightly-loaded node on the left from the random choices and picking its least-loaded neighbor, we show how to find a subgraph in which the total load on both sides is exactly n and the maximum-loaded bin has load $\log\log n/\log d$. By increasing the number of choices from d to $\log n$, the maximum load can be reduced to 4.

By combining the load-balancing algorithms with the traditional augmenting paths algorithms, we show how to find in time $O(m\log^2 n)$ a subgraph where every node on the left has load exactly one and every node on the right has load at most two. This can be generalized to finding a subgraph where every node has load either k, $k+1$ or $k-1$ in time $O(km\log n + m\log^2 n)$, implying that a near-perfect fractional matching where each right node has load within $1 \pm \epsilon$ can be computed in $O(m \log n/\epsilon)$ time. Finally, we present an algorithm which in linear time finds a subgraph with n edges where the degree of every node is at most 2.

2 Summary of Results

Given a bipartite graph with n vertices on each side, we associate a color with each vertex on the left. View the vertices on the left as *throwers* and those on the right as *bins*. Each left vertex can throw a ball of its color into any one of its *neighboring* bins. The objective is for each left vertex to throw k balls so that each bin on the right gets close to k balls. This gives a subgraph (with edge repetition) in which each left vertex has degree k and each right vertex has degree close to k; or, if we assign each edge a weight of $\frac{1}{k}$, it gives a near-perfect fractional matching. Define the *load* or the *height* of a vertex to be the number of balls in that bin, or in the weighted case, the total degree in the subgraph. In what follows, when we say *subgraph*, it could be a weighted subgraph, or a multi-graph.

In Section 3, we propose a set of "load balancing" Algorithms and study their performance.

- **Round-Robin Algorithm:** Perform iteratively for k rounds: in each round, each thrower throws its ball into the least-loaded neighboring bin sequentially (in an arbitrary order).
- **Random-Color Algorithm:** Choose a ball of a random color (or equivalently, choose a left vertex randomly) and throw into its least-loaded neighboring bin.
- **Move-to-Low Algorithm:** Start with any assignment having k balls of each color. Perform iteratively: find some ball that can be moved to a bin of lower height, and move it.

We show that the distribution obtained from Random-Color is no worse than randomly throwing kn balls into n bins, implying that the maximum load of any bin is at most $k + O(\sqrt{k \log n} + \log n)$. When the graph does not have a perfect matching, this gives us a near-optimal fractional matching, where an optimal fractional matching corresponds to one where all vertices on the left have load 1 and the maximum load on a right vertex is minimized. We obtain the same upper bound for Move-to-Low. The bounds are tight for those two algorithms. The Round-Robin Algorithm has the weakest bound, which may not be tight, but the lower bound of $k + O(\sqrt{k \log n} + \log n)$ still holds. We prove that after k iterations of the Round-Robin Algorithm, the maximum height of the bins is at most $8k + O(\log n)$.

Another algorithm (see Section 4) is based on the d-choice load-balancing of balls and bins. It gives a bound of $\log \log n / \log d$ on the maximum load of a bin, the average load being 1. By increasing the number of choices to $\log n$, the maximum load can be reduced to 4.

In Section 5, we combine the load-balancing algorithms with the traditional augmenting path algorithms. We show how to find in time $O(m \log n)$ a subgraph where every node on the left has load exactly 1 and every node on the right has load at most 2. This can be generalized to finding a subgraph where every left node has load exactly k and every right node has load either k, $k+1$ or $k-1$ in time $O(km \log n)$.

Finally in Section 6 we consider a variant of the load balancing algorithm. Instead of viewing balls and bins as nodes, we view them as edges and consider the height of a ball to be the sum of loads of its two endpoints. By iteratively adding lowest potential edges, we show that in linear time it finds a subgraph with n edges where the degree of every node is at most 2.

Using Algorithm Random-Color or Move-to-Low, choosing k to be sufficiently large, we obtain in time $m \log n/\epsilon^2$ a fractional matching where every node on the left has load 1 and every node on the right has load at most $1 + \epsilon$. The augmenting path algorithms gives a near-perfect fractional matching where each right node has load within $1 \pm \epsilon$, with a better running time $O(m \log n/\epsilon)$.

