The Random Graph Threshold for k-orientiability and a Fast Algorithm for Optimal Multiple-Choice Allocation

Julie Anne Cain* Peter Sanders† Nick Wormald‡

Abstract

We investigate a linear time greedy algorithm for the following load balancing problem: Assign m balls to n bins such that the maximum occupancy is minimized. Each ball can be placed into one of *two* randomly choosen bins. This problem is closely related to the problem of orienting the edges of an undirected graph to obtain a directed graph with minimum in-degree. Using differential equation methods, we derive thresholds for the solution quality achieved by our algorithm. Since these thresholds coincide with lower bounds for the achievable solution quality, this proves the optimality of our algorithm (as $n \to \infty$, in a probabilistic sense) and establishes the thresholds for k-orientability of random graphs. This proves an assertion of Karp and Saks.

1 Introduction

Randomized load balancing is a simple and powerful technique. Many of its applications can be modelled using a balls into bins terminology — m balls are to be allocated into n bins. Let ℓ_i denote the number of balls allocated to bin i. Good load balance means that the maximum occupancy $L := \max\{\ell_i : i \in \{1,\ldots,n\}\}$ is small. The simplest variant of this model, where each balls has a single random choice, has a problem however. Only for fairly large m, we are beginning to observe good load balance. For example $m = \Omega(n \ln n)$ balls are needed to have $\mathbf{E}[L] = \mathcal{O}(m/n)$. Interestingly, the situation improves radically if the balls are given just *two* choices. There are simple linear time algorithms that achieve $L = \mathcal{O}(m/n)$ [15, 10, 16] or $L = m/n + \mathcal{O}(\log\log n)$ [1, 2]. Optimal allocations can be computed in polynomial time [6] using maximum flow computations and with high probability achieve $L = \lceil m/n \rceil$ or $L = \lceil m/n \rceil + 1$ [21]. Empirically, there are sharp thresholds between these two cases. See Figure 1 for the relation between m/n and L. Hence, there seems to be a tradeoff between simplicity and speed of the method on the one hand and the quality of load balancing on the other hand. We close this gap by giving simple linear time algorithms that compute an optimal schedule asymptotically almost surely (a.a.s.) for random allocation.

Many algorithms for two-choice allocation are best understood using a graph model. The bins are vertices and a ball that can be placed into bins u and v forms an undirected edge $\{u, v\}$. The load balancing problem then amounts to *orienting* the edges of the allocation graph G such that its maximum in-degree is small. In other words, given the undirected allocation graph $G = (V, E)$ we are computing a directed output graph $G' = (V, E')$.

Several allocation algorithms and their analysis are intimately connected to the concept of *cores* of a graph. The *k-core* of a graph is the maximum vertex induced subgraph with minimum degree $\geq k$. Indeed Czumaj and Stemann [8] give a linear time algorithm achieving maximum load $\mathcal{O}(m/n)$ based on a well known algorithm for computing all cores: Repeatedly choose a vertex v with minimum degree and remove it from the graph, thus *committing* all its incident edges (balls) to be allocated to vertex (bin) v, i.e., all edges of the form $\{v, w\}$ in G are moved to the output graph G' using the direction (w, v). Unfortunately, this is not optimal. In particular, as $m/n \to \infty$, L will approach $2m/n$ (see also the end of Section 2).

Our algorithms can be viewed as a variation of the same basic approach. The algorithms first *guess* the optimal L. This is easy since with high probability there are only two likeley values — $L = \lceil m/n \rceil$ or $L = \lceil m/n \rceil + 1$. We can simply start with $L = \lceil m/n \rceil$. If the algorithm fails, we increment L until a solution is found. The min-degree rule is used exactly for the "safe" vertices with degree $\leq L$. The application of the min-degree rule will first reduce the graph to its $(L + 1)$-core. After this we are stuck. We therefore need an additional, lower priority rule that gets things going again. This rule picks a single edge $\{u, w\}$ in G, according to some rule, and moves it to G' choosing an appropriate direction.

*U. Melbourne, Australia, J.Cain@ms.unimelb.edu.au

†Universität Karlsruhe, Germany, sanders@ira.uka.de

‡University of Waterloo, Canada, nwormald@uwaterloo.ca

We also call this process “comitting an edge”. After committing an edge, the mindegree rule is again tried. But now we have to be a bit careful — only those nodes qualify that have degree in G plus in-degree in $G' \leq L$.

