Optimal File Sharing in Distributed Networks

(Preliminary Version)

MONI NAOR* RON M. ROTH*†

Abstract

Given a distributed network of processors represented by an undirected graph $G = (V, E)$, and a file size k, we consider the following problem: An arbitrary file $\mathbf{w}$ of k bits is to be distributed among all nodes of the network 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 $\mathbf{w}$. 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 size of memory 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 size of the memory at each node in G, as well as the mapping of any file $\mathbf{w}$ into these 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 for k being much larger than $\log \Delta_G$ is necessary in order to have total memory size close to the integer programming lower bound.

1 Introduction

We consider the following file distribution problem: A distributed 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 G, 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

*IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120. e-mail: naor@almaden.ibm.com

†On leave from the Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel. e-mail: ronny@cs.technion.ac.il

CH3062-7/91/0000/0515$01.00 © 1991 IEEE

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 pairs of networks and file sizes $\left\{(G_l,k_l)\right\}_{l=0}^{\infty}$ 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 $1+\varepsilon(G_l,k_l)$ times larger than the integer (or linear) lower bound, with $\liminf_{l\to\infty}\varepsilon(G_l,k_l) \geq \frac{1}{4}$. 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 which 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 in G which are adjacent to u. The degree of u is denoted by $\Delta(u) \triangleq |\Gamma(u)|$, and the maximum degree in G is denoted by $\Delta_G \triangleq \max_{u\in V}\Delta(u)$.

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 $\mathbf{y}\cdot\mathbf{1}$, 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 $F_2 \triangleq GF(2)$, 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 u-th 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*

$$\mathcal{E}_u : F_2^k \to F_2^{x_u} \quad \text{for every } u\in V\,;$$

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*

$$\mathcal{D}_u : F_2^{(A_G\mathbf{x})_u} \to F_2^k \quad \text{for every } u\in V\,.$$

The memory allocation, encoding mappings and decoding mappings satisfy the requirement

$$\mathcal{D}_u\left([\mathcal{E}_v(\mathbf{w})]_{v\in\Gamma(u)}\right) \equiv \mathbf{w}\,, \quad \mathbf{w}\in F_2^k\,. \qquad (1)$$

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 the nodes v which are adjacent to u.

[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 $[z_v]_{v\in\Gamma(u)}$.

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

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

where $0 < j(u;1) < j(u;2) < \ldots < j(u;x_u)$. 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 computational complexity of 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 complexity of computation we mean the running time of a Turing machine which performs the computation, plus the size of some universal coding of that Turing machine.

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 number of files (each, say, of the same size k) which 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 is, in the nutshell, 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 it, 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.

Denote by $\langle n\rangle$ the set $\{1,2,\ldots,n\}$. Let C be an (n,K) code over Φ and let S be a subset of $\langle n\rangle$. 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 $(G, k) \mapsto \chi(G, k)$ which 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 whenever $k \gg \Delta_G^2 \log \Delta_G$.

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

$$\begin{aligned} & I(G,k) = \min \|\mathbf{y}\|, \\ \mathrm{IP}(G,k): \quad & \text{ranging over all integer } \mathbf{y} \text{ such that} \\ & A_G \mathbf{y} \geq k \cdot \mathbf{1} \quad \text{and} \quad \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{2}$$

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

$$\begin{aligned} & \rho_G = \min \|\mathbf{z}\| , \\ \mathrm{LP}(G): \quad & \text{ranging over all rational } \mathbf{z} \text{ such that} \\ & A_G \mathbf{z} \geq \mathbf{1} \quad \text{and} \quad \mathbf{z} \geq \mathbf{0} . \end{aligned} \tag{3}$$

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

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

$$\rho_G \cdot k \leq I(G,k) \leq M(G,k) \leq k \cdot I(G,1) \leq k \cdot |V| . \tag{4}$$

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

Substituting $k = 1$ in (4) we obtain the equality $M(G, 1) = I(G, 1)$. 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 (this argument applies also to the case where the encoding and decoding mappings are computable in polynomial-time).