3 Performance of Load Balancing Algorithms

3.1 Algorithm Move-to-Low

Lemma 1 *If algorithm Move-to-Low is allowed to run to a fixed point, the fraction of bins with load at least $k+j$ is at most $k^j \frac{k!}{(k+j)!}$; the maximum load on any bin is $O(\sqrt{k \log n} + \log n)$ above the average.*

Proof: We will compute a recurrence relation on the number of bins with load j. Observe that after equilibrium, if a ball of a certain color is in a bin of load j, all adjacent bins of that color are at least at height $j-1$. Let p_j denote the number of bins at height j or more. The total number of balls in such bins is at least jp_j. Since there are only k balls of each color, the number of different colors in these balls is at least jp_j/k. If there is a perfect matching, these throwers (colors) have at least as many neighbors all of which have load at least $j-1$.

So
$\frac{jp_j}{k} \leq p_{j-1}$ or $p_j \leq \frac{k}{j}.p_{j-1}$
Or $p_{k+j} \leq \frac{k}{k+j} p_{k+j-1}$. After simplification since $p_k \leq n$, we get $p_{k+j} \leq k^j \frac{k!}{(k+j)!} n$
If $j = O(\sqrt{k \log n} + \log n)$ this becomes less than one. □

The fraction $k^j \frac{k!}{(k+j)!}$ is remarkably close to the fraction of bins that receive $k+j$ balls when nk balls are randomly thrown into k bins, which is $\binom{kn}{k+j}(1/n)^{k+j}(1-1/n)^{nk-k-j} \approx k^{k+j}e^{-k}/(k+j)! = k^j \frac{(k/e)^k}{(k+j)!}$.

The above analysis is tight. We can construct a subgraph where $\lceil \frac{jp_j}{k} \rceil = p_{j-1}$ and none of the balls can be moved to lower heights, by iteratively creating x new throwers for each bin of load x, and then creating x bins of load $x-1$ for each new throwers.

3.2 Algorithm Random-Color

We show that the distribution of balls into bins obtained by Random Color algorithm is no worse than the distribution obtained when each ball is randomly thrown into a bin regardless of its color (we call it the **Pure-Random Algorithm**). It is known that if kn balls are randomly thrown into n bins, then the maximum bin size is $k + O(\sqrt{klogn} + \log n)$ with high probability. (More precisely, when k is constant, the maximum load is $k + O(\log n / \log \log n)$ balls; when $k > \log n$ the maximum load is $k + O(\sqrt{k \log n})$ [22].) This bound is tight for Random-Color, because it is tight for Pure-Random, and when the graph has only n edges the two processes are equivalent.

The essential observation is that since there is a perfect matching, we can associate each color i with a unique bin b_i, that is, its matched neighbor in the perfect matching. Since this is a one-to-one mapping between colors and bins, picking a random color is the same as picking a random bin. When a ball of a certain color i is chosen, the algorithm always have the option to place the ball into b_i. It will end up placing the ball into a bin with height at most that of b_i. This amounts to picking a random bin b_i and placing the new ball into a bin of height at most b_i. In this sense, we are always doing better than the random process of assigning balls to random bins.

To formalize the argument, we use the notion of "majorization" and coupling (see also [4]).

Definition 1 *Majorization: A load vector $u = (u_1, \ldots, u_n)$ of balls into bins is majorized by a load vector v, written as $u \leq v$, if for all i, the total number of balls in the i most heavily loaded bins of u is at most that of v, that is, $\forall i, \sum_{1 \leq j \leq i} u_{\pi(j)} \leq \sum_{1 \leq j \leq i} v_{\sigma(j)}$, where π and σ are permutations of $1, \ldots, n$ such that $u_{\pi(1)} \geq u_{\pi(2)} \geq \ldots \geq u_{\pi(n)}$, and $v_{\sigma(1)} \geq v_{\sigma(2)} \geq \ldots \geq v_{\sigma(n)}$.*

The intuition is that if $u \leq v$, v can be converted into u by moving balls to lower heights. We are going to use the coupling technique to compare Random-Color and Pure-Random. Here coupling means that the two considered stochastic processes are tied together (sharing the same random bits) such that each process for itself looks exactly like the original process, but at any point of time the load vector of Random-Color is majorized by that of Pure-Random.

Theorem 2 *There is a coupling between Random-Color and Pure-Random, such that the load vector obtained by Random-Color is majorized by the load vector obtained using Pure-Random.*

Proof: We prove by induction on the number of balls thrown. Let u be the load vector obtained after throwing some fixed number of balls using Random-Color, and v be the vector after throwing the same number of balls using Pure-Random. Without loss of generality, assume u is ordered, i.e. $u_1 \geq u_2 \geq \ldots \geq u_n$; so is v. By induction, we assume that $u \leq v$.