In Section 2 we analyze the *Selfless algorithm* originally proposed in [20]. The *load-degree* of a vertex v is its degree in G plus twice its in-degree in G'. To understand the idea behind this definition, note that the load-degree of v is twice the expected in-degree of v in G' if we were to orient all remaining edges in G randomly. The algorithm chooses a random vertex v with minimum load degree, then a random edge $\{v, w\}$ incident to v, and moves the edge to G' using direction (w, v). We show that a.a.s. the threshold value where the L found by Selfless switches from $\lceil m/n \rceil$ to $\lceil m/n \rceil + 1$ coincides with the threshold for the $(\lceil m/n \rceil + 1)$-core to have average degree $2 \lceil m/n \rceil$. From this it follows that Selfless is indeed optimal.

We say that a graph G is *k-orientable* if there exists an orientation of the edges of G in which no vertex has in-degree greater than k. As we show in Section 2, our analysis of Selfless determines the threshold value for non-k-orientability of the common random graph models, and equivalently for the appearance of a subgraph of average degree at least $2k$. This is the first proof of an assertion of Karp and Saks. The case average degree 3 was announced to be solved by Molloy [18].

Independently, Fernholz and Ramachandran [12] have also obtained the general result. They use a different algorithm *EDR* (excess degree reduction). EDR uses linear recursion and always eliminates a node v with minimum degree d. If $d > 2L$ the EDR fails.
If $L < d \leq 2L$, EDR replaces $d - L$ paths of the form (u, v, w) by *bypass edges* $\{u, w\}$ and then applies the final case: Orient all remaining $\leq L$ edges incident to v towards v.

Section 3 discusses implementation details of algorithm Selfless. In Section 4 we report experiments which indicate that Selfless (and EDR) also performs well for small n. We also present a similar, slightly simpler algorithm that also performs very well and that might turn out to be easier to analyze. Section 5 summarizes the results and states some questions for future research.

1.1 Some Applications

A well known application for multiple choice applications are parallel disk servers (see [21] and the references therein). Logical data blocks are randomly mapped to two out of n identical disks. In the I/O model for external computing [22], accessing one block from each disk takes one I/O step. Hence, retrieving any m different data blocks which have two possible locations is equivalent to allocating m blocks (balls) with two random choices to n disks (bins). L will be the number of I/O steps needed to fetch the data. The duplicate allocation gives us some degree of fault tolerance and allows very good load balancing for arbitrary access patterns. Fast algorithms for solving this disk scheduling problem might become even more important in a currently emerging variant of this setting: Solid state disks contain NAND-flash memory chips. Similar to a hard disk, a NAND flash is accessed in a blocked manner and has a fairly large access latency (which is much smaller however than for hard disks). Since these chips are much smaller than hard disks with respect to size, price, and capacity, solid state disks often have a large number of chips. Hence, we have to solve larger load balancing problems in less time resulting in a requirement for very fast scheduling algorithms.

Dietzfelbinger and Weidling [11] have developed a highly space efficient hash table data structure with worst case constant access time based on allocating table entries to one out of two buckets of capacity k. Our results are useful for constructing static hash tables using this approch: Given m objects to store, we first we use the threshold for k-orientability to allocate a table just big enough to accommodate all m objects. Then the Selfless algorithm is used to allocate the objects. If this fails, we retry with fresh hash functions until the construction succeeds.

1.2 More Related Work

Refer to [17] for a survey of further results on multiple choice allocation. A survey more oriented to applications can be found in [20]. Czumaj et al. show that for $m > cn \log n$ for some appropriate constant c, $L = \lceil m/n \rceil$ can be achieved with high probabilty. They give a randomized local search algorithm which alway converges to an optimal solution and has polynomial expected running time.

2 The Selfless Algorithm

To state the threshold results for k-cores in random graphs we need some notation. For $k \geq 0$ integer and λ a positive real, define

$$f_k(\lambda) = e^{\lambda} - \sum_{i=0}^{k-1} \frac{\lambda^i}{i!} = \sum_{i \geq k} \frac{\lambda^i}{i!}.$$

Let $h_k(\mu) = \frac{\mu}{e^{-\mu} f_{k-1}(\mu)}$ and define

$$c_k = \inf\{h_k(\mu) : \mu > 0\}.$$

For $c > c_k$ it can be checked that the equation $h_k(\mu) = c$ has two positive roots (and just one for $c = c_k$). Define $\mu_{k,c}$ to be the larger one. In [19] the following theorem was obtained, in a more precise form, though the part only on number of edges was only implicit in the proofs. (It was stated explicitly and rederived in [5].)

