Optimal File Sharing in Distributed Networks*

Moni Naor[†] Ron M. Roth[‡]

Abstract

The following file distribution problem is considered: Given a network of processors represented by an undirected graph $G = (V, E)$, and a file size k, an arbitrary file $\mathbf{w}$ of k bits is to be distributed among all nodes of G. To this end, each node is assigned a memory device such that, by accessing the memory of its own and of its adjacent nodes, the node can reconstruct the contents of $\mathbf{w}$. The objective is to minimize the total size of memory in the network. This paper presents a file distribution scheme which realizes this objective for $k \gg \log \Delta_G$, where Δ_G stands for the maximum degree in G: For this range of k, the total memory size required by the suggested scheme approaches an integer programming lower bound on that size. The scheme is also constructive in the sense that, given G and k, the memory size at each node in G, as well as the mapping of any file $\mathbf{w}$ into the node memory devices, can be computed in time complexity which is polynomial in k and $|V|$. Furthermore, each node can reconstruct the contents of such a file $\mathbf{w}$ in $O(k^2)$ bit operations. Finally, it is shown that the requirement of k being much larger than $\log \Delta_G$ is necessary in order to have total memory size close to the integer programming lower bound.

Key words: De-randomization; Distributed networks; File assignment; Integer Programming; Linear codes; Linear Programming; Probabilistic algorithms; Resource sharing; Set cover.
AMS (MOS) subject classifications: 68P20, 68M10, 68Q20, 68R99, 94B05.

*This work was presented in part at the 32nd IEEE Symposium on Foundations of Computer Science, Puerto Rico, October 1991.

[†]Department of Applied Mathematics and Computer Science, Weizmann Institute, Rehovot 76100, Israel. e-mail: naor@wisdom.weizmann.ac.il. Part of this work was done while the author was with the IBM Research Division, Almaden Research Center.

[‡]Computer Science Department, Technion, Haifa 32000, Israel. e-mail: ronny@cs.technion.ac.il. Part of this work was done while the author was visiting IBM Research Division, Almaden Research Center.

1 Introduction

Consider the following file distribution problem: A network of processors is represented by an undirected graph G. An arbitrary file $\mathbf{w}$ of a prescribed size k (measured, say, in bits) is to be distributed among all nodes of G. We are to assign memory devices to the nodes of G such that, by accessing the memory of its own and of its adjacent nodes, each node can reconstruct the contents of $\mathbf{w}$. Given G and k, the objective is to find a *static* memory allocation to the nodes of G, independent of $\mathbf{w}$, as to minimize the total size of memory in the network. Although we do not restrict the file distribution or reconstruction algorithms to be of any particular form, we aim at simple and efficient ones.

The problem of file allocation in a network, i.e., of storing a file in a network so that every processor has "easy" access to the file, has been considered in many variants (see [4] for a survey). The specific version of reconstruction from adjacent nodes only has received attention in the form of *file segmentation*, where the task is to partition the file so that, for each node u in the network, the union of the file segments stored at nodes adjacent to u is the complete file [4][8][13]. As we shall see, allowing more general reconstruction procedures than simply taking the union of file segments at adjacent nodes can result in a considerable savings of the total amount of memory required: Letting Δ_G denote the maximum degree of any node in G, the memory requirement of the best segmentation scheme can be $\Omega(\log \Delta_G)$ times larger than the optimal requirement in the general scheme; this bound is tight.

We start by deriving linear and integer programming lower bounds on the total size of memory required for any network G and file size k. We then present a simple scheme that attains these bounds for sufficiently large values of k. In this scheme, however, the file size k must be, in some cases, much larger than $\Delta_G \log \Delta_G$ in order to approach the above-mentioned lower bounds. We regard this as a great disadvantage for two reasons: such a scheme may turn out to be efficient only for large files, and, even then, it requires addressing large units of stored data each time a node accesses the file. Thus we devote considerable attention to the problem of finding a scheme that is close to the linear and integer programming bounds with file size that is as small as possible.

Our main result is that the critical file size above which the linear or integer programming bounds can be approached is of the order of $\log \Delta_G$: We present a file distribution scheme for any network G and file size k, of total memory size that is within a multiplicative factor of $1 + \varepsilon(G, k)$ from the linear programming bound, where $\varepsilon(G, k)$ stands for a term which approaches zero as $k/\log \Delta_G$ increases. On the other hand, we present an infinite sequence of network–file-size pairs $\left\{(G_l, k_l)\right\}_{l=0}^{\infty}$ such that $k_l \geq \log \Delta_{G_l}$, and yet any file distribution scheme, when applied to a pair (G_l, k_l), requires memory size which is $1 + \delta(G_l, k_l)$ times larger than the integer (or linear) lower bound, with $\liminf_{l\to\infty} \delta(G_l, k_l) \geq \frac{1}{4}$. This proves that a file size of the order of $\log \Delta_G$ is, indeed, a critical point.

The rest of the paper is organized as follows. In Section 2 we provide the necessary background and definitions. In Section 3 we describe the linear and integer programming lower bounds and prove that the linear programming lower bound can be approached for large file sizes k. In Section 4 we prove our main result, namely, we present a file distribution scheme that approaches the linear programming bound as the ratio $k/\log \Delta_G$ increases. Finally, in Section 5 we exhibit the fact that a file size of $\log \Delta_G$ is a critical point, below which there exist infinite families of networks for which the linear and integer programming lower bounds cannot be attained.

2 Background and definitions

Throughout this paper we assume the underlying network to be presented by an undirected graph $G = (V, E)$, with a set of nodes $V = V_G$ and a set of edges $E = E_G$ such that —

(i) G does not have parallel edges; and —

(ii) each node contains a self loop. This stands for the fact that each node can access its own memory.

An undirected graph satisfying conditions (i) and (ii) will be referred to as a *network graph*.

Two nodes u and v in a network graph $G = (V, E)$ are adjacent if there is an edge in G connecting u and v. The adjacency matrix of a network graph $G = (V, E)$ is the $|V| \times |V|$ matrix $A_G = [a_{u,v}]_{u,v\in V}$, where $a_{u,v} = 1$ when u and v are adjacent, and $a_{u,v} = 0$ otherwise. Note that, by the definition of a network graph, every node $u \in V$ is adjacent to itself and, thus, $a_{u,u} = 1$.

For every $u \in V$, let $\Gamma(u)$ be the set of nodes that are adjacent to u in G. The degree of u is denoted by $\Delta(u) \stackrel{\Delta}{=} |\Gamma(u)|$, and the maximum degree in G is denoted by $\Delta_G \stackrel{\Delta}{=} \max_{u \in V} \Delta(u)$.

Two real vectors $\mathbf{y} = [y_i]_i$ and $\mathbf{z} = [z_i]_i$ are said to satisfy the relation $\mathbf{y} \geq \mathbf{z}$ if $y_i \geq z_i$ for all i. The scalar product $\mathbf{y} \cdot \mathbf{z}$ of these vectors is defined, as usual, by $\sum_i y_i z_i$. A real vector $\mathbf{y}$ is called nonnegative if $\mathbf{y} \geq \mathbf{0}$, where $\mathbf{0}$ denotes the all-zero vector. By the *norm* of a nonnegative vector $\mathbf{y}$ we mean the L_1-norm $\|\mathbf{y}\| \stackrel{\Delta}{=} \mathbf{y} \cdot \mathbf{1}$, where $\mathbf{1}$ denotes the all-one vector.

Given a network graph $G = (V, E)$ and a positive integer k, a *file distribution protocol for* (G, k) is, intuitively, a procedure for allocating memory devices to the nodes of G, and to map an arbitrary file $\mathbf{w}$ of size k into these memory devices, such that each node u can reconstruct $\mathbf{w}$ by reading the memory contents at nodes adjacent to u.

More precisely, let $F_2 \stackrel{\Delta}{=} GF(2)$, let $G = (V, E)$ be a network graph, and let k be a positive integer. For $u \in V$ and a real vector $\mathbf{z} = [z_u]_{u \in V}$ denote by $(A_G \mathbf{z})_u$ the uth entry[1] of $A_G \mathbf{z}$; this entry is equal to $\sum_{v \in \Gamma(u)} z_v$. A file distribution protocol χ for (G, k) is a list $\left(\mathbf{x}\,;\, [\mathcal{E}_u]_{u \in V}\,;\, [\mathcal{D}_u]_{u \in V}\right)$, consisting of —

- *memory allocation*, which is a nonnegative integer vector $\mathbf{x} = [x_u]_{u \in V}$; the entry x_u denotes the size of memory (in bits) assigned to node u;
- *encoding mappings*
$$\mathcal{E}_u \,:\, F_2^k \;\rightarrow\; F_2^{x_u} \quad \text{for every } u \in V\ ;$$
these mappings define the coding rule of any file $\mathbf{w}$ of size k into the memory devices at the nodes: the contents of the memory at node u is given by $\mathcal{E}_u(\mathbf{w})$;
- *decoding (reconstruction) mappings*
$$\mathcal{D}_u \,:\, F_2^{(A_G \mathbf{x})_u} \;\rightarrow\; F_2^k \quad \text{for every } u \in V\ .$$

The memory allocation, encoding mappings, and decoding mappings satisfy the requirement
$$\mathcal{D}_u \left([\,\mathcal{E}_v(\mathbf{w})\,]_{v \in \Gamma(u)} \right) \;\equiv\; \mathbf{w}\ , \quad \mathbf{w} \in F_2^k\ . \tag{1}$$

[1] As we have not defined any order on the set of nodes V, the order of entries in vectors such as $\mathbf{z}$ can be fixed arbitrarily. The same applies to rows and columns of the adjacency matrix A_G, or to subvectors such as $[z_v]_{v \in \Gamma(u)}$.

Equation (1) guarantees that each node u is able to reconstruct the value (contents) of any file $\mathbf{w}$ of size k out of the memory contents $\mathcal{E}_v(\mathbf{w})$ at nodes v adjacent to u.

The *memory size* of a file distribution protocol $\chi = \left(\mathbf{x}\,;\,[\mathcal{E}_u]_{u\in V}\,;\,[\mathcal{D}_u]_{u\in V}\right)$ for (G,k) is defined as the norm $\|\mathbf{x}\|$ and is denoted $|\chi|$. That is, the memory size of a file distribution protocol is the total number of bits assigned to the nodes. The minimum memory size of any file distribution protocol for (G,k) is denoted by $M(G,k)$.

Example 1. The file segmentation method mentioned in Section 1 can be described as a file distribution protocol for (G,k) with memory allocation $\mathbf{x} = [x_u]_{u\in V}$ and associated encoding mappings $\mathcal{E}_u : F_2^k \to F_2^{x_u}$ of the form

$$\mathcal{E}_u : [w_1\, w_2\, \ldots\, w_k] \;\mapsto\; [w_{j(u;1)}\, w_{j(u;2)}\, \ldots\, w_{j(u;x_u)}] \;,$$

where $0 < j(u;1) < j(u;2) < \ldots < j(u;x_u) \le k$. For a node $u \in V$ to be able to reconstruct the original file $\mathbf{w}$, the mappings $\mathcal{E}_v$, $v \in \Gamma(u)$, must be such that every entry w_i of $\mathbf{w}$ appears in at least one $\mathcal{E}_v(\mathbf{w})$. This implies that the set of nodes S_i which w_i is mapped to under the encoding mappings must be a dominating set in G; that is, each node $u \in G$ is adjacent to some node in S_i. On the other hand, given a dominating set S in G, we can construct a file segmentation protocol for (G,k) of memory size $k\cdot|S| \le k\cdot|V|$ (the case $S = V$ corresponds to simply replicating the original file $\mathbf{w}$ into each node in G). •

A *file distribution scheme* is a function $(G,k) \mapsto \chi(G,k)$ which maps every network graph G and positive integer k into a file distribution protocol $\chi(G,k)$ for (G,k).

A file distribution scheme $(G,k) \mapsto \chi(G,k) = \left(\mathbf{x}\,;\,[\mathcal{E}_u]_{u\in V}\,;\,[\mathcal{D}_u]_{u\in V}\right)$ is *constructive* if —

(a) the complexity of computing the memory allocation $\mathbf{x}$ is polynomial in k and $|V|$;

(b) for every $\mathbf{w} \in F_2^k$, the complexity of computing the encoded values $[\,\mathcal{E}_u(\mathbf{w})\,]_{u\in V}$ is polynomial in the memory size $\|\mathbf{x}\|$; and —

(c) for every $u \in V$ and $\mathbf{c} \in F_2^{(A_G\mathbf{x})_u}$, the complexity of reconstructing $\mathbf{w} = \mathcal{D}_u(\mathbf{c})$ out of $\mathbf{c}$ is polynomial in the original file size k.

By computational complexity of a problem we mean the running time of a Turing machine that solves this problem.

Remark 1. In the definition of memory size of file distribution protocols we chose not to count the amount of memory required at each node u to store and run the routines which implement the decoding mappings $\mathcal{D}_u(\cdot)$. The reasoning for neglecting this auxiliary memory is that, in practice, there are a number of files (each, say, of the same size k) that are to be distributed in the network. The file distribution protocol can be implemented independently for each such file, using the *same* program and the same working space to handle all these files. To this end, we might better think of k as the size of the smallest information unit (e.g., a word, or a record) that is addressed at each access to any file. From a complexity point of view, we would prefer k to be as small as possible. The motivation of this paper can be summarized as finding a constructive file distribution scheme $(G, k) \mapsto \chi(G, k)$ which maintains a ratio of memory-size to file-size virtually equal to $\lim_{l\to\infty} M(G, l)/l$ for relatively small file sizes k. •

Remark 2. One might think of a weaker definition for constructiveness by allowing non-polynomial pre-computation of $\mathbf{x}$ (item (a)) and, possibly, of some other data structures which depend on G and k, but not on $\mathbf{w}$ (e.g., calculating suitable representations for $\mathcal{E}_u$ and $\mathcal{D}_u$); such schemes may be justified by the assumption that these pre-computation steps should be done once for a given network graph G and file size k. On the other hand, items (b) and (c) in the constructiveness definition involve the complexity of the more frequent occasions when the file is encoded and — even more so — reconstructed. In this paper, however, we aim at finding file distribution schemes which are constructive in the way we have defined, i.e., in the strong sense: satisfying all three requirements (a)–(c). •

We end this section by introducing a few terms which will be used in describing the mappings $\mathcal{E}_u$ and $\mathcal{D}_u$ of the proposed file distribution schemes. Let Φ be a finite alphabet of q elements. An (n, K) *code* C over Φ is a nonempty subset of Φ^n of size K; the parameter n is called the *length* of C, and the members of C are referred to as *codewords*. The *minimum distance* of an (n, K) code C over Φ is the minimum integer d such that any two distinct codewords in C differ in at least d coordinates.