Remark 3. A result of Lovász [11] states that $I(G, 1) \leq \rho_G \log_2 \Delta_G$; on the other hand, one can construct an infinite family of network graphs $\{G_l\}_l$ (such as the one presented in Section 5) for which $I(G_l, 1) \geq \frac{1}{4} \rho_{G_l} \log_2 \Delta_{G_l}$ (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, $k \cdot I(G, 1)$, 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 might be unlikely to find 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*

$$\frac{|\chi(G,k)|}{\rho_G \cdot k} = 1 + O\left(\max\left\{\frac{\log \Delta_G}{k}\,;\,\sqrt{\frac{\log \Delta_G}{k}}\right\}\right). \quad (5)$$

(The maximum in the right-hand side of (5) is determined according to whether k is smaller, or larger, than $\log \Delta_G$. Also, by (4), Equation (5) implies that the ratios $|\chi(G,k)|/M(G,k)$, $M(G,k)/I(G,k)$, and $I(G,k)/(\rho_G \cdot k)$ 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 (5); 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 (5).

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

$$\frac{|\chi(G,k)|}{\rho_G \cdot k} = 1 + O\left(\frac{\Delta_G \cdot \log(\Delta_G \cdot k)}{k}\right). \quad (6)$$

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) (as in Step 1 above). Let $\mathbf{z} = [z_u]_{u \in V}$ be an optimal solution to the linear programming problem LP(G) in (3). 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

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

Clearly, $\|\mathbf{y}\| \le \rho_G \cdot (l + \Delta_G)$, and, since $A_G\mathbf{z} \ge \mathbf{1}$, we also have $A_G\mathbf{y} \ge l \cdot \mathbf{1}$. 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 the nodes $u \in V$ in some arbitrary order, and at each node u update 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 to be updated. 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

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

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 $|\Phi| \geq \Delta_G \cdot k \geq \Delta_G \cdot l$). Given a file $\mathbf{w}$ of k bits, we regard $\mathbf{w}$ as a polynomial $w(t)$ of degree $< \lceil k/h \rceil = l$ over Φ and, as such, we compute the values $w_j = w(\alpha_j)$, $1 \leq j \leq \Delta_G \cdot l$. At each node $u \in V$ we now store the values w_j, $j \in C_u$, requiring memory allocation of $x_u = h \cdot y_u$ at that node. Since each u has access to values w_j of $w(t)$ 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; such a characterization will be used also in Section 4 to prove our main result (recall the definitions at the end of Section 2). Let B_{RS} be an $l \times (\Delta_G l)$ matrix over $\Phi = GF(2^h)$ defined by $(B_{\mathrm{RS}})_{i,j} = \alpha_j^{i-1}$, $1 \leq i \leq l$, $1 \leq j \leq \Delta_G l$. 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 $\langle \Delta_G l \rangle$, 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 [17] (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 Subsection 4.1 we show how to find a memory allocation by scaling and perturbing a solution to the linear programming problem LP(G) defined in (3) (resulting in an integer vector $\mathbf{x}$ whose norm $\|\mathbf{x}\|$ is much closer to $\rho_G \cdot k$ than the one obtained by the simple construction of Subsection 3.3). Having found a memory allocation $\mathbf{x}$, we then describe the encoding and decoding mappings In Subsection 4.2.

We make use of the following known lemma (hereafter e stands for the base of natural logarithms).

Lemma 3. (Lovász Local Lemma [5][18]). *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* $\mathrm{Prob}\{\mathcal{A}_i\} \leq p$ *for all* $1 \leq i \leq n$. *If* $ep\delta < 1$, *then* $\mathrm{Prob}\left\{\bigwedge_{i=1}^{n} \overline{\mathcal{A}_i}\right\} > 0$.