We use the abbreviation u.a.r. for uniformly at random. An event A_n defined on a family of probability spaces indexed by n holds *asymptotically almost surely* (a.a.s.) if $\mathbf{P}(A_n) \to 1$ as $n \to \infty$.

Theorem 2.1. *Let $c > 0$ and integer $k \geq 3$ be fixed. Suppose that $m \sim cn/2$, and $G \in \mathcal{G}(n, m)$. For $c < c_k$, G has empty k-core a.a.s. For $c > c_k$, the k-core of G a.a.s. has $e^{-\mu_{k,c}} f_k(\mu_{k,c}) n(1 + o(1))$ vertices and $\frac{1}{2}\mu_{k,c} e^{-\mu_{k,c}} f_{k-1}(\mu_{k,c}) n(1 + o(1))$ edges.*

Now define $\bar{\mu}_k$ to be the positive solution of $\bar{\mu}_k f_k(\bar{\mu}_k)/f_{k+1}(\bar{\mu}_k) = 2k$. By Theorem 2.1, this is the value of $\mu_{k+1,c}$ for the threshold of c when the $(k+1)$-core first reaches average degree $2k$. The value of this c is then

$$\rho_k := h_{k+1}(\bar{\mu}_k).$$

We need to explain the algorithm a little more. At each point, the graph has some edges oriented, and some unoriented. The load-degree of a vertex is equal to the number of incidences it has with unoriented edges plus twice its in-degree (number of edges directed towards it). Let $V_{d,j}$ denote the set of vertices with load-degree d and in-degree j. It is convenient to assign to each vertex a *priority* that is a function of load-degree and in-degree. If v is a vertex in $V_{d,j}$ with $d - j \leq k$ and $d-2j > 0$, then the remaining $d-2j$ edges incident with v may be directed to v, after which v has load-degree j' with

$$j' = j + (d - 2j) = d - j \leq k.$$

Such vertices are treated as vertices with the highest priority for selection in the algorithm, and will be called *priority 1 vertices*. When there are no priority 1 vertices, a vertex of minimum load-degree is selected.

Note that the algorithm gives the same result, in terms of success versus failure, whether all edges are similtaneously directed to priority 1 vertices, or they are randomly chosen one at a time. We take the latter approach for purposes of the "step-by-step" method of analysis. So we now (re-)define the algorithm Selfless(k) is defined as follows, where *allocate to v* means "select an undirected edge incident with v u.a.r. and direct this edge towards v."

Selfless(k): If there is a priority 1 vertex, select one such vertex u.a.r. and allocate to it. Otherwise, if there is a vertex of load-degree less than $2k + 1$ and at least one incident undirected edge, select one such vertex of minimum load-degree u.a.r. and allocate to it. Otherwise, stop.

When the algorithm stops, either all edges are oriented, in which case the maximum load is at most k, or some are left unoriented, which constitutes failure to achieve the desired maximum load.

Theorem 2.2. *Let $\epsilon > 0$ and integer $k \geq 2$ be fixed. Suppose that n vertices are given, that $m = \rho n/2$ where $\rho = \rho(n) < \rho_k - \epsilon$, and let M be a random multigraph in which the two ends of each of its m edges are chosen u.a.r. as a pair of distinct vertices (with replacement). Then Selfless(k) applied to M a.a.s. orients all edges such that maximum indegree is at most k.*

Note that, if $\rho > \rho_k + \epsilon$, then Theorem 2.1 implies that the $(k + 1)$-core a.a.s. has average degree greater than $2k$. In this case it cannot be k-oriented by any algorithm, and thus ρ_k determines the threshold for k-orientability of the random multigraph, and simultaneously the theshold at which Selfless(k) passes from a.a.s. succeeding (with maximum load k) to a.a.s. failing. Returning to the terminology of Section 1, $n\rho_L/2$ is the asymptotic threshold of the number of edges (twice the number of balls) to obtain a given L. The conclusions also hold for $k = 1$ by well known properties of cycles in the random graph. Table 1 gives numeric values of $\rho_L/2$.