Let C be an (n, K) code over Φ and let S be a subset of $\langle n \rangle \triangleq \{1, 2, \ldots, n\}$. We say that C is *separable with respect to* S if every two distinct codewords in C differ in at least one coordinate indexed by S. The next lemma follows directly from the definition of minimum distance.

Lemma 1. *The minimum distance of an* (n, K) *code* C *over* Φ *is the minimum integer* d *for which* C *is separable with respect to every set* $S \subseteq \langle n \rangle$ *of size* $n - d + 1$.

Let q be a power of a prime. An (n, K) code C over a field $\Phi = GF(q)$ is *linear* if C is a linear subspace of Φ^n; in this case we have $K = q^k$ where k is the dimension of C. A generator matrix B of a linear (n, q^k) code C over Φ is a $k \times n$ matrix B over Φ whose rows span the codewords of C.

For a $k \times n$ matrix B (such as a generator matrix) and a set $S \subseteq \langle n \rangle$, denote by $(B)_S$ the $k \times |S|$ matrix consisting of all columns of B indexed by S. The following lemma is easily verified.

Lemma 2. *Let* C *be an* (n, q^k) *linear code over a field* Φ, *let* B *be a generator matrix of* C, *and let* S *be a subset of* $\langle n \rangle$. *Then,* C *is separable with respect to* S *if and only if* $(B)_S$ *has rank* k.

3 Lower bounds and statement of main result

In this section we first derive lower bounds on $M(G, k)$, i.e., on the memory size of any file distribution protocol for (G, k). Then, we state our main result (Theorem 2) which establishes the existence of a constructive file distribution scheme $(G, k) \mapsto \chi(G, k)$ that attains these lower bounds whenever $k \gg \log \Delta_G$. As the proof of Theorem 2 is somewhat long, it is deferred to Section 4. Instead, we present in this section a simple file distribution scheme which attains the lower bounds when $k = \Omega(\Delta_G^2 \log \Delta_G)$.

3.1 Lower bounds

Let $\mathbf{x} = [x_u]_{u \in V}$ be a memory allocation of some file distribution protocol for (G, k). Assigning x_u bits to each node $u \in V$, each node must "see" at least k memory bits at its adjacent nodes, or else (1) would not hold. Therefore, for every $u \in V$ we must have $\sum_{v \in \Gamma(u)} x_v \geq k$ or, in vector notation,

$$A_G \mathbf{x} \;\geq\; k \cdot \mathbf{1} \; .$$

Let $J(G, k)$ denote the minimum value attained by the following integer programming problem:

$$\mathrm{IP}(G,k): \qquad \begin{array}{c} J(G,k) = \min \|\mathbf{y}\|, \\ \text{ranging over all integer } \mathbf{y} \text{ such that} \\ A_G \mathbf{y} \;\geq\; k \cdot \mathbf{1} \quad \text{and} \quad \mathbf{y} \geq \mathbf{0} \; . \end{array}$$

Also, let ρ_G denote the minimum value attained by the following (rational) linear programming problem:

$$\mathrm{LP}(G): \qquad \begin{array}{c} \rho_G \;=\; \min \; \|\mathbf{z}\| \; , \\ \text{ranging over all rational } \mathbf{z} \text{ such that} \\ A_G \mathbf{z} \;\geq\; \mathbf{1} \quad \text{and} \quad \mathbf{z} \;\geq\; \mathbf{0} \; . \end{array} \tag{2}$$

The next theorem follows from the previous definitions, Example 1, and the fact that $J(G, 1)$ is the size of a (smallest) dominating set in G.

Theorem 1. *For every network graph G and positive integer k,*

$$\rho_G \cdot k \;\leq\; J(G,k) \;\leq\; M(G,k) \;\leq\; k \cdot J(G,1) \;\leq\; k \cdot |V| \; .$$

We call $J(G, k)$ the *integer programming bound*, whereas $\rho_G \cdot k$ is referred to as the *linear programming bound.*

For $k = 1$, Theorem 1 becomes $M(G, 1) = J(G, 1)$. The problem of deciding whether a network graph G has a dominating set of size $\leq s$ is well-known to be NP-complete [6]. The next corollary immediately follows.

Corollary 1. *Given an instance of a network graph G and positive integers k and s, the problem of deciding whether there exists a file distribution protocol for (G, k) of memory size $\leq s$ (i.e., whether $M(G, k) \leq s$) is NP-hard.*

Note that we do not know whether the decision problem of Corollary 1 is in NP (and therefore, whether it is NP-complete) since it is unclear how to verify (1) in polynomial-time, even when the encoding and decoding mappings are computable in polynomial-time.

Remark 3. A result of Lovász [11] states that $J(G,1) \leq \rho_G \log_2 \Delta_G$; on the other hand, one can construct an infinite family of network graphs $\{G_l\}_l$ (such as the ones presented in Section 5) for which $J(G_l,1) \geq \frac{1}{4}\,\rho_{G_l} \log_2 \Delta_{G_l}$ (see also [7]). In terms of file segmentation schemes (Example 1) this means that there always exists a file distribution protocol for (G,k) based on segmentation whose memory size, $k \cdot J(G,1)$, is within a multiplicative factor of $\log_2 \Delta_G$ from the linear programming bound $\rho_G \cdot k$. Yet, on the other hand, there are families of network graphs for which such a multiplicative gap is definitive (up to a constant 4), even when k tends to infinity. •

3.2 Statement of main result

Corollary 1 suggests that it is unlikely that there exists an efficient algorithm for generating a file distribution scheme $(G,k) \mapsto \chi(G,k)$ with $|\chi(G,k)| = M(G,k)$. This directs our objective to finding a constructive file distribution scheme $(G,k) \mapsto \chi(G,k)$ such that $|\chi(G,k)|/(\rho_G \cdot k)$ is close to 1 for values of k as small as possible.

More specifically, we prove the following theorem.

Theorem 2. *There exists a constructive file distribution scheme $(G,k) \mapsto \chi(G,k)$ such that*

$$\frac{|\chi(G,k)|}{\rho_G \cdot k} = 1 + O\left(\max\left\{\frac{\log \Delta_G}{k}\,;\,\sqrt{\frac{\log \Delta_G}{k}}\right\}\right). \tag{3}$$

(The maximum in the right-hand side of (3) is determined according to whether k is smaller, or larger, than $\log \Delta_G$. Also, by Theorem 1, the ratios $|\chi(G,k)|/M(G,k)$, $M(G,k)/J(G,k)$, and $J(G,k)/(\rho_G \cdot k)$ all approach 1 when $k \gg \log \Delta_G$.)

In Section 4 we prove Theorem 2 by presenting an algorithm for generating a constructive file distribution scheme $(G,k) \mapsto \chi(G,k)$ which satisfies (3); in particular, the computational complexity of the encoding mappings in the resulting scheme (item (b) in the constructiveness requirements) is $O(k \cdot |\chi(G,k)|)$, whereas applying the decoding mapping at each node (item (c)) requires $O(k^2)$ bits operations. Returning to our discussion in Remark 1, the complexity of these mappings suggests that the file size k should be as small as possible, still greater than $\log \Delta_G$. This means that files distributed in the network should be segmented into records of size $k = a \cdot \log \Delta_G$ for some (large) constant a, each record being encoded and decoded independently. Information can be retrieved from the file by reading whole records of size $a \cdot \log \Delta_G$ bits each, requiring $O(a^2 \log^2 \Delta_G)$ bit operations, whereby the ratio between the memory size required in the network and the file size k is at most $1 + O(1/\sqrt{a})$ times that ratio for $k \to \infty$.

Our file distribution algorithm is divided into two major steps:

Step 1. Finding a memory allocation $\mathbf{x} = [x_u]_{u \in V}$ for (G,k) by finding an approximate solution to an integer programming problem; the resulting memory size $|\chi(G,k)| = \|\mathbf{x}\|$ will satisfy (3).

Step 2. Constructing a set of $k \times x_u$ matrices B_u, $u \in V$, over F_2; these matrices define the encoding mappings $\mathcal{E}_u : F_2^k \to F_2^{x_u}$ by $\mathcal{E}_u : \mathbf{w} \mapsto \mathbf{w}B_u$, $u \in V$. The choice of the matrices B_u, in turn, is such that each $k \times (A_G\mathbf{x})_u$ matrix $[B_v]_{v \in \Gamma(u)}$ is of rank k, thus yielding decoding mappings $\mathcal{D}_u : F_2^{(A_G\mathbf{x})_u} \to F_2^k$ which satisfy (1).

3.3 File distribution scheme for large files

In this section we present a fairly simple constructive file distribution scheme $(G,k) \mapsto \chi(G,k)$ for which

$$\frac{|\chi(G,k)|}{\rho_G \cdot k} = 1 + O\left(\frac{\Delta_G \cdot \log(\Delta_G \cdot k)}{k}\right) .$$

Note that this proves Theorem 2 whenever $k = \Omega(\Delta_G^2 \log \Delta_G)$.

Given a network graph $G = (V,E)$ and a positive integer k, we first compute a memory allocation $\mathbf{x} = [x_u]_{u \in V}$ for (G,k) (Step 1 above). Let $\mathbf{z} = [z_u]_{u \in V}$ be an optimal solution to the linear programming problem LP(G) in (2). Such a vector $\mathbf{z}$ can be found in time complexity which is polynomial in $|V|$ (e.g., by using Karmarkar's algorithm [9]). Set $h \triangleq \lceil \log_2(\Delta_G \cdot k) \rceil$

and $l \triangleq \lceil k/h \rceil$, and define the integer vector $\mathbf{y} = [y_u]_{u \in V}$ by

$$y_u \;\triangleq\; \min\Big\{ l\,;\ \lfloor (l + \Delta_G) \cdot z_u \rfloor \Big\}\,, \quad u \in V\,.$$

Clearly, $\|\mathbf{y}\| \;\le\; \rho_G \cdot (l + \Delta_G)$; furthermore, since $A_G\mathbf{z} \ge \mathbf{1}$, we also have

$$(A_G\mathbf{y})_u \;\ge\; \min\Big\{ l\,;\ (l + \Delta_G)(A_G\mathbf{z})_u - \Delta_G \Big\} \;\ge\; l\,, \quad u \in V\,,$$

i.e., $A_G\mathbf{y} \ge l \cdot \mathbf{1}$. The memory allocation for (G, k) is defined by $\mathbf{x} \triangleq h \cdot \mathbf{y}$, and it is easy to verify that $\|\mathbf{x}\|/(\rho_G \cdot k) = 1 + O\Big((\Delta_G/k)\log(\Delta_G \cdot k)\Big)$.

We now turn to defining the encoding and decoding mappings (Step 2 above). To this end, we first assign $\Delta_G \cdot l$ colors to the nodes of G, with each node u assigned a set C_u of y_u colors, such that $\left|\bigcup_{v \in \Gamma(u)} C_v\right| \ge l$, $u \in V$. In other words, we multi-color the nodes of G in such a way that each node "sees" at least l colors at its adjacent nodes.

Such a coloring can be obtained in the following greedy manner: Start with $C_u \leftarrow \emptyset$ for every $u \in V$. Call a node u *saturated* if $\left|\bigcup_{v \in \Gamma(u)} C_v\right| \ge l$ (hence, at the beginning all nodes are unsaturated, whereas at the end all should become saturated). Scan each node $u \in V$ once, and, at each visited node u, re-define the set C_u to have y_u distinct colors not contained in sets C_v already assigned to nodes $v \in \Gamma(u')$ for all unsaturated nodes $u' \in \Gamma(u)$.

To verify that such a procedure yields, indeed, an all-saturated network, we first show that at each step there are enough colors to assign to the current node. Let $\sigma(u)$ denote the number of unsaturated nodes $u' \in \Gamma(u) - \{u\}$ when C_u is being re-defined. Recalling that $y_v \le l$ for every $v \in V$, it is easy to verify that the number of disqualified colors for C_u is at most $\sigma(u) \cdot (l-1) + (\Delta(u) - \sigma(u) - 1) \cdot l \le \Delta_G l - l \le \Delta_G l - y_u$. This leaves at least y_u qualified colors to assign to node u. We now claim that each node becomes saturated at some point. For if node u remained unsaturated all along, then the sets C_v, $v \in \Gamma(u)$, had to be disjoint; but in that case we would have

$$\left|\bigcup\nolimits_{v \in \Gamma(u)} C_v\right| = \sum_{v \in \Gamma(u)} |C_v| = \sum_{v \in \Gamma(u)} y_u = (A_G\mathbf{y})_u \ge l,$$

contradicting the fact that u was unsaturated.

Let $\alpha_1, \alpha_2, \ldots, \alpha_{\Delta_G \cdot l}$ be distinct elements in $\Phi \triangleq GF(2^h)$, each α_j corresponding to some color j (note that $|\Phi| \ge \Delta_G \cdot k \ge \Delta_G \cdot l$). Given a file $\mathbf{w}$ of k bits, we group the entries of $\mathbf{w}$ into h-tuples to form the coefficients of a polynomial $w(t)$ of degree $< \lceil k/h \rceil = l$ over Φ. We now compute the values $w_j = w(\alpha_j)$, $1 \le j \le \Delta_G \cdot l$, and store at each node $u \in V$ the values w_j, $j \in C_u$, requiring memory allocation of $x_u = h \cdot y_u$ bits. Since each u has access to images w_j of $w(t)$ evaluated at l distinct elements α_j, each node can interpolate the polynomial $w(t)$ and, hence, reconstruct the file $\mathbf{w}$.