In most applications of the lemma (as well as 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.

4.1 Step 1. Solving for a memory allocation

The goal of this subsection is to prove the following.

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 an integer vector $\mathbf{x}$ satisfying $A_G \mathbf{x} \geq m \cdot \mathbf{1}$ such that*

$$\frac{\|\mathbf{x}\|}{\|m \cdot \mathbf{z}\|} \leq 1 + c \cdot \max\left\{\frac{\log_e \Delta_G}{m} ; \sqrt{\frac{\log_e \Delta_G}{m}}\right\} \quad (7)$$

for some universal constant c.

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

Theorem 3 is proven via a "randomized rounding" argument (see [15][16][18]): solve first the corresponding linear programming problem LP(G) in (3) (say, by Karmarkar's algorithm [9]), and use the rational solution to define a probability measure on integer vectors which are candidates for $\mathbf{x}$. This probability space is then shown to contain an integer vector $\mathbf{x}$ which satisfies the conditions of Theorem 3. Furthermore, we can derive a polynomial-time algorithm for finding such an $\mathbf{x}$. Note that if we are interested in a weaker result, where $\log|V|$ replaces $\log \Delta_G$ in (5) (or (7)), 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 Lovász Local Lemma (Lemma 3); another option is to use the "method of alternation" (see [18]) 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. Both methods can be used to prove Theorem 3; furthermore, and both of them can be constructive and deterministic: the Local Lemma by using Beck's method [2] and the method

of alternation by applying Raghavan's pessimistic estimators. We show here the method of alternation.

Given a nonnegative real vector $\mathbf{z} = [z_u]_{u \in V}$ such that $A_G \mathbf{z} \geq \mathbf{1}$ 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

$$s_u \triangleq \lfloor \ell \cdot z_u \rfloor \quad \text{and} \quad p_u \triangleq \ell \cdot z_u - s_u \; ; \quad u \in V \; ; \tag{8}$$

note that $0 \leq p_u < 1$ 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

$$\text{Prob}\Big\{Y_u = 1\Big\} = p_u \, , \quad u \in V \, . \tag{9}$$

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

$$\mathbf{X} \triangleq \mathbf{s} + \mathbf{Y} \, . \tag{10}$$

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

$$E\Big(\mathbf{a} \cdot \mathbf{X}\Big) = \mathbf{a} \cdot \mathbf{s} + \mathbf{a} \cdot \mathbf{p} = \ell \cdot \mathbf{a} \cdot \mathbf{z} \geq \ell \, . \tag{11}$$

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

$$E\Big(A_G \mathbf{X}\Big) \geq \ell \cdot \mathbf{1} \, ,$$

making $\mathbf{X}$ a candidate in this case for the desired solution $\mathbf{x}$, provided that ℓ is chosen properly. Showing the existence of such an $\mathbf{x}$ makes use of the following two propositions, which are similar to those stated in [15]. The proofs of these two propositions will be given in the final version.

For $\beta \geq 1$ and $\eta \geq 0$, let $L\{\beta, \eta\}$ be defined by

$$L\{\beta, \eta\} \triangleq \max\Big\{\log_e \beta \, ; \, \sqrt{\eta \cdot \log_e \beta}\Big\} \, .$$

Proposition 1. *Let $\mathbf{X} = [X_u]_{u \in V}$ be defined by (8)–(10), let $\mathbf{a}$ be a real vector in $[0, 1]^{|V|}$ such that $\mathbf{a} \cdot \mathbf{p} \geq 1$, and let m be a positive integer. There exists a constant c_0 such that, for every $\beta \geq 1$,*

$$\text{Prob}\Big\{\mathbf{a} \cdot \mathbf{X} < m\Big\} \leq \frac{1}{\beta} \tag{12}$$

whenever $\ell \geq m + c_0 \cdot L\{\beta, m\}$.