Bollobás and Frieze [3] and Chvátal [7] used the following model of random graphs. Take a set of $2m$ balls arranged in m pairs and throw each ball sequentially into one of n buckets chosen u.a.r. Call this an *allocation* of $2m$ balls into n buckets. From each allocation we obtain a pseudograph, (perhaps containing loops and multiple edges), with each bucket representing a vertex, each ball a point in a vertex and each pair of balls forming an edge. Let $\mathcal{M}(n, m)$ denote the probability space of pseudographs which result. We use this model, as in [5], and prove the following.

Theorem 2.3. *Theorem 2.2 holds if M is replaced by a random pseudograph with the distribution of $\mathcal{M}(n, m)$.*

Note that $\mathcal{M}(n, m)$ is not a uniform probability space of pseudographs. However, it is clear that, restricted to simple graphs (i.e. with no loops or multiple edges), it reduces to $\mathcal{G}(n, m)$, the uniform probability space of simple graphs with n vertices and m edges. Also, restricted to pseudographs with no loops, it gives the random multigraph M of Theorem 2.2.

Table 1: The thresholds $\rho_L/2$ relating the maximum occupancy L to the number m of edges.

L	1	2	3	4	5	6	7	8	9	10	11
$m/n \le$	0.5	1.794	2.877	3.921	4.947	5.964	6.975	7.982	8.987	9.991	10.993

It is easy to compute (and was shown in [7]) that for $M \in \mathcal{M}(n,m)$ and $c = 2m/n$, the probability that M is simple is asymptotic to $\binom{N}{m} m! 2^m/n^{2m} \sim \exp(-c/2 - c^2/4)$. It follows that anything true a.a.s. for pseudographs in $\mathcal{M}(n,m)$ is true a.a.s. for the other two models just mentioned.

From this we have two conclusions. Firstly, to prove Theorem 2.2 it is enough to prove Theorem 2.3.

Secondly, the same result holds if M is replaced by $G \in \mathcal{G}(n,m)$. This proves the following result, for which an incomplete proof by Karp and Saks is referred to in [14].

Theorem 2.4. *Let $\epsilon > 0$ and integer $k \ge 1$ be fixed, and $\rho = \rho(n)$. For $m = \rho n/2$, $G \in \mathcal{G}(n,m)$ is a.a.s. k-orientable if $\rho < \rho_k - \epsilon$ for all n, and not if $\rho > \rho_k + \epsilon$ for all n.*

It is well know, for example from Hakimi's theorem in [13], that a graph (or pseudograph) G is k-orientable if and only if G contains no subgraph with average degree greater than $2k$. So the theorem above determines the threshold for this property.

We finish our statements of results with some asymptotic observations.

Simple analysis shows that, for large k, firstly $\bar{\mu}_k > k$ and then $\bar{\mu}_k = 2k - O(\sqrt{k})$. This implies that $\rho_k = 2k - (2/e + o(1))^k$ as $k \to \infty$. This gives an approximation of what density graph Selfless(k) succeeds for, when k is large.

The simple algorithm in [8], mentioned in Section 1, that repeatedly orients all edges to the lowest degree vertex, must naturally give load at least j, if the j-core exists. As shown in [19, eqn. (1.3)], $c_j = j + \sqrt{j \log j} + O(\log j) \sim j$. Thus, by Theorem 2.1, the maximum load resulting from this algorithm when $c \sim \rho_k$ is asymptotically $\rho_k \sim 2k$ as $k \to \infty$. (The rate of k going to infinity here is arbitrarily slow, independent of n.) Thus, for large k the load achieved by this algorithm is approximately double the optimal load.

The proof of Theorem 2.3 occupies most of the PhD thesis of the first author [4] and will appear in the long version of this paper. It is available in the companion document to this paper, at `http://www.math.uwaterloo.ca/~nwormald/lbalappendix.pdf`, which also gives the proof of the relevant result from [5], i.e. Theorem 2.1.

Proof of Theorem 2.3. (Sketch) Consider the algorithm applied to a random pseudograph $M_0 \in \mathcal{M}(n_0, m_0)$ with $m < (\rho_k - \epsilon)n/2$. Any vertex whose degree exceeds its out-degree by at most k will have priority 1. When there are no such vertices left, the unoriented edges of M_0 form precisely the $(k+1)$-core of M_0 (and all other vertices can now be ignored). Call this M. Conditional upon M having n vertices and m edges, then it is straightforward to show that it is distributed precisely as in $\mathcal{M}(n,m)$ conditional upon all vertices having degree at least $k+1$. So we apply the algorithm to this M. Put $\rho = 2m/n$. By Theorem 2.1, or, more properly, the analogue for pseudographs as proved in [5], together with monotonicity of the size of the 2-core as m increases, we may assume that $\rho < 2k - \epsilon$ (for a slightly different ϵ).