The above encoding procedure can be described also in terms of linear codes (refer to the end of Section 2). Such a characterization will turn out to be useful in Sections 4 and 5. Let B_{RS} be an $l \times (\Delta_G l)$ matrix over $\Phi = GF(2^h)$ defined by $(B_{\mathrm{RS}})_{i,j} = \alpha_j^{i-1}$, $1 \le i \le l$, $1 \le j \le \Delta_G l$. For every node $u \in V$, let C_u be the set of colors assigned to u and let $B_u \triangleq (B_{\mathrm{RS}})_{C_u}$; that is, regarding C_u as a subset of $\{1, 2, \ldots, \Delta_G l\}$, B_u consists of all columns of B_{RS} indexed by C_u. The mappings $\mathcal{E}_u : F_2^k \to F_2^{x_u}$, or, rather, $\mathcal{E}_u : \Phi^l \to \Phi^{y_u}$, are defined by $\mathcal{E}_u : \mathbf{w} \mapsto \mathbf{w}B_u$, $u \in V$, $\mathbf{w} \in \Phi^l$. The matrix B_{RS} is known as a generator matrix of a $(\Delta_G l, 2^{hl})$ generalized Reed-Solomon code over Φ [12, Chs. 10–11]. Note that since every l columns in B_{RS} are linearly independent, every $l \times (A_G \mathbf{y})_u$ matrix $[B_v]_{v \in \Gamma(u)}$ has rank l, allowing each node u to reconstruct $\mathbf{w}$ out of $[\mathbf{w}B_v]_{v \in \Gamma(u)}$.

We remark that Reed-Solomon codes have been extensively applied to some other reconstruction problems in networks, such as Shamir's secret sharing [18] (see also [10][14]).

The file distribution scheme described in this section is not satisfactory when the file size k is, say, $O(\Delta_G)$, in which case the ratio $\chi(G,k)/(\rho_G \cdot k)$ might be bounded away from 1. This will be rectified in our next construction which is presented in Section 4.

4 Proof of main result

In this section we present a file distribution scheme which attains the memory size stated in Theorem 2. In Section 4.1 we present a randomized algorithm for finding a memory allocation by scaling and perturbing a solution to the linear programming problem LP(G) defined in (2). Having found a memory allocation $\mathbf{x}$, we describe in Section 4.2 a second randomized algorithm for obtaining the encoding and decoding mappings. Both algorithms are then de-randomized in Section 4.3 to obtain a deterministic procedure for computing the file distribution scheme claimed in Theorem 2. In Section 4.4 we present an alternative proof of the theorem using the Lovász Local Lemma. In Section 4.5 we consider a variant of the cost measure used in the rest of the paper: instead of looking for a near optimal solution with respect to the total memory requirement of the system, we consider approximating the best solution such that the maximum amount of memory required in any node is close to the minimum feasible. This is done using the techniques of Section 5.

4.1 Step 1. Solving for a memory allocation

The goal of this section is to prove the following (hereafter e stands for the base of natural logarithms).

Theorem 3. *Given a network graph G and an integer m, let $\mathbf{z} = [z_u]_{u \in V}$ be a nonnegative real vector satisfying $A_G \mathbf{z} \geq \mathbf{1}$. Then there is a nonnegative integer vector $\mathbf{x}$ satisfying $A_G \mathbf{x} \geq m \cdot \mathbf{1}$ such that*

$$\frac{\|\mathbf{x}\|}{\|m \cdot \mathbf{z}\|} \leq 1 + c \cdot \max \left\{ \frac{\log_e \Delta_G}{m} \; ; \; \sqrt{\frac{\log_e \Delta_G}{m}} \right\} \tag{4}$$

for some absolute constant c.

In fact, we provide also an efficient algorithm to compute the nonnegative integer vector $\mathbf{x} = [x_u]_{u \in V}$ guaranteed by the theorem. The vector $\mathbf{x}$ will serve as the memory allocation of the computed file distribution protocol for an instance (G, k), where we will need to take m slightly larger than k in order to construct the encoding and decoding mappings in Section 4.2.

Theorem 3 is proved via a 'randomized rounding' argument (see [15][17]): We first solve the corresponding linear programming problem LP(G) in (2) (say, by Karmarkar's algorithm [9]), and use the rational solution to define a probability measure on integer vectors that are candidates for $\mathbf{x}$. We then show that this probability space contains an integer vector $\mathbf{x}$ which satisfies the conditions of Theorem 3. Furthermore, such a vector can be found by a polynomial-time (randomized) algorithm. Note that if we are interested in a weaker result, where $\log |V|$ replaces $\log \Delta_G$ in Theorem 2 (or in Theorem 3), then a slight modification of Raghavan's lattice approximation method can be applied [15]. However, to prove Theorem 3 as is, we need a so-called 'local' technique. One possibility is to use the 'method of alteration' (see [19]) where a random integer vector selected from the above probability space is perturbed in a few coordinates so as to satisfy the conditions of the theorem. Another option is to use the Lovász Local Lemma. Both methods can be used to prove Theorem 3, and both can be made constructive and deterministic: the method of alteration by applying the method of conditional probabilities (see Spencer [19, p. 31] and Raghavan [15]), and the Local Lemma by using Beck's method [2]. We show here the method of alteration, and present a second existence proof using the Local Lemma in Section 4.4.

Given a nonnegative real vector $\mathbf{z} = [z_u]_{u \in V}$ and a real number $\ell > 0$, define the vectors $\mathbf{s} = [s_u]_{u \in V}$ and $\mathbf{p} = [p_u]_{u \in V}$ by

$$s_u \triangleq \lfloor \ell \cdot z_u \rfloor \quad \text{and} \quad p_u \triangleq \ell \cdot z_u - s_u \ ; \quad u \in V \ ; \tag{5}$$

note that $0 \le p_u < 1$ for every $u \in V$. Let $\mathbf{Y} = [Y_u]_{u \in V}$ be a random vector of independent random variables Y_u over $\{0, 1\}$ such that

$$\text{Prob}\left\{Y_u = 1\right\} = p_u \ , \quad u \in V \ , \tag{6}$$

and let $\mathbf{X} = [X_u]_{u \in V}$ be a random vector defined by

$$\mathbf{X} \triangleq \mathbf{s} + \mathbf{Y} \ . \tag{7}$$

Fix $\mathbf{a}$ to be a real vector in the unit hyper-cube $[0, 1]^{|V|}$ such that $\mathbf{a} \cdot \mathbf{z} \ge 1$. Since the expectation vector $E\big(\mathbf{Y}\big)$ is equal to $\mathbf{p}$, we have

$$E\big(\mathbf{a} \cdot \mathbf{X}\big) = \mathbf{a} \cdot \mathbf{s} + \mathbf{a} \cdot \mathbf{p} = \ell \cdot \mathbf{a} \cdot \mathbf{z} \ge \ell \ .$$

In particular, if $\mathbf{z}$ is a rational vector satisfying $A_G \mathbf{z} \ge \mathbf{1}$, then

$$E\big(A_G \mathbf{X}\big) \ge \ell \cdot A_G \mathbf{z} \ge \mathbf{1} \ge \ell \cdot \mathbf{1} \ .$$

Showing the existence of an instance of $\mathbf{X}$ which can serve as the desired memory allocation $\mathbf{x}$ makes use of the following two propositions. The proofs of these propositions are given in the Appendix, as similar statements can be found also in [15].

Throughout this section, $L\{\beta, \eta\}$ stands for $\max\left\{\log_e \beta \ ; \ \sqrt{\eta \cdot \log_e \beta}\right\}$.

Proposition 1. *Given a nonnegative real vector* $\mathbf{z}$ *and an integer* ℓ, *let* $\mathbf{X} = [X_u]_{u \in V}$ *be defined by (5)–(7), let* $\mathbf{a}$ *be a real vector in* $[0,1]^{|V|}$ *such that* $\mathbf{a} \cdot \mathbf{z} \geq 1$, *and let* m *be a positive integer. There exists a constant* c_1 *such that, for every* $\beta \geq 1$,

$$\mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{X} \ < \ m\right\} \ \leq \ \frac{1}{\beta}$$

whenever $\ell \ \geq \ m \ + \ c_1 \cdot L\{\beta, m\}$.

Proposition 2. *Given a nonnegative real vector* $\mathbf{z}$ *and an integer* ℓ, *let* $\mathbf{X} = [X_u]_{u \in V}$ *be defined by (5)–(7) and let* $\mathbf{a}$ *be a real vector in* $[0,1]^{|V|}$. *There exists a constant* c_2 *such that, for every* $\beta \geq 1$,

$$\mathrm{Prob}\ \left\{\mathbf{a} \cdot \mathbf{X} \ > \ E\Big(\mathbf{a} \cdot \mathbf{X}\Big) \ + \ c_2 \cdot L\left\{\beta, E\Big(\mathbf{a} \cdot \mathbf{X}\Big)\right\}\right\} \ \leq \ \frac{1}{\beta}\,.$$

Consider the following algorithm for computing a nonnegative integer vector $\mathbf{x}$ for an instance (G, m):

Algorithm 1.

1. *Set* $\beta = \beta_G \triangleq 2\Delta_G^2$ and $\ell = m + c_1 \cdot L\{\beta_G, m\}$.
2. *Solve the linear programming problem LP*(G) *(defined by (2)) for* $\mathbf{z}$.
3. *Generate an instance of the random vector* $\mathbf{X} = [s_u]_u + [Y_u]_u$ *as in (5)–(7).*
4. *The integer vector* $\mathbf{x} = [x_u]_{u \in V}$ *is given by*

$$x_u \triangleq \begin{cases} s_u + 1 & \textit{if there exists } v \in \Gamma(u) \textit{ with } (A_G\mathbf{X})_v < m \\ s_u + Y_u & \textit{otherwise} \end{cases}.$$

Theorem 3 is a consequence of the following lemma.

Lemma 3. *The vector* $\|\mathbf{x}\|$ *obtained by Algorithm 1 satisfies Inequality (4) with probability* $\geq \frac{1}{2} - \frac{1}{\beta_G}$.

Proof. Call a node v *deficient* if $(A_G\mathbf{X})_v < m$ for the generated vector $\mathbf{X}$. First note that x_u is either X_u or $X_u + 1$ and that

$$A_G\mathbf{x} \;\geq\; m \cdot \mathbf{1}\,; \tag{8}$$

in fact, for deficient nodes v we have $(A_G\mathbf{x})_v \geq \ell \cdot (A_G\mathbf{z})_v \geq \ell \geq m$.

Now, by Proposition 1, for every node $v \in V$,

$$\text{Prob}\Big\{\text{ node } v \text{ is deficient }\Big\} \;=\; \text{Prob}\Big\{(A_G\mathbf{X})_v \;<\; m\Big\} \;\leq\; \frac{1}{\beta_G}\,.$$

Hence, for each node $u \in V$,

$$\text{Prob}\Big\{\; x_u \;=\; X_u + 1 \;\Big\} \;\leq\; \Delta_G \cdot \text{Prob}\Big\{\text{ node } v \text{ is deficient }\Big\} \;\leq\; \frac{\Delta_G}{\beta_G} \;=\; \frac{1}{2\Delta_G}\,.$$

Therefore, the expected number of nodes u for which $x_u = X_u + 1$ is at most $\frac{|V|}{2\Delta_G}$ and, with probability at least $\frac{1}{2}$, there are no more than $\frac{|V|}{\Delta_G}$ such nodes u. Observing that

$$\|\mathbf{z}\| \;\geq\; \frac{1}{\Delta_G}\sum_{u\in V}(A_G\mathbf{z})_u \;\geq\; \frac{1}{\Delta_G}\sum_{u\in V} 1 \;=\; \frac{|V|}{\Delta_G}\,,$$

we thus obtain, with probability $\geq \frac{1}{2}$,

$$\|\mathbf{x}\| \;\leq\; \|\mathbf{X}\| \;+\; \frac{|V|}{\Delta_G} \;\leq\; \|\mathbf{X}\| \;+\; \|\mathbf{z}\|\,. \tag{9}$$

Recalling that $E\Big(\|\mathbf{X}\|\Big) = \ell \cdot \|\mathbf{z}\|$, we apply Proposition 2 with $\mathbf{a} = \mathbf{1}$ to obtain

$$\text{Prob}\Big\{\|\mathbf{X}\| \;>\; \ell \cdot \|\mathbf{z}\| \;+\; c_2 \cdot L\{\beta_G, \ell \cdot \|\mathbf{z}\|\}\Big\} \;\leq\; \frac{1}{\beta_G}\,. \tag{10}$$

Hence, by (8), (9), and (10) we conclude that, with probability $\geq \frac{1}{2} - \frac{1}{\beta_G}$, the integer vector $\mathbf{x}$ satisfies both

$$A_G\mathbf{x} \;\geq\; m \cdot \mathbf{1} \tag{11}$$

and

$$\begin{aligned} \|\mathbf{x}\| \;&\leq\; (\ell+1) \cdot \|\mathbf{z}\| + c_2 \cdot L\Big\{\beta_G, \ell \cdot \|\mathbf{z}\|\Big\} \\ &\leq\; (\ell+1) \cdot \|\mathbf{z}\| + c_2 \cdot \|\mathbf{z}\| \cdot L\Big\{\beta_G, \ell\Big\}\,. \end{aligned}$$

The last inequality implies

$$\frac{\|\mathbf{x}\|}{\|\ell \cdot \mathbf{z}\|} \;\leq\; 1 \;+\; \frac{1}{\ell} \;+\; c_2 \left(\frac{\log_e \beta_G}{\ell} \;+\; \sqrt{\frac{\log_e \beta_G}{\ell}} \right) , \tag{12}$$

and the lemma now follows by substituting

$$\ell \;=\; m \cdot \left(1 \;+\; c_1 \cdot \max \left\{ \frac{\log_e \beta_G}{m} \; ; \; \sqrt{\frac{\log_e \beta_G}{m}} \right\} \right)$$

and $\beta_G = 2\Delta_G^2$ in (12). $\square$

Note that for $m = k + O(\log \Delta_G)$ we also have

$$\frac{\|\mathbf{x}\|}{\|k \cdot \mathbf{z}\|} \;=\; 1 \;+\; O\left(\max \left\{ \frac{\log \Delta_G}{k} \; ; \; \sqrt{\frac{\log \Delta_G}{k}} \right\} \right) \tag{13}$$

(compare with the right-hand side of (3)). The vector $\mathbf{x}$, computed for $m = k + O(\log \Delta_G)$, will serve, with a slight modification, as the memory allocation of $\chi(G, k)$. In Section 4.3 we shall apply the method of conditional probabilities to make Algorithm 1 deterministic.