Proposition 2. *Let $\mathbf{X} = [X_u]_{u \in V}$ be defined by (8)–(10) and let $\mathbf{a}$ be a real vector in $[0, 1]^{|V|}$ such that $\mathbf{a} \cdot \mathbf{p} \geq 1$. There exists a constant c_0 such that, for every $\beta \geq 1$,*

$$\text{Prob}\Big\{\mathbf{a} \cdot \mathbf{X} > E\Big(\mathbf{a} \cdot \mathbf{X}\Big) + \tau\Big\} \leq \frac{1}{\beta} \tag{13}$$

whenever $\tau \geq c_0 \cdot L\Big\{\beta, E\Big(\mathbf{a} \cdot \mathbf{X}\Big)\Big\}$.

Given network graph G and an integer k, we assume from now on that $\mathbf{z}$ is a nonnegative rational vector satisfying $A_G \mathbf{z} \geq \mathbf{1}$, and we set $m = k + 2\lceil \log_2 \Delta_G \rceil + 1$, $\beta = \beta_G \triangleq 2\Delta_G^2$, $\tau = c_0 \cdot L\{\beta_G, \ell \|\mathbf{z}\|\}$ and $\ell = m + c_0 \cdot L\{\beta_G, m\}$. By Proposition 1 we have

$$\text{Prob}\Big\{(A_G \mathbf{X})_u < m\Big\} \leq \frac{1}{\beta_G} \quad \text{for every } u \in V \tag{14}$$

Inequality (14) bounds from above the probability of the 'bad' event of not allocating enough memory to the nodes in $\Gamma(u)$.

By Proposition 2 we have that

$$\text{Prob}\Big\{\|\mathbf{X}\| > E(\|\mathbf{X}\|) + \tau\Big\} \leq \frac{1}{\beta_G} \tag{15}$$

Consider the following procedure for finding a good memory allocation: Generate $\mathbf{X}$ as defined by (10). For every node u such that $(A_G \mathbf{X})_u < m$, round up the allocation at all its neighbors (including itself). In other words, if v is a neighbor of a deficient node, then no matter what Y_v is, allocate $s_v + 1$ for v. Thus, the allocation is increased by at most 1 compared with X_v. By assumption on $\mathbf{z}$, it is clear that the sum of the allocations around u following the rounding is at least $\ell \geq m$. The probability that the allocation at any given node is increased is therefore bounded by the probability that any of its neighbors was not covered properly, which in turn is at most $\Delta_G / \beta_G = \frac{1}{2\Delta_G}$. Hence the expected number of nodes whose allocation is increased is at most $\frac{|V|}{2\Delta_G}$ and with probability at least $\frac{1}{2}$ there are no more than $\frac{|V|}{\Delta_G}$ nodes that need an increase in their allocation. Since $\frac{|V|}{\Delta_G} \leq \|\mathbf{z}\|$, the total increase by the rounding is at most $\|\mathbf{z}\|$.

On the other hand, with probability at least $1 - 1/\beta_G$ the total size of the (original) allocation, i.e. $\|\mathbf{X}\|$, does not exceed $E(\|\mathbf{X}\|) + \tau = \ell \|\mathbf{z}\| + \tau$.

Therefore, we can conclude that there must be a nonnegative integer vector $\mathbf{x}$ satisfying

$$(A_G \mathbf{x})_u \geq m \tag{16}$$

and

$$\|\mathbf{x}\| \leq (\ell + 1) \cdot \|\mathbf{z}\| + c_0 \cdot L\Big\{\beta_G, \ell \cdot \|\mathbf{z}\|\Big\} \tag{17}$$

for all $u \in V$.