Define the random variable $Y_{d,j}(t) = |V_{d,j}|$ after t edges of the $(k+1)$-core M have been oriented by the algorithm. Then define A to be the set of ordered pairs (d,j) with $d < 2k$ and $d - j k B$ the ones for which $d < 2k$ and $d - j > k$. These refer to vertices of load-degree at most $2k - 1$, A for priority 1 vertices and B for the others. Then set

$$X(t) := \sum_{(d,j)\in B} (2k - d) Y_{d,j}(t) + \sum_{(d,j)\in A} (d - 2j) Y_{d,j}(t),$$

$$Z(t) := \sum_{\{(d,j):d>2k\}} (d - 2k) Y_{d,j}.$$

Here, $X(t)$ is basically the maximum number of edges that could instantaneously be oriented towards vertices of load-degree less than $2k$ without any load-degrees exceeding $2k$, and $Z(t)$ is a dual variable, the minimum number of edges needed to be oriented away from vertices of load-degree at least $2k+1$ to bring all their load-degrees down below $2k+1$. Actually if $d > 2k$ and $j > 0$ then $Y_{d,j} = 0$ unless the minimum load-degree already grew to at least $2k$; there is no other way for the algorithm to direct an edge towards a vertex of load-degree at least $2k$. This possibility can be ignored, as the proof shows that the algorithm a.a.s. does not reach such a situation.

It can be checked that $X(0) - Z(0) = n(2k - \rho)$, so the bound on ρ above gives $X(0) > Z(0) + \epsilon n$. The proof has two main parts. Firstly, we show that for any constants ξ, $\epsilon > 0$, there exist constants ϵ_c, $\omega > 0$ such that

a.a.s. if $X(0) > Z(0) + \epsilon n$ then

(2.1) $X(t) > \xi Z(t) + \epsilon_c n$ at some $t < T_f - \omega n$

where $T_f = m$ is the number of steps required to orient all edges.

To prove (2.1), we first define $T_l = T_l(\epsilon_b)$, for any $\epsilon_b > 0$, to be the time t when a certain function of the $Y_{d,j}(t)$ first exceeds $1 - \epsilon_b$. This function signifies the growth rate of priority 1 vertices; if it exceeds 1 we would expect an explosion in their numbers. We use a branching process argument to show that before time T_l the total change in $X - Z$ is very small. Part of the reason is that orienting an edge towards a (low load-degree) vertex usually leads to no change in $X - Z$. Since both X and Z are decreasing, this implies that the ratio X/Z is, essentially, increasing, and we may deduce that, assuming $X(0) > Z(0) + \epsilon n$, a.a.s. either T_l is reached or X/Z becomes arbitrarily large before the end of the process: for any constants ξ, ϵ, $\epsilon_b > 0$, there exist constants ϵ_c, $\omega > 0$ such that

> a.a.s. if $X(0) > Z(0) + \epsilon n$ then either $T_l(\epsilon_b) < T_f - \omega n$ or $X(t) > \xi Z(t) + \epsilon_c n$ for some $t < T_f - \omega n$.

We define another stopping time for arbitrary $\epsilon_a > 0$: $T_g(\epsilon_a) := \min\{t : g(X, Z, H) > \epsilon_a\}$ where $g(x, z, h) = \frac{x}{z}\left(1 - \frac{2z}{h}\right) - 1$ and H is the total load-degree of all vertices of load-degree at least $2k + 1$. (Note that edges directed away from vertices do not contribute to their load-degree.) We may use the differential equation method of [23] together with equations describing the behaviour of the variables $Y_{d,j}$ that, provided T_g occurs early enough, the ratio X/Z becomes arbitrarily large before either X or Z gets close to zero: for any constants ξ, ω_a, $\epsilon_a > 0$, there exist constants ϵ_c, $\omega_b > 0$ such that

> a.a.s. if $T_g(\epsilon_a) < T_f - \omega_a n$ then
> $X(t) > \xi Z(t) + \epsilon_c n$ for some $t < T_f - \omega_b n$.