4.2 Step 2. Defining the encoding mappings

Having found a memory allocation $\mathbf{x}$, we now provide a randomized algorithm for constructing the encoding and decoding mappings. The construction makes use of the following lemma.

Lemma 4. [12, p. 444]. *Let* $\mathbf{S}$ *denote a random matrix, uniformly distributed over all* $k \times m$ *matrices over* F_2. *Then,*

$$\mathrm{Prob}\left\{ \mathrm{rank}\,(\mathbf{S}) = k \right\} \;=\; \prod_{i=0}^{k-1} (1 - 2^{i-m}) \;>\; 1 - 2^{k-m} \; .$$

Given an instance (G, k), let $\mathbf{x} = [x_u]_{u \in V}$ be the nonnegative integer vector obtained by Algorithm 1 for $m = k + 3\lceil \log_2 \Delta_G \rceil + 1$. The following algorithm computes for each node u a matrix B_u to be used for the encoding mappings.

Algorithm 2.

1. *For each $u \in V$, assign at random a matrix $\mathbf{Q}_u$ uniformly distributed over all $k \times x_u$ matrices over F_2.*

2. *For each $u \in V$, let $\mathbf{S}_u \triangleq [\mathbf{Q}_v]_{v \in \Gamma(u)}$, and define the encoding matrix B_u by*

$$B_u \triangleq \begin{cases} \mathbf{Q}_u & \textit{if}\ \mathrm{rank}\,(\mathbf{S}_u) = k \\ I_k & \textit{if}\ \mathrm{rank}\,(\mathbf{S}_u) < k \end{cases}, \tag{14}$$

where I_k stands for the $k \times k$ identity matrix.

Note that each B_u is a $k \times \hat{x}_u$ binary matrix with

$$\hat{x}_u = \begin{cases} x_u & \text{if rank}\,(\mathbf{S}_u) = k \\ k & \text{if rank}\,(\mathbf{S}_u) < k \end{cases}. \tag{15}$$

The vector $\hat{\mathbf{x}} \triangleq [\hat{x}_u]_{u \in V}$ will serve as the (final) memory allocation for $\chi(G, k)$. As we show later on in this section, the excess of $\|\hat{\mathbf{x}}\|$ over $\|\mathbf{x}\|$, if any, is small enough to let Equation (13) hold also with respect to the memory allocation $\hat{\mathbf{x}}$. This will establish the memory size claimed in Theorem 2. The associated encoding mappings $\mathcal{E}_u : F_2^k \to F_2^{\hat{x}_u}$ are given by $\mathcal{E}_u : \mathbf{w} \mapsto \mathbf{w}B_u$, $u \in V$, and the overall process of encoding $\mathbf{w}$ into $[\,\mathcal{E}_u(\mathbf{w})\,]_{u \in V}$ requires $O(k \cdot \|\hat{\mathbf{x}}\|)$ multiplications and additions over F_2.

Recalling the definitions in Section 2, note that for each node u, the $k \times \|\hat{\mathbf{x}}\|$ matrix $B \triangleq [B_v]_{v \in V}$ is separable with respect to the set $\Gamma(u)$; that is, the rank of $(B)_{\Gamma(u)} = [B_v]_{v \in \Gamma(u)}$ is k. Therefore, each node u, knowing the values $[\,\mathcal{E}_v(\mathbf{w})\,]_{v \in \Gamma(u)} = [\mathbf{w}B_v]_{v \in \Gamma(u)}$, is able to reconstruct the file $\mathbf{w}$. To this end, node u has to process only k fixed coordinates of $\mathbf{w}(B)_{\Gamma(u)}$, namely, k coordinates which correspond to k linearly independent columns of $(B)_{\Gamma(u)}$. Let such a set of coordinates be indexed by the set T_u, $u \in V$. Assuming a 'hard-wired' connection between node u and the k entries of $\mathbf{w}(B)_{\Gamma(u)}$ indexed by T_u, the decoding process at u sums up to multiplying the vector $\mathbf{w}(B)_{T_u} \in F_2^k$ by the inverse of $(B)_{T_u}$. Hence, the mappings $\mathcal{D}_u$, $u \in V$, are given by $\mathcal{D}_u(\mathbf{c}) = (\mathbf{c})_{T_u}\left((B)_{T_u}\right)^{-1}$ for every $\mathbf{c} \in F_2^{(A_G\hat{\mathbf{x}})_u}$. The decoding process at each node thus requires $O(k^2)$ multiplications and additions over F_2. Note that in those cases where we set B_u in (14) to be the identity matrix, the decoding process is trivial, since the whole file is written at node u.

We now turn to estimating the memory size $\hat{\mathbf{x}}$. First note that for every node u, the matrix $\mathbf{S}_u$ is uniformly distributed over all $k \times (A_G\mathbf{x})_u$ matrices over F_2. Recalling that, by construction, $(A_G\mathbf{x})_u \geq m = k + 3\lceil \log_2 \Delta_G \rceil + 1$, we have, by Lemma 4,

$$\mathrm{Prob}\left\{\mathrm{rank}\,(\mathbf{S}_u) < k\right\} \; < \; 2^{k-m} \; \leq \; \frac{1}{2\Delta_G^3} \, .$$

Hence, the expected number of nodes for which $\hat{x}_u > x_u$ in (15) is at most $|V|/(2\Delta_G^3)$. Therefore, with probability at least $\frac{1}{2}$, there are no more than $|V|/\Delta_G^3$ nodes u whose memory allocation x_u has been increased to $\hat{x}_u = k$. Since $|V|/\Delta_G^3 \leq \|\mathbf{z}\|/\Delta_G^2$, the total memory-size increase in (15) is bounded from above by $(k/\Delta_G^2)\|\mathbf{z}\|$. Hence, by (13),

$$\frac{\|\hat{\mathbf{x}}\|}{\|k \cdot \mathbf{z}\|} \; \leq \; \frac{\|\mathbf{x}\| + (k/\Delta_G^2)\|\mathbf{z}\|}{\|k \cdot \mathbf{z}\|} \; = \; 1 \; + \; O\left(\max\left\{\frac{\log \Delta_G}{k} \; ; \; \sqrt{\frac{\log \Delta_G}{k}}\right\}\right)$$

whenever $k = O(\Delta_G^2 \log \Delta_G)$. Recall that the construction of Section 3.3 covers Theorem 2 for larger values of k.

In Section 4.3 we apply the method of conditional probabilities (see [19, p. 31] and [15]) in order to make the computation of the matrices B_u deterministic.

Remark 4. It is worthwhile comparing the file distribution scheme described in Sections 4.1 and 4.2 with the scheme of Section 3.3, modified to employ Algorithm 1 on $(G, \lceil k/h \rceil)$, $h = \lceil \log_2(\Delta_G \cdot k) \rceil$, to solve for the memory allocation there. It can be verified that the resulting file distribution scheme is slightly worse than the one obtained here: every term $\log \Delta_G$ in (3) should be changed to $\log(\Delta_G \cdot k) \log \Delta_G$. In particular, this method has critical file size of $\log^2 \Delta_G$. •

4.3 A deterministic algorithm

We now show how to make Algorithms 1 and 2 deterministic using the method of conditional probabilities of Spencer [19, p. 31] and Raghavan [15], adapted to conditional expectation values. The idea of the method of conditional probabilities is to search the probability space defined by the random choices. At each iteration the probability space is bisected by setting one of the random variables. Throughout the search we estimate the probability of success, conditional on the choices we have fixed so far. The value of the next random variable is chosen as the one that maximizes the estimator function.

In de-randomizing Algorithms 1 and 2 we employ as an estimator the expected value of the size of the allocation. At every step the conditional expectation for both possibilities for the value of the next random variable are computed and the setting that is smaller (thus increasing the probability of success) is chosen. Unlike Raghavan [15], we do not employ a "pessimistic estimator," but rather a conditional expectation estimator which is fairly easy to compute.

We start with de-randomizing the computation of the (initial) memory allocation $\mathbf{x}$. Let $\mathbf{z} = [z_u]_u = [s_u + p_u]_u$, $\mathbf{X} = [X_u]_u$, and $\mathbf{x} = [x_u]_u$ be the vectors computed in the course of Algorithm 1. Recall that for every $u \in V$, the entry X_u is a random variable given by $X_u = s_u + Y_u$, with $\text{Prob}\,\{Y_u = 1\} = p_u$. Now,

$$\begin{aligned} E\Big(\|\mathbf{x}\|\Big) &= E\Big(\|\mathbf{X}\|\Big) + \sum_{u \in V} \text{Prob}\Big\{ x_u = X_u + 1 \Big\} \\ &\le E\Big(\|\mathbf{X}\|\Big) + \Delta_G \cdot \sum_{u \in V} \text{Prob}\Big\{ \text{node } u \text{ is deficient} \Big\} \\ &\triangleq \hat{E} \; . \end{aligned}$$

We refer to $\hat{E}$ as the *expectation estimator* for $\mathbf{x}$, and we have,

$$\begin{aligned} E\Big(\|\mathbf{x}\|\Big) \le \hat{E} &= E\Big(\|\mathbf{X}\|\Big) + \Delta_G \cdot \sum_{u \in V} \text{Prob}\Big\{(A_G \mathbf{X})_u < m\Big\} \\ &= \ell \cdot \|\mathbf{z}\| + \Delta_G \cdot \sum_{u \in V} \text{Prob}\Big\{ \sum_{v \in \Gamma(u)} X_v < m\Big\} \\ &\le \ell \cdot \|\mathbf{z}\| + \frac{|V|}{2\Delta_G} \le (\ell + \tfrac{1}{2}) \cdot \|\mathbf{z}\| \; . \end{aligned}$$

Comparing the last inequality with (12), it would suffice if we found a memory allocation whose size is at most $\hat{E}$. Note that $\hat{E}$ can be computed efficiently by calculating the expressions $\text{Prob}\Big\{\sum_{v \in W_i} X_v < j\Big\}$ for subsets W_i of $\Gamma(u)$ consisting of the first i nodes in $\Gamma(u)$ for $i = 1, 2, \ldots, \Delta(u) = |\Gamma(u)|$, and for $\sum_{u \in W_i} s_u \le j \le m$. Such a computation can be carried out efficiently by dynamic programming.

Let Y_1 denote the first entry of $\mathbf{Y} = \mathbf{X} - \mathbf{s}$ and define the conditional expectation estimators by

$$\hat{E}_b \triangleq E\Big(\|\mathbf{X}\| \;\Big|\; Y_1 = b\Big) + \Delta_G \cdot \sum_{u \in V} \text{Prob}\Big\{ \sum_{v \in \Gamma(u)} X_v < m \;\Big|\; Y_1 = b\Big\} , \quad b = 0, 1 \; .$$

Indeed, we have $E\Big(\|\mathbf{x}\| \;\Big|\; Y_1 = b\Big) \le \hat{E}_b$; furthermore, the two conditional expectation estimators $\hat{E}_0$ and $\hat{E}_1$ have $\hat{E}$ as a convex combination and, therefore, one of them must be bounded from above by $\hat{E}$. We set the entry Y_1 to the bit $y_1 = b$ for which $\hat{E}_b$ is the smallest. Note that, like $\hat{E}$, the conditional expectation estimators can be efficiently computed.

Having determined the first entry in $\mathbf{Y}$, we now re-iterate this process with the second entry, Y_2, now involving the conditional expectation estimators $\hat{E}_{y_1,0}$ and $\hat{E}_{y_1,1}$. Continuing this way with subsequent entries of $\mathbf{Y}$, we end up with a nondecreasing sequence of conditional expectation estimators

$$\begin{aligned}(\ell + \tfrac{1}{2}) \cdot \|\mathbf{z}\| \;&\ge\; \hat{E} \;\ge\; \hat{E}_{y_1} \;\ge\; \hat{E}_{y_1,y_2} \;\ge\; \ldots \;\ge\; \hat{E}_{y_1,y_2,\ldots,y_{|V|}} \\ &\ge\; E\Big(\|\mathbf{x}\| \;\Big|\; Y_1 = y_1, Y_2 = y_2, \ldots, Y_{|V|} = y_{|V|}\Big) ,\end{aligned}$$

thus determining the whole vector $\mathbf{Y}$, and therefore the vectors $\mathbf{X}$ and $\mathbf{x}$, the latter having memory size $\le (\ell + \frac{1}{2}) \cdot \|\mathbf{z}\|$.

We now turn to making the computation of the encoding mappings deterministic. Recall that Algorithm 2 first assigns a random $k \times x_u$ matrix $\mathbf{Q}_u$ to each node u. We may regard this assignment as an $\|\mathbf{x}\|$-step procedure, where at the nth step a random column of F_2^k is added to a node v with less than x_v already-assigned columns. Denote by $\mathbf{Q}_{u;n}$ the (partial) matrix at node $u \in V$ after the nth step. The assignment of the random matrices $\mathbf{Q}_u$ to the nodes of the network can thus be described as a random process $\{\mathbf{U}_n\}_{n=1}^{\|\mathbf{x}\|}$, where $\mathbf{U}_n = \{\mathbf{Q}_{u;n}\}_{u \in V}$ is a random column configuration denoting the contents of each node after adding the nth column to the network graph. We shall use the notation $\mathbf{U}_0$ for the initial column configuration where no columns have been assigned yet to any node.