Let u' and v' denote the load vectors after throwing one more ball using the two algorithms respectively. We use the following coupling of Random-Color and Pure-Random. We choose uniformly at random a number i from 1 to n. In Random-Color, we choose the left vertex that is matched to bin x in the perfect matching, and put the ball into the least loaded bin adjacent to the vertex. In Pure-Random, we directly throw the ball into the xth bin. Note that the two bins may not be the same because bins in u and v are ordered according to the loads. It is easy to see that the probabilities for these assignments remains the same as those in the original processes. Thus coupling is well defined. We are going to prove that $u' \leq v'$.

Random-Color will always put the ball in a bin with load at most u_i. Consider the load vector obtained by adding the new ball into bin x of u, denoted by u''. It is easy to see that $u' \leq u''$. We only need to show $u'' \leq v'$, which follows a known property of majorization (see Lemma 3.4 in [2]): for any two ordered load vectors u and v, $u \leq v$ implies $u + e_i \leq v + e_i$, where e_i denotes the ith unit vector. □

3.3 Algorithm Round-Robin

Theorem 3 *After one iteration of algorithm Round-Robin, the maximum height of bins is at most* $\log n+1$. *After k iterations, the maximum height of bins is at most* $8k + O(\log n)$.

Proof: First we show that the maximum height after one round is at most $\log n + 1$. Let $T(i)$ denote the number of bins with height at least i. We will show that $T(i) \geq \sum_{j=i+1}^{h} T(j)$ where h is the maximum height after one round. We will argue that there are at least $\sum_{j=i+1}^{h} T(j)$ distinct bins with heights at least i. For each of the heights j ranging from $i+1$ through h, there are at least $T(j)$ distinct balls. This gives a total of $\sum_{j=i+1}^{h} T(j)$ balls each with a different color. For every ball of a given color at height j there is a unique bin corresponding to its assignment an optimal matching; the height of that bin is at least $j-1$ as otherwise the ball would not be at height j. Since each ball gives a distinct bin with height at least $i-1$, the recurrence follows. Since $T(h)$ is at least 1, $T(h-1)$ is also at least 1 implying $T(0) \geq 2^{h-1}$. So h is at most $\log n + 1$.

We will inductively show that after r rounds the maximum fraction of bins with height $8k+i$ is at most α^i where $\alpha < 1$ will be specified later. Assume it is true after r rounds but is not true after the $r+1$ round. This means there is a height $8(r+1)+i$ where the number of balls is at least $n\alpha^i$. Look at the lowest such height. By induction, there were at most $n\alpha^{i+8}$ balls at this height after r rounds. So at least $n\alpha^i - n\alpha^{i+8}$ new balls were added in the $r+1$th round. Also the number of balls at height $8(r+1)+j-1$ must also be at least $n\alpha^i$. So at least $n\alpha^i - n\alpha^{i+7}$ new balls were added at this height. So again as before we can argue that total number of bins with height at least $8(r+1)+i-2$ is at least $n\alpha^i - n\alpha^{i+8} + n\alpha^i - n\alpha^{i+7}$. If this is greater than $n\alpha^{i-2}$ then we get a contradiction as it gives a lower height where the induction hypothesis is not true; observe that if $i \leq 2$ this implies there are more than n balls at height $8(r+1)+i-2$ which is impossible. So we need $n\alpha^i - n\alpha^{i+8} + n\alpha^i - n\alpha^{i+7} > n\alpha^{i-2}$ or $1 - \alpha^8 + 1 - \alpha^7 > \alpha^{-2}$. This holds for α about 0.8. □

The above bound might not be tight, but the lower bound $k + O(\sqrt{k \log n} + \log n)$ still holds. Consider the graph where the ith left vertex links to all right vertices with indices equal or higher than i. In Round-

Robin, if we break the tie by making each ball go to the bin with the highest index, then the load vector is worse than the load vector obtained by Random-Color.

3.4 Graphs without Perfect Matchings

The analysis of Random-Color and Move-to-Low algorithm can be generalized to graphs without perfect matchings. The graph can even have different numbers of vertices on left and right.