Finally, we may show, assuming $X(0) > Z(0) + \epsilon n$ for some $\epsilon > 0$, that

> a.a.s. if there exist sufficiently small constants ϵ_b, $\omega > 0$ such that $T_l(\epsilon_b) < T_f - \omega n$ then there exists $\epsilon_a > 0$ such that $T_g(\epsilon_a) < T_l(\epsilon_b)$.

To prove this requires a long, detailed analysis bounding the solutions of the differential equations associated with the process (which we do not solve explicitly).

These conclusions together imply (2.1).

For the second part of the proof, we define the stopping time T_h to be the least t for which $X(t) = 0$. Most of this part of the proof is to show the following.

Lemma 2.1. *For any $\delta > 0$, there is a constant $\xi = \xi(\delta)$ such that, for any constants ϵ_c, $\omega > 0$, a.a.s. if there exists $T < T_f - \omega n$ such that $X(T) > \xi Z(T) + \epsilon_c n$, then $\mathbf{P}(Z(T_h) = 0) > 1 - \delta$.*

Since this is true for arbitrarily small δ, we may replace the conclusion by $Z(T_h) = 0$ a.a.s. In view of (2.1) and the observations above it, this conclusion has been established unconditionally. A short argument shows that the algorithm will successfully k-orient M if and only if $Z(T_h) = 0$. So the proof of Theorem 2.3 is then complete. ∎

We finish this section with some comments on the proof of Lemma 2.1. The intuition behind it is summarised as follows. A *heavy* vertex is one of load-degree at least $2k + 1$. Note that $Z(T_h)$ will become zero if enough of the H unoriented edges incident with heavy vertices are directed away from them by time T_h, and $X(t)$ is the number of places where edges can be allocated without the maximum in-degree exceeding k. Qualitatively speaking, if $X(t)/Z(t) \to \infty$ it seems that it should be possible to do this, unless a dense subgraph containing heavy vertices manages to survive somehow until T_h.

The likelihood of such a subgraph seems very small when X is much bigger than Z, and yet, the proof of Lemma 2.1 is quite involved. Firstly, we fix (i.e. condition on) the degree sequence of the subgraph G' induced by unoriented edges at time T, the distribution of degrees of heavy vertices being truncated Poisson as shown in [5]. We assign each vertex a number of "points" equal to its degree. It is possible then to regard the remaining unoriented edges at time T as a random set of pairs of the points. Taking the appropriate model, the pairing of points can be shown to occur u.a.r. Thus we can regard the algorithm as acting on the common configuration model (also called pairing model) of random graphs with given degrees. Assuming that the time T in the lemma exists, we then define a colouring algorithm which builds a matching, M_F, on some of these points as the algorithm proceeds. Finally, we can deduce properties of this matching which imply that $\mathbf{P}(Z(T_h) = 0)$ is arbitrarily close to 1 a.a.s.

The definition of M_F involves several steps and rules for colouring points red and blue as well as some other colours. These colours change dynamically and enable us to identify "sources" (points in heavy vertices) and "sinks" (some points in vertices that, at some time in the process, have degree less than $2k$), and M_F is a matching of these sources and sinks. The generation of M_F involves identifying directed paths of allocated edges, from the sources to the sinks. These paths are not fully completed until time T_h. Each of the $H(t)$ points in heavy vertices at time t is a source, and these are coloured red. There is the potential for $X(t)$ sinks, and $X(t)$ points get coloured blue dynamically. M_F is defined so as to contain all $H(t) + X(t)$ of the red and blue points.

Let us say that a point is *free* at a given time if the edge it corresponds to is not yet oriented. The motivation for defining M_F is that, firstly, any point matched to a blue point in M_F is not free at time T_h. We define a certain graph, G_R, from the red-red pairs in M_F at time T_h, and show that any free red point at time T_h is in the 2-core of G_R. Secondly, on showing that M_F has the distribution of a uniformly random matching, we deduce that G_R can be generated by a simple process, and by analysing a related branching process we bound the probability that G_R contains a cycle. This turns out to vanish a.a.s., showing that no red points (corresponding to points in heavy vertices) remain at this time.

3 Efficient Implementation

Without loss of generality, we can assume the our bins (vertices) are numbered from 1 to n. The most natural inputs for allocation algorithms is a list of m edges. Using bucket sort these can be transformed into an adjacency list representation of the allocation graph G. This graph must support edge deletion in constant time or marking deleted edges.