Let $\mathbf{S}_u$ denote the random matrix $[\mathbf{Q}_v]_{v \in \Gamma(u)}$ (as in Algorithm 2) and let R be the number of nodes u for which $\operatorname{rank}(\mathbf{S}_u) < k$. Recall that Algorithm 2 was based on the inequality

$$E(R) \;\triangleq\; E(R \mid \mathbf{U}_0) \;<\; \frac{|V|}{2\Delta_G} ,$$

which then allowed us to give a probabilistic estimate of $2E(R) < \frac{|V|}{\Delta_G}$ for the number of nodes u that required replacing $\mathbf{Q}_u$ by I_k. Instead, we compute here a sequence of column configurations $U_n = \{Q_{u;n}\}_{u \in V}$, $n = 1, 2, \ldots, \|\mathbf{x}\|$, such that

$$E(R \mid \mathbf{U}_n = U_n) \;\le\; E(R \mid \mathbf{U}_{n-1} = U_{n-1}) \; ; \tag{16}$$

in particular, we will have

$$E(R \mid \mathbf{U}_{||\mathbf{x}||} = U_{||\mathbf{x}||}) \;<\; \frac{|V|}{2\Delta_G}\;,$$

i.e., the number of nodes u for which B_u is set to I_k in (14) is guaranteed to be less than $\frac{|V|}{2\Delta_G}$.

In order to attain the inequality chain (16) we proceed as follows: Let U_0 be the empty column configuration and assume, by induction, that the column configuration U_{n-1} has been determined for some $n \geq 1$. Let v be a node which has been assigned less than x_v columns in U_{n-1}. We now determine the column which will be added to v to obtain U_n. This is done in a manner similar to the process described before for de-randomizing Algorithm 1: Set the first entry, b_1, of the added column to be 0, assume the other entries to be random bits, and compute the expected value, E_0, of R conditioned on $\mathbf{U}_{n-1} = U_{n-1}$ and on $b_1 = 0$. Now repeat the process with b_1 being set to 1, resulting in a conditional expected value E_1 of R. Since the two conditional expected values E_0 and E_1 average to $E(R \mid \mathbf{U}_{n-1} = U_{n-1})$, one of them must be at most that average. The first entry b_1 in the column added to v is set to the bit b for which E_b is the smallest. This process is now iterated for the second bit b_2 of the column added to v, resulting in two conditional expected values $E_{b_1,0}$ and $E_{b_1,1}$ of R, the smaller of which determines b_2. Continuing this way, we obtain a sequence of conditional expected values of R,

$$E(R \mid \mathbf{U}_{n-1} = U_{n-1}) \;\geq\; E_{b_1} \;\geq\; E_{b_1,b_2} \;\geq\; \ldots \;\geq\; E_{b_1,b_2,\ldots,b_k}\;,$$

thus determining the entire column added to v. Note that, indeed,

$$E(R \mid \mathbf{U}_n = U_n) \;=\; E_{b_1,b_2,\ldots,b_k} \;\leq\; E(R \mid \mathbf{U}_{n-1} = U_{n-1})\;,$$

in accordance with (16).

It remains to show how to compute the conditional expected values of R which are used to determine the column configurations U_n. It is easy to verify that, for any event $\mathcal{A}$,

$$E(R \mid \mathcal{A}) \;=\; \sum_{u \in V} \mathrm{Prob}\left\{\mathrm{rank}\,(\mathbf{S}_u) < k \mid \mathcal{A}\right\}. \tag{17}$$

Hence, the computation of the conditional expected values of R boils down to the following problem:

Let $\mathbf{S}$ *denote a* $k \times m$ *random matrix over* F_2 *whose first* l *columns, as well as the first* t *entries in its* $(l+1)$*st column, are preset, and the rest of its entries are independent random bits with probability* $\frac{1}{2}$ *of being zero. What is the probability of* $\mathbf{S}$ *having rank* k*?*

Let H denote the $k \times l$ matrix consisting of the first l (preset) columns of such a random matrix $\mathbf{S}$. Denote by $\mathbf{T}$ the matrix consisting of the first $l+1$ columns of $\mathbf{S}$ and by $\mathbf{W}$ the matrix consisting of the last $m-l-1$ columns of $\mathbf{S}$. Also, let the random variable ρ denote the rank of $\mathbf{T}$. Clearly, ρ may take only two values, namely, $\mathrm{rank}\,(H)$ or $\mathrm{rank}\,(H)+1$. We now show that

$$\mathrm{Prob}\left\{\mathrm{rank}\,(\mathbf{S}) < k \mid \rho = r\right\} \;=\; 1 \;-\; \prod_{i=0}^{k-r-1} (1 - 2^{i+l+1-m}) \;<\; 2^{k+l+1-m-r}\;. \tag{18}$$

Indeed, without loss of generality assume that the first r rows of $\mathbf{T}$ are linearly independent. We assume that the entries of $\mathbf{W}$ are chosen randomly row by row. Having selected the first r rows of $\mathbf{W}$, we thus obtain the first r rows in $\mathbf{S}$ which, in turn, are linearly independent. Next we select the $(r+1)$st row in $\mathbf{W}$. Clearly, there are 2^{m-l-1} choices for such a row, out of which one row will result in an $(r+1)$st row in $\mathbf{S}$ which is spanned by the first r rows in $\mathbf{S}$. Hence, given that the first r rows in $\mathbf{W}$ have been set, the probability that the first $r+1$ rows in $\mathbf{S}$ will be linearly independent is $1-2^{l+1-m}$. Conditioning upon the linear independence of the first $r+1$ rows in $\mathbf{S}$, we now select the $(r+2)$nd row in $\mathbf{W}$. In this case there are two choices of this row that yield an $(r+2)$nd row in $\mathbf{S}$ which is spanned by the first $r+1$ rows in $\mathbf{S}$. Hence, the probability of the first $r+2$ rows in $\mathbf{S}$ to be linearly independent (given the linear independence of the first $r+1$ rows) is $1-2^{l+2-m}$. In general, assuming linear independence of the first $r+i$ rows in $\mathbf{S}$, there are 2^i choices for the $(r+i+1)$st row of $\mathbf{W}$ that yield a row in $\mathbf{S}$ belonging to the linear span of the first $r+i$ rows in $\mathbf{S}$. The conditional probability for the first $r+i+1$ rows in $\mathbf{S}$ to be linearly independent thus becomes $1-2^{i+l+1-m}$. Equation (18) is obtained by re-iterating the process for all rows of $\mathbf{W}$.

To complete the computation of the probability of $\mathbf{S}$ having rank k, we need to calculate the probability of ρ being $r = \mathrm{rank}\,(H)$. Let H_t denote the first t rows of H with $r_t \triangleq \mathrm{rank}\,(H_t)$ and let $\mathbf{c}$ denote the first t (preset) entries of the $(l+1)$st column of $\mathbf{S}$ (or of $\mathbf{T}$). We now show that

$$\mathrm{Prob}\left\{\rho = r = \mathrm{rank}\,(H)\right\} \;=\; \begin{cases} 2^{r-r_t-k+t} & \text{if } \mathrm{rank}\,([H_t\,;\,\mathbf{c}]) = r_t \\ 0 & \text{if } \mathrm{rank}\,([H_t\,;\,\mathbf{c}]) = r_t+1 \end{cases}\;. \tag{19}$$

We first perform elementary operations on the columns of H so that (i) the first r_t columns in H_t are linearly independent whereas the remaining $l-r_t$ columns in H_t are zero, and (ii) the first r columns in H are linearly independent whereas the remaining $l-r$ columns in H are zero. Now, if $\mathbf{c}$ is not in the linear span of the columns of H_t, then $\rho = \text{rank}\,(\mathbf{T}) = \text{rank}\,(H)+1$. Otherwise, there are 2^{r-r_t} ways to select the last $k-t$ entries of the $(l+1)$st column of $\mathbf{T}$ to have that column spanned by the columns of H: each such choice corresponds to one linear combination of the last $r-r_t$ nonzero columns of H. Therefore, conditioning upon $\text{rank}\,([H_t\,;\,\mathbf{c}]) = r_t$, the probability of having $\text{rank}\,(T) = \text{rank}\,(H)$ equals 2^{r-r_t-k+t}.

Equations (18) and (19) can be now applied to $\mathbf{S}_u$ to compute the right-hand side of (17), where $\mathcal{A}$ stands for the event of having $n-1$ columns in $\mathbf{U}_n$ set to U_{n-1}, and t bits of the currently-added nth column set to $b_1, b_2, \ldots, b_t$.

4.4 Proof using the Lovász Local Lemma

In this section we present an alternative proof for the existence of a memory allocation $\mathbf{x}$ satisfying (3) and of $k \times x_u$ binary matrices B_u for the encoding mappings $\mathcal{E}_u : \mathbf{w} \mapsto \mathbf{w}B_u\;,\quad u \in V$. The techniques used will turn out to be useful in Section refvariations. To this end, we make use of the following lemma.

Lemma 5. (The Lovász Local Lemma [5][19]). *Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ be events in an arbitrary probability space. Suppose that each event $\mathcal{A}_i$ is mutually independent of a set of all, but at most δ, events $\mathcal{A}_j$ and that* $\text{Prob}\,\{\mathcal{A}_i\} \le p$ *for all* $1 \le i \le n$. *If* $e\,p\,\delta < 1$, *then* $\text{Prob}\left\{\bigwedge_{i=1}^n \overline{\mathcal{A}}_i\right\} > 0$.

In most applications of the lemma (as well as in its use in the sequel), the $\mathcal{A}_i$'s stand for 'bad' events; hence, if the probability of each bad event is at most p, and if the bad events are not-too-dependent of one another (in the sense stated in the lemma), there is a strictly positive probability that none of the bad events will occur. However, this probability might be exponentially small. Recently, Beck [2] has proposed a constructive technique that can be used in most applications of the lemma for finding an element of $\bigwedge_{i=1}^n \overline{\mathcal{A}}_i$ (see also [1]). We shall be mainly concentrating on an existence proof, as the construction will then follow by a technique similar to the one in [2].

We start by using the local lemma to present an alternative proof of Theorem 3. Given a network graph $G = (V, E_G)$ and an integer m, we construct a directed graph $H = (V, E_H)$ which satisfies the following four properties:

(i) there is an edge $u \to v$ in H whenever u is adjacent to v in G;

(ii) there are no parallel edges in H;

(iii) each node in H has the same in-degree $\Delta_H = O(\Delta_G)$;

(iv) each node in H has an out-degree which is bounded from above by $\Lambda_H = O(\Delta_G)$.

Lemma 6. *A directed graph H satisfying (i)–(iv) always exists.*

Proof. When $\Delta_G > \frac{1}{2}|V|$ we take H as the complete graph (i.e., the adjacency matrix A_H is the all-one matrix and $\Delta_H = \Lambda_H = |V| < 2\Delta_G$). Otherwise, we construct H out of G as follows: Make every self loop in G a directed edge in H, and change all other edges in G into two anti-parallel edges in H. Finally, adjoin extra edges (not parallel to existing ones) to have in-degree $\Delta_H = \Delta_G$ and out-degree $\leq \Lambda_H = 2\Delta_G$ at each node in H. To realize this last step, we scan the nodes of H and add incoming edges to nodes whose in-degree is less than Δ_G — one node at a time. Let u be such a node and let $\overline{\Gamma}(u)$ be the set of nodes in H with no outgoing edges that terminate at u. We show that at least one of the nodes in $\overline{\Gamma}(u)$ has out-degree less than $2\Delta_G$, thus allowing us to adjoin a new incoming edge to u from that node. The proof then continues inductively. Now, since the in-degree of each node in H at each stage is at most Δ_G, the total number of edges outgoing from nodes in $\overline{\Gamma}(u)$ is bounded from above by $\Delta_G \cdot (|V| - 1)$. On the other hand, $\overline{\Gamma}(u)$ contains at least $|V| - \Delta_G + 1$ nodes. Hence, there exists at least one node in $\overline{\Gamma}(u)$ whose out-degree is at most $(\Delta_G \cdot (|V| - 1))/(|V| - \Delta_G + 1)$; this number, in turn, is less than $2\Delta_G$ whenever $\Delta_G \leq \frac{1}{2}|V|$.□

Proof of Theorem 3 using the Local Lemma. Let $\mathbf{z}$ be a solution the linear programming problem LP(G) of (2). By property (i), $\mathbf{z}$ satisfies the inequality $A_H \mathbf{z} \geq \mathbf{1}$. Re-define β_G to be $8e\Delta_G^2$ (and ℓ accordingly to be $m + c_1 \cdot L\{\beta_G, m\}$), and let $\mathbf{X}$ be obtained by (5)–(7). By Proposition 1 we have

$$\text{Prob}\left\{(A_G\mathbf{X})_u \; < \; m\right\} \; \leq \; \frac{1}{\beta_G} \, , \tag{20}$$

and by property (ii) and Proposition 2 we have,

$$\text{Prob}\Big\{(A_H\mathbf{X})_u \ > \ \ell \cdot (A_H\mathbf{z})_u + c_2 \cdot L\left\{\beta_G, \ell \cdot (A_H\mathbf{z})_u\right\}\Big\} \ \leq \ \frac{1}{\beta_G} \tag{21}$$

for each node $u \in V$.