Hence, we have

$$\frac{\|\mathbf{x}\|}{\|\ell \cdot \mathbf{z}\|} \leq 1 + \frac{1}{\ell} + c_0 \left(\frac{\log_e \beta_G}{\ell} + \sqrt{\frac{\log_e \beta_G}{\ell}}\right) \tag{18}$$

Inequality (7) now easily follows by substituting

$$\ell = m \cdot \left(1 + c_0 \cdot \max\left\{\frac{\log_e \beta_G}{m} \, ; \, \sqrt{\frac{\log_e \beta_G}{m}}\right\}\right)$$

and $\beta_G = 2\Delta_G^2$ in (18). This concludes the proof of Theorem 3.

Observe that, since $m = k + 2\lceil \log_2 \Delta_G \rceil + 1$, we also have

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

(compare with the right-hand side of (5)).

The vector $\mathbf{x}$ will serve as the memory allocation of $\chi(G,k)$. As mentioned above, Raghavan's method of pessimistic estimators can be applied to obtain a deterministic Polynomial time algorithm for finding $\mathbf{x}$.

4.2 Step 2. Defining the encoding mappings

The construction of the encoding and decoding mappings makes use of the following lemma.

Lemma 4. *Let $S_1, S_2, \ldots S_t$ be subsets of $\langle n \rangle$, each S_i of size $s_i \geq s$, and no subset intersects more than δ subsets. Let q be a power of a prime and let k be a nonnegative integer satisfying*

$$e \cdot \delta \cdot q^{-s-1} < q^{-k}\,. \tag{19}$$

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

Proof. We construct inductively $l \times n$ matrices B_l, $1 \leq l \leq k$, each generating a linear code which is separable with respect to every S_i; that is, each $(B_l)_{S_i}$ 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 $l \leq k$. We are now to append an l-th row to B_{l-1}.

Given such a matrix B_{l-1}, a row vector in Φ^n is 'good' with respect to S_i if, when appended to B_{l-1}, yields a matrix B_l such that $(B_l)_{S_i}$ has rank l; otherwise, a row vector is 'bad' with respect to that S_i. Now, for each i, the row span of $(B_{l-1})_{S_i}$ consists of q^{l-1} vectors in Φ^{s_i}; this means that the probability of a randomly selected row to be bad with respect to S_i is $q^{-s_i+l-1} \leq q^{-s-1+k} < 1/(e \cdot \delta)$. Observe that the events "the row vector is bad with respect to S_i" and "the row vector is bad with respect to S_j" are independent if $S_i \bigcap S_j = \emptyset$; thus, by Lemma 3 we are guaranteed to have a row vector in Φ^n which is good with respect to every S_i. This vector can now be appended to B_{l-1} to obtain a generator matrix B_l with $(B_l)_{S_i}$ having rank l for all i. □

Up till recently there was no known way of applying the Local Lemma that lead to a polynomial-time algorithm which actually finds the desired object. However, Beck [2] has found a constructive method that can be used in most applications of the lemma, including ours (see also [1]). Thus, as in Subsection 4.1, we can obtain such a code constructively.

We turn now to defining the encoding and decoding mappings for a given instance (G,k). Let $\mathbf{x}$ be the nonnegative integer vector obtained by the algorithm of Subsection 4.1 and which satisfies both (16) and (17). Partition the set $\langle \|\mathbf{x}\| \rangle$ into $|V|$ (disjoint) subsets Q_u with $|Q_u| = x_u$ and let $S_u \triangleq \cup_{v \in \Gamma(u)} Q_v$, $u \in V$. By (16) we have $s_u \triangleq |S_u| = (A_G\mathbf{x})_u \geq m = k + 2\lceil \log_2 \Delta_G \rceil + 1$. This, in turn, implies the inequality $e \cdot \Delta_G^2 \cdot 2^{-s_u - 1} < 2^{-k}$. Furthermore, each S_u intersects at most $(\Delta_G - 1)^2 + 1$ sets S_v; hence, by Lemma 4 there exists a linear $(\|\mathbf{x}\|, 2^k)$ code over F_2 which is separable with respect to each S_u. Note that there is also a polynomial time algorithm for finding a generator matrix, namely, B, of such a code. For each $u \in V$ let $B_u \triangleq (B)_{Q_u}$; i.e., B_u is the $k \times x_u$ matrix consisting of all columns of B indexed by Q_u. The encoding mappings $\mathcal{E}_u : F_2^k \to F_2^{x_u}$ are now defined by $\mathcal{E}_u : \mathbf{w} \mapsto \mathbf{w}B_u$, $u \in V$, and the overall process of encoding $\mathbf{w}$ into $[\mathcal{E}_u(\mathbf{w})]_{u \in V} = \mathbf{w}B$ requires $O(k \cdot \|\mathbf{x}\|)$ multiplications and additions over F_2.