Define the minimum expansion of a bipartite graph to be $min_{V \subset L} \frac{|N(V)|}{|V|}$. Suppose the minimum expansion of a bipartite graph G is c ($c \leq 1$; equality holds only when G has a perfect matching), then the distribution of bin heights obtained by Random-Color is no worse than that obtained by Pure-Random on cn bins. This is because the probability that a ball falls into one of the x heaviest loaded bins in Random Color is at most x/cn, while in Pure-Random, this probability is exactly x/cn. By a similar coupling argument as theorem 2, the load vector of Random-Color is majorized by that of Pure-Random. Therefore, with high probability, the maximum bin size is $k/c + O(\sqrt{k \log n/c} + \log n)$. Note that an optimal fractional matching, where all vertices on the left have load 1 and the maximum load on a right vertex is minimized, has the maximum bin size k/c. So Random-Color still gives a near-perfect matching.

Theorem 4 *There is a coupling between Random-Color and Pure-Random on cn bins such that the load vector obtained by Random-Color is majorized by the load vector of Pure-Random.*

Proof: We prove the theorem by induction on the number of balls thrown. Assume u is the (ordered) load vector using Random-Color after throwing some balls; v is the vector of Pure-Random. $u \leq v$ by induction. Let u' and v' be the load vectors after throwing one more ball. We are going to show $u' \leq v'$.

We refer to a set of bins with the same height in a load vector as a *plateau*. A load vector consists of a sequence of plateaus with decreasing heights. Consider the first cn bins of u. Let the number of bins in the ith plateau of $u[1..cn]$ be x_i. Let y_i be the prefix sum of x_i, i.e. $y_i = \sum_{j=1}^{i} x_j$ ($y_i \leq cn$).

We claim that given u, the probability that a ball is thrown into any bin in the first i plateaus is at most y_i/cn. The reason is as follows. Let X be the set of left vertices (colors) such that any ball in X will be put in the first i plateaus by Random-Color. Assume $|X| > y_i/c$, then $|N(X)| > y_i$ because the minimum expansion is c. This contradicts with the definition of X, because there is at least one bin in $N(X)$ with height less than the height of plateau i, so Random-Color will put the ball there. Therefore, $|X| \leq y_i/c$. The probability of choosing any color is exactly $1/n$, so the probability of a ball thrown into the first i plateaus is at most y_i/cn.

We define a third stochastic process. It looks at the first cn bins of u and places the ball in plateau i with probability x_i/cn. It makes no difference which specific bin within plateau i is chosen, so we assume the process always chooses the last bin y_i. Denote by u'' the load vector after throwing one more ball to u by this process. $u' \leq u''$ because the process only moves the ball toward a higher plateau compared to Random Color.

Define another process corresponding to Pure-Random. It looks at v and put the new ball to bin y_i if Pure-Random put it in some bin in $[y_{i-1}+1, y_i]$. Denote by v'' the load vector obtained by this process. $v'' \leq v'$ because the process only moves the ball downward. The probability that the ball is put in bin y_i is x_i/cn. (Note the y_is are with respect to the load vector u.)

Both of the new processes put the ball in bin y_i with probability x_i/cn. We couple the two processes so that they throw the new ball in the same bin. It is proved in [2] that if $u \leq v$, and the new ball is thrown into the same bin, then $u'' \leq v''$. Therefore, $u' \leq u'' \leq v'' \leq v'$. □

Similar bounds holds for Move-to-Low on graphs without perfect matchings. The analysis is similar to Lemma 1, except that the derivation is $p_{k/c+j} \leq \frac{k/c}{k/c+j} p_{k/c+j-1}$ instead of $p_{k+j} \leq \frac{k}{k+j} p_{k+j-1}$.

We can further generalize the algorithm/analysis to arbitrary demands/capacities of vertices. Each vertex v has a demand/capacity $c(v)$, and we look for a (fractional) matching where the total weight of a vertex in the matching is close to $c(v)$. When pick a ball, we pick v with probability $c(v)/\sum_u c(u)$; when choose the least loaded bin, normalize the load with respect to its capacity.

4 Using d-Choice Load Balancing

The above algorithms produce a maximum load of at least $O(\log n/\log\log n)$ with a total of n balls. We now show how ideas based on d-choice hashing can be used to find a better distribution. It is known that while throwing n balls into n bins if each ball picks $d \geq 2$ bins at random and is inserted into the least loaded of the d bins, then the maximum load of any bin is at most $\log\log n/\log d + O(1)$. We now show an algorithm that produces a subset of the edges (allowing repetitions) of the bipartite graph so that the total degree on each side is n and the maximum degree is $\log\log n/\log d + O(1)$ in time $O(md)$. In particular for $d = O(\log n)$ the maximum load is 4.