For every $u \in V$ define the event $\mathcal{A}_u$ as

$$\mathcal{A}_u \ \triangleq \ \left\{ \begin{array}{c} (A_G\mathbf{X})_u \ < \ m \\ \text{or} \\ (A_H\mathbf{X})_u \ > \ \ell \cdot (A_H\mathbf{z})_u \ + \ c_2 \cdot L\Big\{\beta_G, \ell \cdot (A_H\mathbf{z})_u\Big\} \end{array} \right\} . \tag{22}$$

By (20) and (21) it follows that $\text{Prob}\left\{\mathcal{A}_u\right\} \leq 2/\beta_G < 1/(4e\Delta_G^2)$. For every node u in H, denote by $\Gamma_{\text{out}}(u)$ the set of terminal nodes of the edges outgoing from u in H. Then, for every node u, the event $\mathcal{A}_u$ is mutually independent of all events $\mathcal{A}_v$ such that $\Gamma_{\text{out}}(u) \cap \Gamma_{\text{out}}(v) = \emptyset$. Hence, by properties (iii) and (iv), each $\mathcal{A}_u$ depends on at most $\Lambda_H(\Delta_H - 1) + 1 \leq 4\Delta_G^2$ events A_v and, therefore, by Lemma 5 there exists a nonnegative integer vector $\mathbf{x}$ satisfying both

$$(A_G\mathbf{x})_u \ \geq \ m \tag{23}$$

and

$$(A_H\mathbf{x})_u \ \leq \ \ell \cdot (A_H\mathbf{z})_u \ + \ c_2 \cdot L\Big\{\beta_G, \ell \cdot (A_H\mathbf{z})_u\Big\} \tag{24}$$

for all $u \in V$.

We now show that $\|\mathbf{x}\|$ satisfies the inequality

$$\frac{\|\mathbf{x}\|}{\|\ell \cdot \mathbf{z}\|} \ \leq \ 1 \ + \ c_2 \left(\frac{\log_e \beta_G}{\ell} \ + \ \sqrt{\frac{\log_e \beta_G}{\ell}} \right) . \tag{25}$$

By (24) and the fact that each node in H has in-degree Δ_H we have,

$$\begin{aligned} \|\mathbf{x}\| \ &= \ \frac{1}{\Delta_H} \sum_{u \in V} (A_H\mathbf{x})_u \\ &\leq \ \frac{\ell}{\Delta_H} \sum_{u \in V} (A_H\mathbf{z})_u \ + \ \frac{c_2}{\Delta_H} \sum_{u \in V} L\Big\{\beta_G, \ell \cdot (A_H\mathbf{z})_u\Big\} \\ &\leq \ \ell \cdot \|\mathbf{z}\| \ + \ c_2 \left(\frac{|V|}{\Delta_H} \log_e \beta_G \ + \ \frac{1}{\Delta_H} \sum_{u \in V} \sqrt{\ell \cdot (A_H\mathbf{z})_u \cdot \log_e \beta_G} \right) . \end{aligned}$$

Now, by the Cauchy-Schwarz inequality,

$$\sum_{u\in V} \sqrt{(A_H \mathbf{z})_u} \;\le\; \sqrt{|V|}\cdot\sqrt{\sum_{u\in V}(A_H\mathbf{z})_u} \;=\; \sqrt{|V|\cdot\Delta_H\cdot\|\mathbf{z}\|}$$

and, therefore,

$$\|\mathbf{x}\| \;\le\; \ell\cdot\|\mathbf{z}\| \;+\; c_2\left(\frac{|V|}{\Delta_H}\log_e\beta_G \;+\; \sqrt{\frac{|V|}{\Delta_H}}\cdot\sqrt{\|\mathbf{z}\|\cdot\ell\cdot\log_e\beta_G}\right)\;.$$

Inequality (25) is now obtained by bounding $|V|/\Delta_H$ from above by $\|\mathbf{z}\|$. Finally, Theorem 3 is a consequence of both (23) and (25). □

We now turn to defining the encoding and decoding mappings for a given instance (G, k). To this end, we shall make use of the following lemma.

Lemma 7. *Let $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_t$ be subsets of $\langle n\rangle \triangleq \{1, 2, \ldots, n\}$, each $\mathcal{S}_i$ of size $\ge s$, and no subset intersects more than δ subsets. Let q be a power of a prime and let k be a nonnegative integer satisfying*

$$e\cdot\delta\cdot q^{-s-1} \;<\; q^{-k}\;.$$

Then there exists an (n, q^k) linear code over $\Phi = GF(q)$ which is separable with respect to each $\mathcal{S}_i$.

Proof. We construct inductively $l\times n$ matrices B_l, $1\le l\le k$, each generating a linear code which is separable with respect to every $\mathcal{S}_i$; that is, each $(B_l)_{\mathcal{S}_i}$ has rank l. Start with an all-one $1\times n$ matrix B_1. As the induction step, assume that a matrix B_{l-1}, with the above property, has already been constructed for some $l\le k$. We are now to append an lth row to B_{l-1}.

Given such a matrix B_{l-1}, a row vector in Φ^n is 'good' with respect to $\mathcal{S}_i$ if, when appended to B_{l-1}, it yields a matrix B_l such that $(B_l)_{\mathcal{S}_i}$ has rank l; otherwise, a row vector is 'bad' with respect to that $\mathcal{S}_i$. Now, for each i, the row span of $(B_{l-1})_{\mathcal{S}_i}$ consists of q^{l-1} vectors in $\Phi^{|\mathcal{S}_i|}$; this means that the probability of a randomly selected row to be bad with respect to $\mathcal{S}_i$ is $q^{-|\mathcal{S}_i|+l-1}\le q^{-s-1+k} < 1/(e\cdot\delta)$. Similarly, if $\mathcal{S}_i\bigcap\mathcal{S}_j=\emptyset$, then the probability of a randomly selected row to be bad with respect to both $\mathcal{S}_i$ and $\mathcal{S}_j$ is $q^{-|\mathcal{S}_i|-|\mathcal{S}_j|+2(l-1)}$. Therefore, when $\mathcal{S}_i\bigcap\mathcal{S}_j=\emptyset$, the events "the row vector is bad with respect to $\mathcal{S}_i$" and "the row vector is bad with respect to $\mathcal{S}_j$" are independent; thus, by Lemma 5 we are guaranteed to have a row vector in Φ^n which is good with respect to every $\mathcal{S}_i$. This vector can now be appended to B_{l-1} to obtain a generator matrix B_l with $(B_l)_{\mathcal{S}_i}$ having rank l for all i. $\square$

Let $\mathbf{x}$ be the integer vector guaranteed by Theorem 3 for $m = k+2\lceil\log_2 \Delta_G\rceil+1$. Partition the set $\langle\|\mathbf{x}\|\rangle$ into $|V|$ (disjoint) subsets $\mathcal{Q}_u$ with $|\mathcal{Q}_u| = x_u$ and let $\mathcal{S}_u \triangleq \cup_{v\in\Gamma(u)}\mathcal{Q}_v$, $u \in V$. We have $|\mathcal{S}_u| = (A_G\mathbf{x})_u \geq m = k + 2\lceil\log_2 \Delta_G\rceil + 1$ and, therefore, $e \cdot \Delta_G^2 \cdot 2^{-|\mathcal{S}_u|-1} < 2^{-k}$. Furthermore, each $\mathcal{S}_u$ intersects at most $(\Delta_G - 1)^2 + 1$ sets $\mathcal{S}_v$; hence, by Lemma 7 there exists a linear $(\|\mathbf{x}\|, 2^k)$ code over F_2 which is separable with respect to each $\mathcal{S}_u$. For each $u \in V$ let $B_u \triangleq (B)_{\mathcal{Q}_u}$; i.e., B_u is the $k \times x_u$ matrix consisting of all columns of B indexed by $\mathcal{Q}_u$. We now use this to define the encoding and decoding mappings as in Section 4.2.

4.5 Variations on the memory cost measure

The techniques used in Section 4.4 can be adapted to obtain file distribution schemes $(G, k) \mapsto \chi(G, k)$ which are close to optimal with respect to other variants of the memory cost measure. For instance, consider the problem where for every instance (G, k), we are looking for a file distribution protocol $\chi(G, k)$ whose memory allocation $\mathbf{x}$ satisfies the following two criterions:

(i) The largest component $x_{\max}$ of $\mathbf{x}$ is the smallest possible.

(ii) Among all file distribution protocols that satisfy (i), we take one whose memory size $\|\mathbf{x}\|$ is the smallest.

This variant of our original problem might suit cases where, say, each node in the network graph (as opposed to some 'network manager') needs to pay for its own memory. Since the respective decision problem is NP-complete, we need to look for approximations to the optimal solution.

Given a network graph $G = (V, E)$ and an integer k, we proceed as follows. Let $\Delta_{\min}$ be $\min_{u\in V} \Delta(u)$. It is clear that $\lceil k/\Delta_{\min}\rceil$ is a lower bound on the largest component of $\mathbf{x}$. Set

$\alpha = \lceil k/\Delta_{\min}\rceil/k$ and consider the following linear program:

$$\mathrm{LP}(G;\alpha): \qquad \begin{array}{c} \rho_{G;\alpha} \;=\; \min\; \|\mathbf{z}\|\;, \\ \text{ranging over all rational } \mathbf{z} = [z_u]_{u\in V} \text{ such that} \\ A_G\mathbf{z} \;\ge\; \mathbf{1} \quad \text{and} \quad 0 \;\le\; z_u \;\le\; \alpha \quad \text{for every } u \in V\;. \end{array} \tag{26}$$

Next, we set $\beta_G = 12e\Delta_G^2$, $m = k + 2\lceil\log_2 \Delta_G\rceil + 1$, and $\ell = m + c_1 \cdot L\{\beta_G, m\}$. Now, let $\mathbf{X} = [X_u]_{u\in V}$ be obtained by (5)–(7) and re-define the events $\mathcal{A}_u$ in (22) as

$$\mathcal{A}_u \;\triangleq\; \left\{ \begin{array}{c} (A_G\mathbf{X})_u \;<\; m \\ \text{or} \\ X_u \;>\; \ell\cdot z_u \;+\; c_2 \cdot L\Big\{\beta_G, \ell\cdot z_u\Big\} \\ \text{or} \\ (A_H\mathbf{X})_u \;>\; \ell\cdot (A_H\mathbf{z})_u \;+\; c_2\cdot L\Big\{\beta_G, \ell\cdot (A_H\mathbf{z})_u\Big\} \end{array} \right\}.$$

By (20) and (21) we have $\mathrm{Prob}\{\mathcal{A}_u\} \le 3/\beta_G < 1/(4e\Delta_G^2)$. Following along the lines of Section 4.4, the Lovász Local Lemma now guarantees a file distribution protocol with memory allocation $\mathbf{x}$ whose maximal component $x_{\max}$ and size $\|\mathbf{x}\|$ satisfy both

$$\frac{x_{\max}}{\alpha\cdot k} \;=\; 1 \;+\; O\left(\max\left\{\frac{\log \Delta_G}{k}\;;\;\sqrt{\frac{\log \Delta_G}{k}}\right\}\right)$$

and

$$\frac{\|\mathbf{x}\|}{\rho_{G;\alpha}\cdot k} \;=\; 1 \;+\; O\left(\max\left\{\frac{\log \Delta_G}{k}\;;\;\sqrt{\frac{\log \Delta_G}{k}}\right\}\right)\;.$$

Both $x_{\max}$ and $\|\mathbf{x}\|$ approach there optimal values as k becomes larger than $\log \Delta_G$.

5 The integer programming bound is not tight

In Section 4 we presented an algorithm for finding a constructive file distribution scheme $(G,k) \mapsto \chi(G,k)$ such that the ratio between the memory size $|\chi(G,k)|$ and $\rho_G\cdot k$ approaches 1 as the ratio $k/\log \Delta_G$ tends to infinity. In this section we present a family of network graphs $\{G_l\}_{l=1}^{\infty}$ for which a file size of $\log \Delta_{G_l}$ is, indeed, a critical point: there exists a sequence of file sizes $k_l \ge \log_2 \Delta_{G_l}$, $l = 1, 2, \ldots$, for which the ratios $M(G_l,k_l)/J(G_l,k_l)$ (and, therefore, $M(G_l,k_l)/(\rho_{G_l}\cdot k_l)$) are bounded away from 1.

For integers m and l, $m \geq l$, define the network graphs $G_{m,l} = (V_{m,l}, E_{m,l})$ as follows: Let U_m be a set of m elements (say, $U_m = \langle m \rangle$) and let $\mathcal{W}_{m,l}$ consist of all subsets of U_m of size l. Set $V_{m,l} = U_m \cup \mathcal{W}_{m,l}$ and draw an edge between two nodes $u, v \in V_{m,l}$ in any of the following cases: (i) both u and v are in U_m (i.e., U_m is a clique); (ii) $u \in U_m$, $v \in \mathcal{W}_{m,l}$, and $u \in v$; (iii) $u = v$ (self loops).

First, we verify that $\rho_{G_{m,l}} = m/l$. Let $\mathbf{z} = [z_u]_{u \in V_{m,l}}$ be a nonnegative real vector satisfying $A_{G_{m,l}}\mathbf{z} \geq \mathbf{1}$ and $\|\mathbf{z}\| = \rho_{G_{m,l}}$. Without loss of generality, we can assume that $z_v = 0$ for every $v \in \mathcal{W}_{m,l}$; otherwise, "remove" the quantity z_v from such a node v and add it to the value z_u at some node $u \in \Gamma(v) - \{v\} \subseteq U_m$. This change results in a new nonnegative vector $\tilde{\mathbf{z}}$ with the same norm as $\mathbf{z}$ and which satisfies $A_{G_{m,l}}\tilde{\mathbf{z}} \geq \mathbf{1}$.

Now, rename the nodes of U_m to have $U_m = \langle m \rangle$ and $z_1 \leq z_2 \leq \ldots \leq z_m$. For the node $\langle l \rangle \in \mathcal{W}_{m,l}$ we have

$$\sum_{u=1}^{l} z_u \ = \ \left(A_{G_{m,l}}\mathbf{z}\right)_{\langle l \rangle} \ \geq \ 1 \ ,$$

and, therefore, $z_u \geq z_l \geq 1/l$ for every node $u \geq l$ in U_m. Hence,

$$\rho_{G_{m,l}} \ = \ \|\mathbf{z}\| \ = \ \sum_{u=1}^{l} z_u \ + \ \sum_{u=l+1}^{m} z_u \ \geq \ 1 \ + \ \frac{m-l}{l} \ = \ \frac{m}{l} \ .$$

Setting $\mathbf{z} = [z_u]_{u \in V_{m,l}}$ to

$$z_u = \begin{cases} 1/l & \text{if} \quad u \in U_m \\ 0 & \text{otherwise} \end{cases} \ ,$$

we obtain the equality $\rho_{G_{m,l}} = m/l$. Furthermore,

$$J(G_{m,l}, r \cdot l) \ = \ \rho_{G_{m,l}} \cdot r \cdot l \ = \ r \cdot m \tag{27}$$

for every positive integer r. A similar analysis for a similar set-covering problem appears also in [7].