As for the decoding mappings $\mathcal{D}_u : F_2^{(A_G\mathbf{x})_u} \to F_2^k$, note that for each $u \in V$, the $k \times (A_G\mathbf{x})_u$ matrix $(B)_{S_u} = [B_v]_{v \in \Gamma(u)}$ is of rank 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 $[\mathbf{w}B_v]_{v \in \Gamma(u)}$, namely, k coordinates which correspond to k linearly independent columns of $[B_v]_{v \in \Gamma(u)}$. Let such a set of coordinates be indexed by the set $T_u \subseteq S_u$, $u \in V$. Assuming a 'hard-wired' connection between node u and the k entries of $[\mathbf{w}B_v]_{v \in \Gamma(u)}$ indexed by T_u (all of which reside in nodes adjacent to u), the decoding process at u sums up to multiplying the vector $([\mathbf{w}B_v]_{v \in \Gamma(u)})_{T_u} \in F_2^k$ by the inverse matrix of $([B_v]_{v \in \Gamma(u)})_{T_u} = (B)_{T_u}$. In other words, the mappings $\mathcal{D}_u$, $u \in V$, are given by $\mathcal{D}_u(\mathbf{c}) = (\mathbf{c})_{T_u}((B)_{T_u})^{-1}$ for every $\mathbf{c} \in F_2^{(A_G\mathbf{x})_u}$. The decoding process at each node thus requires $O(k^2)$ multiplications and additions over F_2.

Remark 4. It is worthwhile comparing the file distribution scheme described in the previous subsections with a scheme using Reed-Solomon codes instead of the code obtained by Lemma 4 (the way we did in Subsection 3.3) while applying the same method of Subsection 4.1 to solve for the memory allocation. It can be verified that the resulting file distribution scheme is slightly worse than that of Subsection 4.2 — one should replace each appearance of the term $\log \Delta_G$ in the right-hand side of (5) by $\log(\Delta_G \cdot k) \log \Delta_G$. In par-

ticular, this method has critical file size of $\log^2 \Delta_G$. •

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 $k_l \geq \log_2 \Delta_{G_l}$, $l = 1,2,\ldots$, for which the ratios $M(G_l,k_l)/I(G_l,k_l)$ (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 $\Theta_{m,l}$ consist of all subsets of U_m of size l. Set $V_{m,l} = U_m \cup \Theta_{m,l}$ 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) $u \in U_m$, $v \in \Theta_{m,l}$, and $u \in v$; (iii) $u = v$ (self loops).

First, we verify that $\rho_{G_{m,l}} = m/l$. Consider a nonnegative real vector $\mathbf{z} = [z_u]_{u \in V_{m,l}}$ such that $\|\mathbf{z}\| = \rho_{G_{m,l}}$ and $A_{G_{m,l}}\mathbf{z} \geq \mathbf{1}$. Without loss of generality, we can assume that $z_v = 0$ for every $v \in \Theta_{m,l}$; 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 $z_1 \leq z_2 \leq \ldots \leq z_m$. Noting that

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

for every node $u \geq l$ in U_m we thus have $z_u \geq z_l \geq 1/l$. Hence,

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

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

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

we arrive at the equality $\rho_{G_{m,l}} = m/l$ (compare with [7]). This also implies that $I(G_{m,l}, r \cdot l) = r \cdot m$ for every positive integer r.