Intuitively - if we pick $d \geq 2$ random vertices on the left side and throw a ball into the least loaded vertex among all their neighbors then this is similar to the d-choice load balancing. The color of the ball corresponds to one of the d vertices whose neighbor was chosen (breaking ties arbitrarily). If this is done n times the maximum load of any bin on the right side is at most $\log\log n/\log d + O(1)$. However, there is no guarantee of a low load on the vertices on the left; that is the maximum number of balls of a certain color. To remedy this we modify the algorithm slightly: Instead of picking d vertices on the left we pick $2d-1$ and consider the d of these that have lowest loads. Then we look at the set of their neighbors on the right and add a ball into the least loaded bin. Again the color of the ball is the same as the vertex whose neighbor was chosen. Essentially if the $2d-1$ chosen left-vertices have loads $u_1, u_2, .., u_{2d-1}$ and their matched neighbors in a perfect matching have loads $v_1, v_2, .., v_{2d-1}$, then we find an edge with load on the left at most the median value of u_i's on load on the right at most the median value of v_i's. This is like throwing n balls into n bins where each ball picks $2d-1$ bins at random and is inserted into one of the d least loaded of the $2d-1$ chosen bins.

Theorem 5 *If n balls are inserted by the above process, we get a subset of the edges (allowing repetitions so it is a multiset) of the bipartite graph so that the total degree on each side is n and the maximum degree is $\log\log n/\log d + O(1)$ with high probability.*

Theorem 6 *If n balls are inserted by above process except that we choose $d = O(\log n)$ nodes for each ball, then we get a subset of the edges (allowing repetitions so it is a multiset) of the bipartite graph so that the total degree on each side is n and the maximum degree is 4 with high probability.*

Lemma 7 *If n balls are thrown into n bins where each ball picks $2d-1$ bins at random and is inserted arbitrarily into one of the d least loaded of the $2d-1$ chosen bins, then the maximum load of any bin is at most $\log\log n/\log d + O(1)$ with high probability. This is true even if instead of inserting a ball in one of the d least loaded of the $2d-1$ chosen bins, it is inserted into any other bin whose load does not exceed the median load the $2d-1$ chosen bins,*

Proof: We use the layered induction technique similar to the one used to bound the load of bins in the two choice hashing [2]. Let p_i denote an upper bound on the fraction of bins with load at least i at the end of the process. We will derive a recurrence relation on p_i. Let us compute the probability that during an insert the new ball lands at a height of $i+1$ or higher. For this to happen at least d of the bins chosen must have height i or greater; otherwise each of the d least loaded of the $2d-1$ bins have load less than i. Probability that this happens is at most $\binom{2d-1}{d} p_i^d \leq 2^{2d} p_i^d = (4p_i)^d$. So the expected number of balls that fall at height $i+1$ or higher is at most $(4p_i)^d n$. This is also a bound on the expected number of bins with load at least $i+1$. So given p_i, the expected value of p_{i+1} is at most $(4p_i)^d$. This can be converted into a high probability bound – probability at least $1 - 1/n^c$ – by using Chernoff bounds as long as $(4p_i)^d n \geq c \log n$ where c is a large enough constant.

This implies, given p_i, if $(4p_i)^d n \geq c \log n$ then with high probability $p_{i+1} \leq (4p_i)^d$, which implies $16p_{i+1} \leq (16p_i)^d$. Now, $p_{32} \leq 1/32$ as at most $1/32$ fraction the bins can have load 32 or higher. Using the recurrence we get $p_{i+32} \leq 1/2^{d^i}$ as long as $1/2^{d^i} \geq c \log n/n$. Let k denote the highest value of $i + 32$ for which this holds; $k = \log\log n/\log d + c'$ where c' is some constant (possibly negative). Now we know that $(4p_k)^d \leq c \log n/n$. We will show that with high probability there is no bin with load $k + 2$. Again Chernoff bounds can be used to show that with high probability $p_{k+1} \leq 2c \log n/n$. Now we argue that the probability that a bin has load $k + 2$ is at most $n(8c \log n/n)^d$ using a simple union bound on the probability that any of the ball throws lands at height $k + 2$. So the probability that there is a ball at height $32 + \log\log n/\log d + c' + 2$ is at most $n(8c \log n/n)^d + n/n^c \leq 1/n^{\Omega(1)}$. The conditioning arguments can be made more precise along the lines of the proof in [2]: Essentially let E_i denote the event that the fraction of bins with height at least $i + 32$ is at most the bound p_{i+32} computed above. Then E_0 is true; probability that E_i holds and E_{i+1} does not is negligible which recursively implies that each E_i holds with high probability except for the sum of the negligible probabilities. □