In the forthcoming discussion we will be concentrating on two types of network graphs $G_{m,l}$, namely:

- $G_l \triangleq G_{2l,l}$, in which case $\rho_{G_l} = 2$ and

$$\log_2 \Delta_{G_l} \ = \ \log_2 \left(2l + \binom{2l-1}{l-1}\right) \ \leq \ 2l \ ;$$

- $H_l \triangleq G_{2^l,l}$, in which case $\rho_{H_l} = 2^l/l$ and

$$\log_2 \Delta_{H_l} = \log_2 \left(2^l + \binom{2^l - 1}{l-1} \right) \le l^2, \quad l \ge 2 .$$

The proof of the next proposition makes use of the following known lemma.

Lemma 8. (The sphere-packing or the Hamming bound [12, Ch. 1]). *Let Φ be an alphabet of q elements. There exists an (n, K) code of minimum distance $2t+1$ over Φ only if*

$$K \cdot \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \ \le \ q^n .$$

Proposition 3. *For any fixed positive integer r,*

$$\lim_{l \to \infty} \frac{M(G_l, r \cdot l)}{\rho_{G_l} \cdot r \cdot l} \ = \ \lim_{l \to \infty} \frac{M(G_l, r \cdot l)}{J(G_l, r \cdot l)} \ \ge \ 1 + \frac{1}{2r} .$$

Proof. Set $k = rl$ for some positive integer r and let $\mathbf{x}$ be the memory allocation of a file distribution protocol χ for (G_l, k) of memory size $|\chi| = \|\mathbf{x}\| = M(G_l, k)$. We assume that $x_v = 0$ for every $v \in \mathcal{W}_{2l,l}$ and that the nodes of $U_{2l} = \langle 2l \rangle$ are renamed to have $x_1 \le x_2 \le \ldots \le x_{2l}$. Letting $h \triangleq x_{l+2}$, we obtain,

$$\begin{array}{rcccccccc} M(G_l, k) & = & \|\mathbf{x}\| & = & \sum_{u=1}^{l} x_u & + & x_{l+1} & + & \sum_{u=l+2}^{2l} x_u \\ & & & \ge & r \cdot l & + & r & + & (l-1)\, h , \end{array} \tag{28}$$

where the inequality follows from $\sum_{u=1}^{l} x_u = (A_{G_l}\mathbf{x})_{\langle l \rangle} \ge k = rl$ which, in turn, implies the inequalities $x_{l+1} \ge x_l \ge r$.

For a file $\mathbf{w} \in F_2^k$, let $\mathbf{c}_{\mathbf{w}}$ denote the encoded memory contents $[\, \mathcal{E}_u(\mathbf{w})\,]_{u=1}^{l+2}$ as determined by the file distribution protocol χ. We now regard the set

$$C \ \triangleq \ \left\{ \mathbf{c}_{\mathbf{w}} \ \middle| \ \mathbf{w} \in F_2^k \right\}$$

as an $(l+2, 2^k)$ code over an alphabet of $q \triangleq 2^h$ elements. The code C must be separable with respect to any subset of $\langle l+2 \rangle$ of size l, or else there would be nodes in $\mathcal{W}_{2l,l}$ that could not reconstruct the file $\mathbf{w}$. Hence, by Lemma 1, the minimum distance of C is at least 3, which readily implies by Lemma 8 the inequality

$$2^k \cdot \Big(1 \ + \ (l+2)(q-1) \Big) \ \le \ q^{l+2} .$$

Substituting $k = rl$ and $q = 2^h$, and noting that $2^h - 1 \geq 2^{h-1}$, we obtain,

$$(l+2) \cdot 2^{rl+h-1} \;\leq\; 2^{(l+2)\,h} \;,$$

or

$$h \;\geq\; \left\lceil \frac{\log_2(l+2) + rl - 1}{l+1} \right\rceil \;=\; r \;+\; \left\lceil \frac{\log_2(l+2) - r - 1}{l+1} \right\rceil \;.$$

Hence, for fixed r and for sufficiently large l we must have $h \geq r+1$. Combining this lower bound on h with (28) yields the inequality

$$\lim_{l\to\infty} \frac{M(G_l, r \cdot l)}{J(G_l, r \cdot l)} \;\geq\; \lim_{l\to\infty} \frac{r\,(l+1) + (r+1)(l-1)}{2rl} \;=\; 1 \;+\; \frac{1}{2r}$$

which, with (27), concludes the proof. $\square$

Corollary 2. *For $k_l \triangleq 2l \geq \log_2 \Delta_{G_l}$,*

$$\lim_{l\to\infty} \frac{M(G_l, k_l)}{\rho_{G_l} \cdot k_l} \;=\; \lim_{l\to\infty} \frac{M(G_l, k_l)}{J(G_l, k_l)} \;\geq\; \frac{5}{4} \;.$$

Corollary 2 exhibits the fact that a file size of $\log \Delta_{G_l}$ is a critical point in the following strong sense: For $k_l = 2l \geq \log_2 \Delta_{G_l}$, the size of any memory allocation for (G_l, k_l) must be bounded away from $\rho_{G_l} \cdot k_l$, not because of a gap between $J(G_l, k_l)$ and $\rho_{G_l} \cdot k_l$, but rather because of a gap between $M(G_l, k_l)$ and $J(G_l, k_l)$.

We point out that, as a counterpart of Proposition 3, we also have

$$\lim_{l\to\infty} \frac{M(G_l, r \cdot l)}{J(G_l, r \cdot l)} \;\leq\; 1 + \frac{2}{r} \;,$$

the proof of which is based on the following result.

Lemma 9. (The Gilbert-Varshamov bound [3, pp. 321–322]). *Let Φ be an alphabet of q elements and let n, K, and d be positive integers satisfying*

$$(K-1) \cdot \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \;<\; q^n \;.$$

Then, there exists an (n, K) code of minimum distance d over Φ.

Set $n = 2l$, $K = 2^{rl}$, $d = l + 1$, $h = r + 2$, and $q = 2^h$; these values satisfy the equality $K \cdot 2^n \cdot q^{d-1} = q^n$ and, therefore, by Lemma 1 and Lemma 9 there exists a $(2l, 2^{rl})$ code C over F_2^h which is separable with respect to any subset of $\langle 2l \rangle$ of size l. Assign the coordinates (over F_2^h) of C to the nodes $u \in U_{2l}$ of G_l and map the files $\mathbf{w} \in F_2^{rl}$ into distinct codewords of C. This protocol allows every node in G_l to reconstruct any such file $\mathbf{w}$, requiring a total memory size of $2(r+2)l$ (compared to $J(G_l, r \cdot l) = 2rl$).

Remark 5. It can be readily verified that $J(G_{m,l}, k) \geq m - l + k$ for every m, l, and k, and, in particular, $J(G_l, 1) \geq l + 1 \geq \frac{1}{4}\rho_{G_l} \log_2 \Delta_{G_l}$. Hence, any file distribution protocol for (G_l, k) based on segmentation will be at least $\frac{1}{4} \log_2 \Delta_{G_l}$ times larger than the linear programming bound $\rho_{G_l} \cdot k$, *even when k tends to infinity* (see Example 1 and Remark 3).●

For file sizes k which are smaller than $\log \Delta_G$, one can find examples where the ratio between $M(G, k)$ and $J(G, k)$ is even larger than stated in Proposition 3. We demonstrate this for the network graphs $H_l = G_{2^l, l}$ in the next proposition, making use of the following lemma.

Lemma 10. (The Plotkin bound [3, p. 315]). *Let C be an (n, K) code of minimum distance d over an alphabet of q elements. Then,*

$$\frac{1}{q} \leq 1 - \frac{d}{n}\left(1 - \frac{1}{K}\right) .$$

Proposition 4.

$$\frac{M(H_l, k)}{J(H_l, k)} \sim \begin{cases} 1 & \text{if } l \mid k \text{ and } k \geq l^2 \\ l^2/k & \text{if } l \mid k \text{ and } l \leq k < l^2 \\ k & \text{if } k < l \end{cases} ,$$

where $f_l(k) \sim g_l(k)$ stands for $\lim_{l \to \infty} f_l(k)/g_l(k) = 1$ uniformly on k.

In particular, when $k = l$, the ratio $M(H_l, k)/J(H_l, k)$ is approximately l which, in turn, is at least $\sqrt{\log_2 \Delta_{H_l}}$.

Proof. We distinguish between the three ranges of k stated in the proposition.

Case 1: $k = rl$ for some integer $r \geq l$. By (27) we have $J(H_l, k) = \rho_{H_l} \cdot k = r \cdot 2^l$. In fact, we also have $M(H_l, k) = J(H_l, k)$: since $r \geq l$, we can construct a $(2^l, 2^{rl})$ generalized Reed-Solomon code C_{RS} over $GF(2^r)$, which is separable with respect to any subset of $\langle 2^l \rangle$ of size l [12, Chs. 10–11] (compare with Section 3.3). Assign the coordinates, over $GF(2^r)$, of C_{RS} to the nodes $u \in U_{2^l}$ of H_l and map the files $\mathbf{w} \in F_2^k$ into distinct codewords of C_{RS}. By the separability of C_{RS} every node in H_l can readily reconstruct any file $\mathbf{w} \in F_2^k$.

Case 2: $k = rl$ for some strictly positive integer $r < l$. Let $\mathbf{x}$ be the memory allocation of a file distribution protocol for (H_l, k) of memory size $\|\mathbf{x}\| = M(H_l, k)$. Again, we assume that $x_v = 0$ for $v \in \mathcal{W}_{2^l,l}$ and that the nodes in U_{2^l} are renamed to have $x_1 \leq x_2 \leq \ldots \leq x_{2^l}$. Defining $n \triangleq \lceil 2^l/l \rceil$ and $h \triangleq x_{n+1}$, we obtain,

$$M(H_l, k) \;=\; \|\mathbf{x}\| \;\geq\; \sum_{u=n+1}^{2^l} x_u \;\geq\; (2^l - n)\, h \;>\; h \cdot 2^l \cdot \Big(1 - \frac{1}{l} - \frac{1}{2^l}\Big) \tag{29}$$

(compare with (28)).

To bound h from below, we regard the set

$$C \;\triangleq\; \Big\{ \, [\, \mathcal{E}_u(\mathbf{w})\,]_{u=1}^{n} \;\Big|\; \mathbf{w} \in F_2^k \, \Big\}$$

as an $(n, 2^k)$ code over an alphabet of $q \triangleq 2^h$ elements. Since C is separable with respect to any subset of $\langle n \rangle$ of size l, its minimum distance must be, by Lemma 1, at least $n - l + 1$. This, in turn, implies by Lemma 10 the inequality

$$\frac{1}{2^h} \;\leq\; 1 \;-\; \Big(1 - \frac{l-1}{n}\Big)\Big(1 - \frac{1}{2^k}\Big) \;\leq\; \frac{l-1}{n} \;+\; \frac{1}{2^k}\,. \tag{30}$$

Since $2^k \geq 2^l \geq n$, we thus have,

$$\frac{1}{2^h} \;\leq\; \frac{l}{n} \;\leq\; \frac{l^2}{2^l}\,,$$

or,

$$h \;\geq\; l \;-\; 2\log_2 l\,.$$

Combining the last inequality with (29) yields

$$M(H_l, k) \;>\; 2^l \cdot (l - 2\log_2 l)\Big(1 - \frac{1}{l} - \frac{1}{2^l}\Big) \;=\; l \cdot 2^l \cdot \Big(1 - o(1)\Big)\,, \tag{31}$$

where $o(1)$ stands for an expression, independent of k, which tends to zero as l goes to infinity. Recalling that $J(H_l, k) = r \cdot 2^l$, we thus obtain,

$$\frac{M(H_l, k)}{J(H_l, k)} \;\geq\; \frac{l}{r} \cdot \Big(1 - o(1)\Big) \;=\; \frac{l^2}{k} \cdot \Big(1 - o(1)\Big)\,. \tag{32}$$

The bounds (31) and (32) are definitive up to a multiplying factor of $1-o(1)$: An upper bound $M(H_l,k) \le l \cdot 2^l$ is obtained by assigning the coordinates of a $(2^l, 2^{rl})$ generalized Reed-Solomon code over $GF(2^l)$ to the nodes $u \in U_{2^l}$ of H_l; such a code is separable with respect to any subset of $\langle 2^l \rangle$ of size r and, therefore, with respect to any subset of size l.

Case 3: $k < l$. Let n and h be defined as in case 2. Noting that (29) and (30) still apply, we have,

$$\frac{1}{2^h} \;\le\; \frac{l-1}{n} \;+\; \frac{1}{2^k} \;<\; 2\,\max\left\{\frac{l^2}{2^l}\,;\,\frac{1}{2^k}\right\}\,,$$

i.e.,

$$h \;\ge\; \min\Big\{l - \lceil 2\log_2 l \rceil\;;\; k\Big\} \;\ge\; \left(1 - \frac{\lceil 2\log_2 l\rceil}{l}\right)\cdot k\,.$$

Combining the last inequality with (29) yields

$$M(H_l,k) \;\ge\; k\cdot 2^l \cdot \Big(1 - o(1)\Big)\,. \tag{33}$$

Turning to $J(H_l,k)$, for $k<l$ we have $J(H_l,k) \le 2^l - l + k$ (and, by Remark 5 we have, in fact, equality): it is easy to verify that the integer vector $\mathbf{y} = [y_u]_{u\in V_{2^l,l}}$ which is defined by

$$y_u = \begin{cases} 1 & \text{if } u \in U_{2^l} \quad \text{and} \quad u \ge l-k+1 \\ 0 & \text{otherwise} \end{cases}\,,$$

satisfies the inequality $A_{H_l}\mathbf{y} \ge k\cdot\mathbf{1}$. Hence, by (33) we obtain

$$\frac{M(H_l,k)}{J(H_l,k)} \;\ge\; k\cdot\Big(1-o(1)\Big)\,. \tag{34}$$

Again, the bounds (33) and (34) are definitive as we can simply replicate the file $\mathbf{w}$ into each node $u \in U_{2^l}$ of H_l, requiring a memory size of $k\cdot 2^l$. $\square$

Acknowledgment

We thank Noga Alon and Cynthia Dwork for the many helpful discussions and the anonymous referee for the useful comments and the suggestion that we consider other variations on the memory cost measure.