The Selfless algorithm maintains a priority queue of vertices ordered by their load-degree. Since keys are integers in the range 0 to $\max_{v \in V} \text{degree}(v) \leq m$ we can use a bucket priority queue [9] — an array b containing one entry for each priority value, i.e., $b[i]$ stores the set of elements with priority i. These sets have to support the following operations in constant (amortized) time: Delete a random node and delete or insert a specified node (in order to move it to another bucket). This is possible by representing a bucket as a growable and shrinkable array (e.g., C++ `vector`) that stores vertex numbers in an unordered fashion. In addition, each node has to identify the position in the bucket where it is currently located.[1]

By keeping track of the smallest nonempty bucket, the deleteMin operation can find it in constant time. When this bucket is empty, the array has to be scanned for the next nonempty one. Although this time may be nonconstant, it is easy to see that the total number of buckets scanned during the execution of the Selfless algorithm is at most m plus the total number of decrementKey operations.[2]

The Selfless algorithm needs to choose random edges incident to a given vertex v. It is not so easy to do this in constant time since edges can also be removed. However, closer inspection shows that it is sufficient to randomly permute the input at the beginning and from then on choose the first remaining edge incident to v in the representation used. This works because this edge will be immediately deleted.

4 Experiments

Experiments have been valuable in testing the quality of several algorithms and of initial hypotheses on the connection between threshold values for core sizes and algorithms. For example, it turned out that the high priority rule for committing vertices with degree $\leq L$ makes a significant difference with respect to solution quality.

We want to single out one additional algorithm — *random round robin (RRR)* — because it is slightly simpler than Selfless and because we have reasons to believe that it may be considerably easier to analyze: RRR also uses the rule for vertices with degree $\leq L$ whenever possible. Apart from that, it traverses the vertices of the $(L+1)$-core in random order and commits a random incident edge for each of these vertices. These traversals are repeated until no vertices remain.

Figure 1 shows how L develops when RRR, Selfless, or an optimal scheduling are run on small instances with 16 or 256 bins. As to be expected, the "jumps" in the values of L are less sharp for small n. But even at $n = 16$ the threshold values are a quite good indications of actual performance. Even near the jumps, the quality of Selfless is indistinguishable from the optimal algorithm

[1] The standard implementation of bucket queues with doubly linked lists does not work because it does not support choosing random elements in constant time.

[2] Note that we are outside the usual operating framework of bucket queues where the minimum element increases monotonically. We remain efficient because priorities are only decremented by a constant amount with each call so that we can charge scanning costs to decrementKey operations.

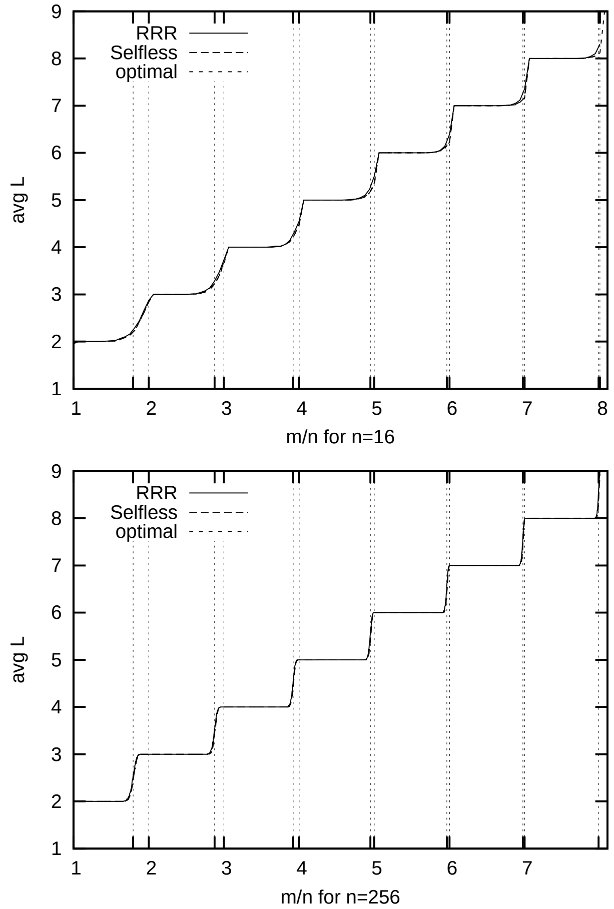

Figure 1: Comparison of the scheduling quality of Selfless and RRR with optimal scheduling for $n = 16$ and $n = 256$. Measurements are averages over 1000 random instances with m balls that can be allocated to two randomly choosen different bins. The noninteger tic marks give the asymptotic threshold values where L makes a jump.

on this scale. RRR has a just visible lag near the smaller threshold values.