Given lemma 7 it is easy to argue that the loads of the vertices of the left side and right side grow according to the process described. For the right side vertices, since $2d - 1$ vertices are chosen at random from the left side, this is equivalent to choosing $2d - 1$ vertices at random on the right side in the optimal matching. After discarding $d - 1$ of them we pick a vertex with load that is no more than that of the least loaded of the d remaining right side bins. So again the arguments in proof of lemma 7 go through; the only difference being instead of placing the ball in one of the d least loaded of the $2d - 1$ chosen bins, it is placed into some bin with possibly lower load but no higher.

Theorem 5 can also be proven similarly: The fraction of nodes on any side with load at least 3 is at most $1/3$ at any time. If $d = O(\log n)$ nodes are chosen at random then, with high probability, $1 - 1/n^2$, the median load is at most 3. Since we can always find an edge whose load is at most the median load on both the left and the right side, we can with high probability, add a ball along an edge so that after the addition the loads are at most 4.

5 Combining Load Balancing with Augmenting Paths

Classical perfect matching algorithms for bipartite graphs leverage max flow algorithms based on augmenting paths. Now we combine the above load balancing algorithms with augmenting paths, and get efficient algorithms to find near-perfect fractional matchings.

Definition 2 *An s-near matching in a bipartite graph G is a multi-set of edges in G, such that the load of any left vertex in the matching is exactly s, while the load of any right vertex is in $[s - 1, s + 1]$.*

In this section, we show an algorithm finding k-near matching in time $O(m \log^2 n + mk \log n)$.

5.1 Finding 1-near matchings

We start with 1-near matching. Recall that the *residual graph* with respect to a subgraph is to flip the directions of all edges in the subgraph (initially all edges are directed from the left side to the right side); *an augmenting path* is a simple path in the residual graph. Given a subgraph where the load of any left vertex is 1, its residual graph has the following property.

Lemma 8 *For every right vertex u of height $h \geq 3$, there exists an augmenting path of length at most $2 \log n$ from u to some right vertex v of height at most $h - 2$.*

Proof: Let d be the length of shortest such augmenting path; N_p be the number of the bins reachable from u within $2p$ steps ($2p < d$). The total number of balls in those bins is at least $(h - 1)N_p$, otherwise we have founded a bin of height $h - 2$ or less. Those balls correspond to $(h - 1)N_p$ distinct left vertices, the size of whose neighborhood is at least $(h - 1)N_p$ because there exists a perfect matching. This whole neighborhood is reachable from u in another 2 steps. Therefore, $N_{p+1} \geq (h-1)N_p \geq (h-1)^{p+1}$. For $h \geq 3$, $(h-1)^{d/2+1} \geq 2^{d/2}$. Because the total number of balls in the system is n, we have $2^{d/2} \leq n$, or $d \leq 2 \log n$. $\square$

Moving one ball from u along the augmenting path to v decreases the height of v by one. Given any subgraph, we can eliminate all highest bins by such moves; do so iteratively until the maximum height is 2. In the end, the load of any left vertex in the subgraph remains 1, and the load of any right vertex is at most 2. Thus we get a 1-near matching.

We plug in the Edmonds-Karp max-flow algorithm to find the augmenting paths ([CLR01 26.2]).

1-near matching Algorithm:

1. each left vertex throws a ball to its least loaded adjacent right bin in an arbitrary order;

2. Let h_0 be the maximum bin height after step 1. For $h = h_0$ to 3, use the Edmonds-Karp algorithm to eliminate bins of height h.

Edmonds-Karp algorithm:

1. Let any bin of height h be a source, with capacity 1; any bin of height $h' \leq h-2$ be a sink, with capacity $h - h' - 1$;

2. while there exists a source do

2-1. find the length of the shortest augmenting path from any source to any sink, denoted by d;

2-2. find a maximal set of edge-disjoint source-sink paths of length d, under the constraint of sink capacity. For each augmenting path selected, flip the edges, i.e. move the ball from the source bin to the sink bin.