Appendix

The proofs of Propositions 1 and 2 make use of the following lemma.

Lemma 11. *Let* $\mathbf{a} = [a_u]_{u \in V}$ *and* $\mathbf{p} = [p_u]_{u \in V}$ *be real vectors in* $[0,1]^{|V|}$ *and let* $\mathbf{Y} = [Y_u]_{u \in V}$ *be a vector of independent random variables over* $\{0,1\}$ *with* $\mathrm{Prob}\{Y_u = 1\} = p_u$. *Then,*

(a) for every $\delta \in [0,1)$ *and* $\tau \le \mathbf{a} \cdot \mathbf{p}$,

$$\mathrm{Prob}\Big\{\mathbf{a} \cdot \mathbf{Y} \ \le\ (1-\delta)\tau\Big\} \ \le\ \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\tau} ;$$

(b) for every $\delta \ge 0$ *and* $\tau \ge \mathbf{a} \cdot \mathbf{p}$,

$$\mathrm{Prob}\Big\{\mathbf{a} \cdot \mathbf{Y} \ \ge\ (1+\delta)\tau\Big\} \ \le\ \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\tau} .$$

Proof. Lemma 11 is proved in [15] and [16]. Part (b) of the lemma appears as is in [15] and for the sake of completeness we include the proof of part (a) here.

For a real random variable Z and constants $\gamma \ge 0$ and b, we have

$$\mathrm{Prob}\Big\{Z \ \le\ b\Big\} \ \le\ E\left(e^{\gamma(b-Z)}\right) ,$$

an inequality known as the Chernoff bound. Letting $Z = \mathbf{a} \cdot \mathbf{Y}$ and $b = (1-\delta)\tau$, we obtain, for every $\gamma \ge 0$,

$$\begin{aligned}
&\mathrm{Prob}\Big\{\mathbf{a} \cdot \mathbf{Y} \ \le\ (1-\delta)\tau\Big\} \\
&\le\ E\left(e^{\gamma((1-\delta)\tau \,-\, \mathbf{a}\cdot\mathbf{Y})}\right) \ =\ e^{\gamma(1-\delta)\tau} E\left(\prod_{u \in V} e^{-\gamma a_u Y_u}\right) \\
&=\ e^{\gamma(1-\delta)\tau} \prod_{u \in V} E\left(e^{-\gamma a_u Y_u}\right) \ =\ e^{\gamma(1-\delta)\tau} \prod_{u \in V} \left(1 - p_u + p_u e^{-\gamma a_u}\right) .
\end{aligned}$$

Substituting $t = e^{-\gamma}$ yields, for every $t \in (0,1]$,

$$\mathrm{Prob}\Big\{\mathbf{a} \cdot \mathbf{Y} \ \le\ (1-\delta)\tau\Big\} \ \le\ \alpha(t) , \tag{35}$$

where

$$\begin{aligned}\alpha(t) &\triangleq t^{-(1-\delta)\tau} \prod_{u \in V} \left(1 - p_u \left(1 - t^{a_u}\right)\right) \leq t^{-(1-\delta)\tau} \prod_{u \in V} \exp\left\{-p_u \left(1 - t^{a_u}\right)\right\} \\ &= t^{-(1-\delta)\tau} \exp\left\{- \sum_{u \in V} p_u \left(1 - t^{a_u}\right)\right\} .\end{aligned}$$

Now, for $a_u \in [0,1]$ and $t \in (0,1]$ we have $1 - t^{a_u} \geq a_u(1-t) \geq 0$. Therefore,

$$\alpha(t) \leq t^{-(1-\delta)\tau} \exp\left\{- \sum_{u \in V} a_u p_u (1-t)\right\} \leq \left(t^{-(1-\delta)} e^{t-1}\right)^{\tau}$$

which, for $t = 1 - \delta$ becomes

$$\alpha(1-\delta) \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\tau} .$$

Part (a) is now obtained by substituting $t = 1 - \delta$ in (35). $\square$

Proof of Proposition 1. Let r denote the difference $\ell - m$ and let

$$\tau \triangleq \ell - \ell \cdot \mathbf{a} \cdot \mathbf{z} + \mathbf{a} \cdot \mathbf{p} . \tag{36}$$

Note that $\mathbf{a} \cdot \mathbf{z} \geq 1$ implies $\tau \leq \mathbf{a} \cdot \mathbf{p}$ and that $\mathbf{a} \cdot \mathbf{p} \leq \ell \cdot \mathbf{a} \cdot \mathbf{z}$ implies $\tau \leq \ell$. Also, let $\mathbf{Y}$ be the random variable as in (7). Then,

$$\begin{aligned}\text{Prob}\left\{\mathbf{a} \cdot \mathbf{X} < m\right\} &= \text{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} + \mathbf{a} \cdot \mathbf{s} < \ell - r\right\} \\ &= \text{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} < \ell - \ell \cdot \mathbf{a} \cdot \mathbf{z} + \mathbf{a} \cdot \mathbf{p} - r\right\} \\ &= \text{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} < \tau - r\right\} ,\end{aligned}$$

which readily proves the proposition for $r \geq \tau$. Hence, we assume from now on that $0 \leq r < \tau$.

Apply Lemma 11(a) with τ as in (36) and with $\delta = r/\tau$ (note that, indeed, $\tau \leq \mathbf{a} \cdot \mathbf{p}$). Defining $\sigma \triangleq \tau - r\,(> 0)$, we thus obtain,

$$\text{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} < \tau - r\right\} \leq \left(\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}}\right)^{\tau} = \left(1 + \frac{r}{\sigma}\right)^{\sigma} e^{-r} .$$

Therefore, to have $\text{Prob}\left\{\mathbf{a} \cdot \mathbf{X} < m\right\} \leq \frac{1}{\beta}$ it suffices to require that

$$r - \sigma \cdot \log_e \left(1 + \frac{r}{\sigma}\right) \geq \log_e \beta . \tag{37}$$

Case 1: $r \le 2\sigma$. It is easy to verify that $\log_e(1+t) \le t - (t^2/6)$ whenever $0 \le t \le 2$. Hence, Inequality (37) is implied by

$$r \;-\; \sigma \cdot \left(\frac{r}{\sigma} \;-\; \frac{r^2}{6\sigma^2} \right) \;\ge\; \log_e \beta$$

which, in turn, is satisfied if $r \;\ge\; \sqrt{6\,\sigma \log_e \beta}$. Recalling that $\sigma = \tau - r \le \ell - r = m$, Inequality (37) is thus implied by

$$r \;\ge\; \sqrt{6\,m \log_e \beta}\;. \tag{38}$$

Case 2: $r > 2\sigma$. In this range,

$$\sigma \cdot \log_e \left(1 + \frac{r}{\sigma}\right) \;=\; r \cdot \log_e \left(\left(1 + \frac{r}{\sigma}\right)^{(\sigma/r)} \right) \;\le\; r \cdot \log_e \sqrt{3}\;.$$

Hence, Inequality (37) is satisfied if

$$r \;\ge\; \frac{\log_e \beta}{1 - \log_e \sqrt{3}}\;. \tag{39}$$

The existence of the constant c_1 is now implied by (38) and (39) (setting $c_1 = 2.5$ will do).$\Box$

Proof of Proposition 2. Let $\mathbf{Y}$ be the random variable as in (7) and let r be a positive number. Then,

$$\mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{X} \;>\; E\Big(\mathbf{a} \cdot \mathbf{X}\Big) \;+\; r\right\} \;=\; \mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} \;>\; \mathbf{a} \cdot \mathbf{p} \;+\; r\right\}.$$

Now, the proposition holds trivially when $\mathbf{a} \cdot \mathbf{p} = 0$, since, in this case, $\mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} = 0\right\} = 1$. Therefore, we assume from now on that $\mathbf{a} \cdot \mathbf{p} > 0$.

Apply Lemma 11(b) with $\tau = \mathbf{a} \cdot \mathbf{p}$ and $\delta = r/(\mathbf{a} \cdot \mathbf{p})$; we obtain,

$$\mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{Y} \;>\; \mathbf{a} \cdot \mathbf{p} \;+\; r\right\} \;\le\; \Big((1+\delta)^{-(1+\delta)} e^{\delta}\Big)^{\tau}\;.$$

Therefore, to have $\mathrm{Prob}\left\{\mathbf{a} \cdot \mathbf{X} > E\Big(\mathbf{a} \cdot \mathbf{X}\Big) + r\right\} \le \frac{1}{\beta}$ it suffices to require that

$$\tau \cdot \Big((1+\delta)\log_e(1+\delta) \;-\; \delta\Big) \;\ge\; \log_e \beta\;. \tag{40}$$

Case 1: $r/(\mathbf{a} \cdot \mathbf{p}) = \delta \le \frac{3}{2}$. Noting that $(1+t)\log_e(1+t) \ge t + (t^2/4)$ for $0 \le t \le \frac{3}{2}$, Inequality (40) is satisfied whenever

$$\frac{\delta^2 \tau}{4} \;=\; \frac{r^2}{4\,(\mathbf{a} \cdot \mathbf{p})} \;\ge\; \log_e \beta$$

which, with $E\big(\mathbf{a} \cdot \mathbf{X}\big) \geq \mathbf{a} \cdot \mathbf{p}$, is implied by

$$r \;\geq\; 2\sqrt{E\big(\mathbf{a} \cdot \mathbf{X}\big) \cdot \log_e \beta}\,. \tag{41}$$

Case 2: $r/(\mathbf{a}\cdot\mathbf{p}) = \delta > \frac{3}{2}$. Noting that $t \mapsto (1+t^{-1})\log_e(1+t)$ is monotonously increasing for $t > 0$, we have

$$\begin{aligned} r \cdot \Big((1+\delta)\log_e(1+\delta) \;-\; \delta\Big) \quad &= \quad r \cdot \Big((1+\delta^{-1})\log_e(1+\delta) \;-\; 1\Big) \\ &\overset{\delta \geq \frac{3}{2}}{\geq} \quad r \cdot \Big(\Big(\tfrac{5}{3}\,\log_e \tfrac{5}{2}\Big) \;-\; 1\Big) \;>\; \tfrac{1}{2}\,r\,; \end{aligned}$$

i.e., Inequality (40) is satisfied if

$$r \;\geq\; 2\,\log_e \beta\,. \tag{42}$$

The existence of the constant c_2 (such as $c_2 = 2$) is now implied by (41) and (42). $\square$

References

[1] N. Alon, *A parallel algorithmic version of the Local Lemma, Random Struct. Alg.,* 2 (1991), 367–378.

[2] J. Beck, *An algorithmic approach to the Lovász Local Lemma, I, Random Struct. Alg.,* 2 (1991), 343–365.

[3] E.R. Berlekamp, *Algebraic Coding Theory,* McGraw-Hill, New York (1968).

[4] L.W. Dowdy, D.V. Foster, *Comparative models of the file assignment problem, Comp. Surveys,* 14 (1982), 287–313.

[5] P. Erdös, L. Lovász, *Problems and results on 3-chromatic hypergraphs and some related questions,* in: *Infinite and Finite Sets* (A. Hajnal et al., Editors), *Colloq. Math. Soc. J. Bolyai,* 11, North Holland, Amsterdam (1975), 609–627.

[6] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman, San Francisco (1979).

[7] D.S. Hochbaum, *On the fractional solution to the set covering problem, SIAM J. Alg. Disc. Meth.,* 4 (1983), 221–222.

[8] G. Kant, J. van Leeuwen, *File distribution problem for processor networks, Proc. Scandinavian Workshop on Algorithmic Theory* (1990), 47–59.

[9] N. Karmarkar, *A new polynomial-time algorithm for linear programming, Combinatorica,* 4 (1984), 373–395.

[10] E.D. Karnin, J.W. Greene, M.H. Hellman, *On secret sharing systems, IEEE Trans. Inform. Theory,* 29 (1983), 35–41.

[11] L. Lovász, *On the ratio of optimal integral and fractional covers, Discrete Math.,* 13 (1975), 383–390.

[12] F.J. MacWilliams, N.J.A. Sloane, *The Theory of Error-Correcting Codes,* North Holland, Amsterdam (1977).

[13] S. Mahmoud, J.S. Riordan, *Optimal allocation of resources in distributed information networks, ACM Trans. Database Sys.,* 1 (1976), 66–78.

[14] M.O. Rabin, *Efficient dispersal of information for security, load balancing, and fault tolerance, J. ACM,* 36 (1989), 335–348.

[15] P. Raghavan, *Probabilistic construction of deterministic algorithms: approximating packing integer programs, J. Comp. Sys. Sciences,* 37 (1988), 130–143 (see also *Proc. 27th IEEE Symp. Found. Comp. Science* (1986), 10–18).

[16] P. Raghavan, Lecture notes on randomized algorithms, Technical Report RC 15340 (#68237), IBM T.J.Watson Research Center, 1990.

[17] P. Raghavan, C.D. Thompson, *Randomized rounding: a technique for provably good algorithms and algorithmic proofs, Combinatorica,* 7 (1987), 365–374.

[18] A. Shamir, *How to share a secret, Comm. ACM,* 22 (1979), 612–613.

[19] J. Spencer, *Ten Lectures on the Probabilistic Method,* SIAM, Philadelphia (1987).