We have also implemented and tested the EDR algorithm from [12]. Its solution quality is indistinguishable from Selfless. A complication compared to Selfless is that to actually compute an edge orientation with EDR, one needs to keep track of bypass edges and later replace them by pairs of oriented edges.

5 Discussion

We have presented simple and fast algorithms for load balancing in the duplicate allocation model which have applications for construction of hash tables and disk scheduling. Probabilistic analysis of one of the algorithms established the threshold of non-k-orientability of a random graph.

The disk scheduling application suggests a generalization to the problem to retrieve r out of w pieces of data that are needed to reconstruct a logical data block. The parameters r and w can be set to obtain different tradeoffs between space waste, fault tolerance, and scheduling flexibilty (currently we only have covered the case $r = 1$ and $w = 2$). The cases $r = 1$, and $r = w - 1$ seem of particular interest.

Acknowledgments: We would like to thank Felix Putze for performing experiments with several algorithm variants.

References

[1] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. In *26th ACM Symposium on the Theory of Computing*, pages 593–602, 1994.

[2] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. In *32th Annual ACM Symposium on Theory of Computing*, pages 745–754, 2000.

[3] B. Bollobás and A. Frieze. On matchings and hamiltonian cycles in random graphs. In M. Karoński and A. Ruciński, editors, *Random Graphs 83*, pages 23–46. North-Holland, 1985.

[4] J. Cain. *Random graph processes and optimisation.* PhD thesis, University of Melbourne, 2006.

[5] J. Cain and N. C. Wormald. Encores on cores. submitted for publication, 2005.

[6] L. T. Chen and D. Rotem. Optimal response time retrieval of replicated data (extended abstract). In *13th ACM Symposium on Principles of Database Systems*, volume 13, pages 36–44. ACM Press, 1994.

[7] V. Chvátal. Almost all graphs with 1.44n edges are 3-colorable. *Random structures and algorithms*, 2(1):11–28, 1991.

[8] A. Czumaj and V. Stemann. Randomized allocation processes. *Random Structures & Algorithms*, 18(4):297–331, 2001.

[9] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633, 1969.

[10] M. Dietzfelbinger and F. Meyer auf der Heide. Simple, efficient shared memory simulations. In *5th ACM Symposium on Parallel Algorithms and Architectures*, pages 110–119, Velen, Germany, June 30–July 2, 1993. SIGACT and SIGARCH.

[11] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed

constant size bins. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 166–178. Springer, 2005.

[12] D. Fernholz and V. Ramachandran. The k-orientability thresholds for $G_{n,p}$. In *18th ACM-SIAM Symposium on Discrete Algorithms*, New Orleans, 2007.

[13] S. L. Hakimi. On the degrees of the vertices of a directed graph. *J. Franklin Inst.*, 279:290–308, 1965.

[14] R. M. Karp. Matchings, cores and dense subgraphs in sparse random graphs. Presentation at DIMACS workshop on probabilistic analysis of algorithms for hard problems, Rutgers University, 1999.

[15] R. M. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM simulation on a distributed memory machine. In *24th ACM Symp. on Theory of Computing*, pages 318–326, May 1992.

[16] F. Meyer auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoret. Comput. Sci.*, 162(2):245–281, August 1996.

[17] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: A survey of the techniques and results. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook of Randomized Computing*. Kluwer, 2000.

[18] M. Molloy. Private communication. 1999.

[19] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in a random graph. *J. Combinatorial Theory, Series B*, 67:111–151, 1996.

[20] P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *LNCS*, pages 82–101. Springer, 2004.

[21] P. Sanders, S. Egner, and Jan Korst. Fast concurrent access to parallel disks. *Algorithmica*, 35(1):21–55, 2003. short version in 11th SODA, pages 849–858, 2000.

[22] J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory, I: Two level memories. *Algorithmica*, 12(2/3):110–147, 1994.

[23] N.C. Wormald. The differential equation method for random graph processes and greedy algorithms. In M. Karoński and H.J. Prömel, editors, *Lectures on Approximation and Randomized Algorithms*, pages 73–155. PWN, Warsaw, 1999.