The algorithm is based on the following property: if we augment along the shortest path, then the shortest distance from source to any node in the residual graph increase monotonically. See [8] Ch26.2 for the proof. This implies in out setting:

Lemma 9 *In the Edmonds-Karp algorithm, d increases monotonically.*

We analyze the time complexity of the above algorithm. Using the standard implementation of Edmonds-Karp algorithm, each iteration takes time $O(m)$. (Use breadth-first search to decide the shortest distance d takes time m. Then use depth-first search to find augmenting paths of length d; whenever we encounter a dead node that does not lead to any sink, label this node so that it is not expanded again in this iteration. Such useless expansion is at most m. All other expansions contribute to augmenting paths, which is at most $n * \log n$ over all iterations, amortized cost per iteration being $n \log n/m = O(m)$.)

Combining Lemma 8 and 9, in one execution of Edmonds-Karp algorithm, the number of iterations is at most $2 \log n$, so the running time is $O(m \log n)$. The algorithm is called h_0 times; by Theorem 3, $h_0 \leq 8 + \log n$. Therefore, the total running time is $O(h_0 m \log n) = O(m \log^2 n)$.

Theorem 10 *A 1-near matching can be found in time $O(m \log^2 n)$.*

5.2 Finding k-near matchings

The above algorithm can be generalized to get an algorithm to find a k-near matching in $O(mk \log n)$ time, where the load of each left vertex in the matching is k, and the load of each right vertex is between $k \pm 1$.

The idea is to recursively double the balls in each bin, and move the balls by finding augmenting paths of length at most $klogn$. We start with $k = 1$ and get a 1-near matching using the above algorithm. Assume we have a $\frac{k}{2}$-near matching, now double each edge, and we get a subgraph where each left vertex has load k and any right vertex has load $k \pm 2$. To convert the matching into a k-near matching, all we need to do is to eliminate bins of height $k+2$ and $k-2$ by finding augmenting paths to bins of height k. We can bound the length of shortest augmenting path similar to Lemma 8.

Lemma 11 *For every bin u of height $k+2$, there exists an augmenting path of length $O(k \log n)$ from u to some bin of height at most k. For every bin u of height $k-2$, there exists an augmenting path of length $O(k \log n)$ from some bin of height at least k to u.*

Proof: Let d be the length of the shortest such augmenting path; N_p be the number of distinct balls in the bins reachable from u within $2p$ steps ($2p < d$). (Two balls are distinct if they correspond to different left vertices.) In another 2 steps, u can reach N_p bins; all those bins have height at least $k+1$, so the total number of balls in those bins is at least $N_p(1+k)$, and the number of distinct balls is $n_{p+1} \geq N_p(1+\frac{1}{k}) \geq (1+\frac{1}{k})^{p+1}$. Because the total number of balls in the system is n, we have $(1+\frac{1}{k})^{d/2+1} \leq n$, or $d \leq 2\log_{1+1/k} n = O(k \log n)$.

Similarly we can prove that for every bin u of height $k-2$, the number of bins that can reach u within $2p$ steps is at least $(1+\frac{1}{k-1})^{p+1}$, so there exists an augmenting path of length $O(k \log n)$ from some bin of height at least k to u. □

Running Edmonds-Karp algorithm to eliminate all bins with height $k+2$ and $k-2$ takes time $O(mk \log n)$, and results in a k-near matching. We need to double the number of balls per left vertex from 1 up to k. The running time of each phase increases geometrically, so the total running time is dominated by the last phase, which is $O(mk \log n)$. To start with, we need to find a 1-near matching, so the total time is $O(m \log^2 n + mk \log n)$.

Theorem 12 *A k-near matching can be found in time $O(m \log^2 n + mk \log n)$.*

6 Edge Load Balancing

In the previous sections, we think of vertices as bins and balls, and do load balancing on vertices. Now we consider load balancing on edges instead. Firstly, we need to define the "load" of an edge.

Definition 3 *In a given subgraph M, let $d_M(u)$ be the load of vertex u in the subgraph, also referred to as the potential of u. The* **potential of an edge** *$e = (u, v)$, denoted by $\phi_M(e)$, is the sum of the loads of its two endpoints, i.e. $\phi_M(e) = d_M(u) + d_M(v)$. The* **potential of the subgraph** *M, denoted by $\phi(M)$, is the sum of potential of all edges in the subgraph, i.e. $\phi(M) = \sum_{e \in M} \phi_M(e)$.*

It is easy to see that $\phi(M) = \sum_{v \in V} d_M(v)^2$ (1)

In a bipartite graph with $2n$ nodes, among all the subgraphs with kn edges, the minimum potential $nk * 2k = 2nk^2$, achieved by using k copies of a perfect matching.