In the forthcoming discussion we will be concentrating on certain 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

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

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

$$\lim_{l \to \infty} \frac{M(G_l, r \cdot l)}{I(G_l, r \cdot l)} \geq 1 + \frac{1}{2r}. \tag{20}$$

Remark 5. Note that Proposition 3 compares the minimum memory size $M(G_l, r \cdot l)$ with the *integer* programming bound $I(G_l, r \cdot l)$ (in fact, we have $I(G_l, r \cdot l) = \rho_{G_l} \cdot r \cdot l$). In particular, for $k_l \triangleq 2l \geq \log_2 \Delta_{G_l}$ Proposition 3 becomes

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

This bound exhibits the fact that a file size of $\log \Delta_{G_l}$ is a critical point in the sense that, 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 $I(G_l, k_l)$ and $\rho_{G_l} \cdot k_l$, but rather because of a gap between $M(G_l, k_l)$ and $I(G_l, k_l)$. •

Proof of Proposition 3. 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 $v \in \Theta_{2l,l}$ and that the nodes of $U_{2l} = \langle 2l \rangle$ are renamed to have $x_1 \leq x_2 \leq \ldots \leq x_{2l}$. Letting $h \triangleq x_{l+2}$, we obtain

$$\begin{aligned} M(G_l,k) &= \|\mathbf{x}\| \\ &= \sum_{u=1}^{l} x_u + x_{l+1} + \sum_{u=l+2}^{2l} x_u \\ &\geq r \cdot l + r + (l-1)h \end{aligned} \tag{21}$$

where the inequality in (21) follows from the fact that $\sum_{u=1}^{l} x_u = (A_{G_l}\mathbf{x})_{\langle l \rangle} \geq k = rl$ which, in turn, implies that $x_{l+1} \geq x_l \geq r$.

For a file $\mathbf{w} \in F_2^k$, let $\mathbf{c_w}$ 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

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

as an $(l+2, 2^k)$ code over an alphabet of $q \triangleq 2^h$ elements. Note that C must be separable with respect

to any subset of $\langle l+2\rangle$ of size l to allow the nodes in $\Theta_{2l,l}$ to reconstruct the file $\mathbf{w}$. Hence, by Lemma 1, the minimum distance of C must be at least 3, which, by the well-known sphere-packing bound [12, Ch. 1], implies the following inequality on the parameters of C:

$$2^k \cdot \Big(1 + (l+2)(q-1)\Big) \leq q^{l+2} . \qquad (22)$$

Substituting $k = rl$ and $q = 2^h$ in (22), and noting that $2^h - 1 \geq 2^{h-1}$, we arrive at

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

or

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

Hence, for fixed r and for sufficiently large l we must have $h \geq r+1$. By (21) we thus obtain

$$\lim_{l\to\infty} \frac{M(G_l, r\cdot l)}{I(G_l, r\cdot l)} \geq 1 + \frac{1}{2r} . \qquad \square$$

Remark 6. It can be readily verified that $I(G_{m,l}, k) \geq m - l + k$ for every m, l, and k, and, in particular, $I(G_l, 1) \geq l+1 \geq \frac{1}{4}\rho_{G_l}\log_2\Delta_{G_l}$. 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 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 $I(G,k)$ is even larger than stated in Proposition 3. We demonstrate this for the network graphs $H_l = G_{2^l,l}$ in the following proposition.

Proposition 4.

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

Here $f_l(k) \sim g_l(k)$ stands for a shorthand notation for $\lim_{l\to\infty} f_l(k)/g_l(k) = 1$ uniformly with respect to k. Note that when $k = l$, the ratio $M(H_l,k)/I(H_l,k)$ is approximately l which, in turn, is at least $\sqrt{\log_2\Delta_{H_l}}$.