We consider the following **Lowest Potential Edge Algorithm**: iteratively add kn edges, each time adding an edge with the lowest potential in the current subgraph. The algorithm takes time nkd, where d is the maximum degree in the graph. We prove that it gives a subgraph of potential $2nk^2 + 2nk$, and each vertex has load at most $2k$.

We divide the kn into k consecutive rounds, each round having n edges. When we need to add an edge in round i, there always exists an edge with potential at most $2i-1$. (Consider the set of edges in the perfect matching. The total potential of the n edges is less than $2ni$; according to the pigeonhole principle, at least one edge has potential less than $2i$.) The potential of the new edge is at most $2i+1$, and it increases the potential of $2i-1$ adjacent edges, so it increases the potential of the subgraph by at most $4i$. The total increase of potential in round i is at most $4ni$, and the potential of the final subgraph is at most $2nk(k+1)$.

We can also perform edge moves similar to Move-to-Low algorithm: starts from any subgraph, and keeps moving an edge e to a position with potential at most $\phi(e)-3$ until no such moves can be performed. When the algorithm is run to a fixed point, the potential of the matching is also at most $2kn(k+1)$. The running time is similar to Lowest Potential Edge Algorithm, if we start from a good subgraph, say the subgraph obtained by 3-choice algorithm in Section 4.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.

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

[3] Y. Azar and A. Litichevskey. "Maximizing throughput in multi-queue switches." In *Proceedings of 12th ESA Conference,* 2004.

[4] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. "Balanced allocations: The heavily loaded case." In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC),* pp. 745–754, 2000.

[5] D.P. Bertsekas. "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem." *Annals of Operations Research,* 14(1988):105–123.

[6] D.P. Bertsekas. "Auction algorithms for network flow problems: A tutorial introduction." *Computational Optimization and Applications,* 1(1992):7–66.

[7] B.V. Cherkassky, A.V. Goldberg, P. Martin, J.C. Setubal, and J. Stolfi. "Augment or push: a computational study of bipartite matching and unit-capacity flow algorithms." *ACM J. Exp. Algorithmics,* 3(1998).

[8] T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms.* Second Edition. MIT Press, 2001.

[9] A. Czumaj, C. Riley, and C. Scheideler. "Perfectly Balanced Allocation." In *Proceedings of the 7th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM'03),* pp. 240–251, 2003.

[10] A. Czumaj and V. Stemann. "Randomized allocation processes." *Random Structures and Algorithms* 18(2001):297–331.

[11] T. Feder and R. Motwani. "Clique Partitions, Graph Compression and Speeding-up Algorithms." *Journal of Computer and System Sciences,* 51(1995):261–272.

[12] J. Hopcroft and R. Karp. "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs." *SIAM Journal on Computing,* 2(1973):225–231.

[13] B. Kalyanasundaram and K.R. Pruhs. "An optimal deterministic algorithm for online b-matching." *Theoretical Computer Science,* 233(2000):319–325.

[14] R.M. Karp, U.V. Vazirani, and V.V. Vazirani. "An optimal algorithm for online bipartite matching." In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing,* 1990.

[15] H.W. Kuhn. "The Hungarian method for the assignment problem." *Naval Res. Logist. Quart.,* 2(1955):83–97.

[16] L. Lovasz and M.D. Plummer. *Matching Theory.* Annals of Discrete Mathematics. North Holland, 1986.

[17] A. Mehta, A. Saberi, U.V. Vazirani and V.V. Vazirani. "AdWords and Generalized On-line Matching." Manuscript, 2005.

[18] S. Micali and V. Vazirani. "An $O(E\sqrt{V})$ algorithm for finding maximum matchings in general graphs." In *Proceedings of the 21st IEEE Symposium on the Foundations of Computer Science,* pp. 17–27, 1980.

[19] M. Mitzenmacher, A. Richa, and R. Sitaraman "The Power of Two Random Choices: A Survey of Techniques and Results." Book chapter in *Handbook of Randomized Computing,* Volume 1, edited by P. Pardalos, S. Rajasekaran, and J. Rolim, pp. 255–312, 2001.

[20] R. Motwani and P. Raghavan. *Randomized Algorithm.* Cambridge University Press, 1995.

[21] R. Panigrahy. Efficient Hashing with Lookups in Two Memory Accesses. In *Proceedings of SODA,* 2005.

[22] M. Raab and A. Steger. "Balls into bins – a simple and tight analysis." In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science,* LNCS volume 1518, pp. 159–170, 1998.

[23] B. Vöcking. "How asymmetry helps load balancing." In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science.* pp. 131–140, 1999.