The proof of Proposition 4 makes use of the following known lemma.

Lemma 5. (The Plotkin bound [3, p. 315]). *Let C be an (n,K) code of minimum distance d over an alphabet of q elements. Then,*

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

Proof of Proposition 4. We distinguish between the several ranges of k.

Case 1: $k = rl$ for some integer $r \geq l$. Note that in this case we have $k \geq l^2 \geq \log_2\Delta_{H_l}$ and $I(H_l,k) = \rho_{H_l}\cdot k = r\cdot 2^l$. In fact, we also have $M(H_l,k) = I(H_l,k)$: 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 Subsection 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 $r < l$. As in case 1, we have $I(H_l,k) = r\cdot 2^l$ also for this range of k. Turning to $M(H_l,k)$, 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 $v \in \Theta_{2^l,l}$ 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

$$M(H_l,k) = \|\mathbf{x}\| > h\cdot 2^l \cdot \left(1 - \frac{1}{l} - \frac{1}{2^l}\right) \qquad (23)$$

(compare with (21)).

To bound h from below, we regard the set

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

as an $(n, 2^k)$ code over an alphabet of $q \triangleq 2^h$ elements. Since C has to be separable with respect to any subset of $\langle n\rangle$ of size l, C must have minimum distance $\geq n - l + 1$ (Lemma 1); this, in turn, implies by Lemma 5 the inequality

$$\frac{1}{2^h} \leq 1 - \left(1 - \frac{l-1}{n}\right)\left(1 - \frac{1}{2^k}\right) \leq \frac{l}{n} \leq \frac{l^2}{2^l} , \qquad (24)$$

i.e.,

$$h \geq l - 2\log_2 l . \qquad (25)$$

Plugging (25) into (23) we thus obtain

$$M(H_l,k) > l\cdot 2^l \cdot \Big(1 - o(1)\Big) , \qquad (26)$$

where $o(1)$ stands for an expression, independent of k, which tends to zero as l goes to infinity. The bound (26) implies, in turn, the inequality

$$\frac{M(H_l,k)}{I(H_l,k)} \geq \frac{l}{r}\cdot\Big(1 - o(1)\Big) = \frac{l^2}{k}\cdot\Big(1 - o(1)\Big) . \qquad (27)$$

We note that, up to a multiplying factor $1 - o(1)$, the bounds (26) and (27) are definitive: an upper bound

$M(H_l,k) \le l \cdot 2^l$ is obtained by assigning the coordinates of a $(2^l, 2^{r\,l})$ 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: $k < l$. In this range of k we have $I(H_l,k) \le 2^l - l + k$ (and, by Remark 6 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

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

satisfies the inequality $A_{H_l}\mathbf{y} \ge k \cdot \mathbf{1}$.

As for $M(H_l,k)$, we first replace (24) by

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

i.e.,

$$h \ge \min\left\{ l - \lceil 2\log_2 l \rceil ; k \right\} \ge \left(1 - \frac{\lceil 2\log_2 l \rceil}{l} \right) \cdot k . \tag{28}$$

Combining (23) with (28) yields

$$M(H_l,k) \ge k \cdot 2^l \cdot \Big(1 - o(1) \Big) \tag{29}$$

and

$$\frac{M(H_l,k)}{I(H_l,k)} \ge k \cdot \Big(1 - o(1) \Big) . \tag{30}$$

Again, the bounds (29) and (30) 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 $k \cdot 2^l$. □

Acknowledgment

The authors thank Noga Alon and Cynthia Dwork for the many helpful discussions.

References

[1] N. Alon, *A parallel algorithmic version of the Local Lemma, these proceedings.*

[2] J. Beck, *An algorithmic approach to the Lovász Local Lemma*, to appear.

[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. Gary, 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. of Comp. Science* (1986), 10–18).

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

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

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