Coded Caching with Low Per-Request Complexity

Barak Farbman*, **Yuval Cassuto*** and **Alex Sprintson**†
*Viterbi Dept. of Electrical Engineering, Technion – Israel Institute of Technology
†Dept. of Electrical and Computer Engineering, Texas A&M University
emails: barakfarbman@gmail.com , ycassuto@ee.technion.ac.il , spalex@tamu.edu

Abstract— **The massive growth in high-rate streams (mostly video) over the Internet warranted the establishment of content distribution networks (CDNs), which bring the content closer to the consumer and hence reduce delays and congestion in the network. In this paper we address the setup of multiple Internet service providers (ISPs) sharing a federated cache. When multiple ISPs with different demands share a cache, storage cost can be reduced when the cache is coded. We address the problem of constructing the coded cache to fulfill all demands from all ISPs, where the constructions simultaneously optimize the cache's three main operational costs: storage, communication, and computation (to encode and decode requested objects). When only the storage needs to be minimized, the code-design problem can be formulated as an index coding problem, and solved by known tools in network coding. However, in practical setups the cache needs to respond efficiently (in communication and computation) to *individual* content-object requests, which renders known index/network-coding solutions non-applicable and non-scalable. We first show constructively that when the number of ISPs is less than or equal to 4, there is an efficient algorithm that achieves optimal storage with only bit-wise XOR operations, and with guaranteed low computation cost for serving requests. For larger numbers of ISPs, we present randomized algorithms that achieve optimal storage with XOR-only coding and low computation complexity per requested objects.**

I. Introduction

The constant growth of video traffic is burdening the network infrastructure at every network deployment point, and in all layers. A key observation toward mitigating this problem is that a significant portion of video traffic is transmitting the same popular content. This property has motivated the introduction of *caching* and *content-distribution networks* (CDNs), which save network resources by bringing the popular content closer to its likely consumers.

A. Caching and CDNs

In data networking, caching is a means to trade off communication and storage costs while meeting the users' demands for content. The population of the (limited-size) cache with data is done in a way to optimize the performance given the system structure and the data access patterns. There has been a vast body of work optimizing network caching in a variety of setups (see, e.g., [1], [2], [3], [4], [5]). Caching at premises of the *Internet service provider (ISP)* is the most natural and effective choice for operating a network cache. An ISP bridges between the broad Internet and a large body of users in a local proprietary network. The communication costs in the local network are orders of magnitude lower than in the links to the Internet backbone – thus an ISP cache saves by serving repeating popular content locally. The next natural step beyond local ISP caching is a federation of multiple ISPs jointly deploying a *shared cache*. The rationale behind a multi-ISP cache is that the same content objects are watched by customers of different ISPs, so sharing a cache means sharing the cost.

By our setup in this paper the ISPs decide which objects they *demand* from the shared cache, and the cache is designed such that it can fulfill all demands by all ISPs with minimal implementation cost. Our study shows that a *coded shared cache* can have substantially lower storage costs compared to a standard (uncoded) cache, while the extra complexity of coding is controlled by new code-design algorithms.

B. Coded caching

While there is clearly some overlap between the demands of different ISPs, they each have distinct caching needs due to differences in their user demands. In our model for the shared cache (Section II), a set of N content objects are agreed upon by the M ISPs to be handled by the cache ($N \gg M$). Then each ISP is allowed to choose independently which subset of content objects it expects to request from the cache, and stores the rest of the objects locally. The agreed set of content objects reflects a mild level of demand commonality, while the independent subset choices give high flexibility to accommodate ISP differences. The cache needs then to be populated with coded objects such that all ISP demands can be fulfilled upon request, and with low cost of serving each object request. Note that ISPs store only uncoded content objects.

Our contributions. Our results in the paper show how to design the coded cache such that its storage cost meets the fundamental lower bounds achieved by known network-coding algorithms, but with the additional constraint that fulfilling a content-object request will require a small number of bit-wise XOR operations on the coded objects. The rest of the paper is organized as follows. Section II introduces the model, definitions and notations that will be used throughout the paper. Section III shows constructively that for up to $M = 4$ ISPs the storage lower bound can be attained with simple bit-wise XOR coding, and with per-request complexity that is a small constant (independent of N). In contrast, existing results in network coding only guarantee binary coding for up to $M = 2$, and they also have per-request complexity that in general grows with N. The results of Section III take the approach of efficiently and optimally partitioning an arbitrary demand matrix for $M = 3, 4$, while for general M optimal partitioning is a known hard problem. Section

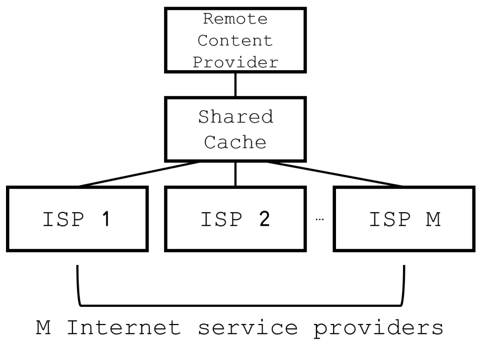

Figure 1: The network model.

IV extends code construction to more than 4 ISPs. For an arbitrary number of ISPs, we propose randomized algorithms that partition the demand matrix to sub-matrices with optimal-storage coding, and save running time by considering the distribution of the demand matrix. The two main contributions of Section IV that drive the proposed algorithms are: 1) analysis of random sequential object-selection processes, and 2) a sufficient+necessary condition for binary coding based on 3-colorability of small graphs. Finally, Section V concludes the paper and suggests directions for future work.

We believe the results of this paper, thanks to our focus on low complexity, bring coded caching closer to practical realization in content distribution networks. The model of a coded shared cache with uncoded ISP storage allows a smoother transition from current ISP caching, and for small numbers of ISPs the proposed code-design algorithms are practical.

II. The Model

A. The network model

Consider a content distribution network model as depicted in Fig. 1. At the top lies a *remote content provider* that stores the content consumed by the network client users. Opposite in the bottom lie M ISPs, each locally serving its clients with the requested content. In addition to local storage at each ISP, the network model adds a caching layer in between the content provider and ISPs. This is a *shared cache* jointly serving all ISPs, whose objective is to offer caching capability to the ISPs beyond what each can afford to store locally.

B. The shared-cache model

The model we assume for the shared cache is now described. In the remainder of this paper we refer to the shared cache simply as *the cache*. In the scope of the cache are N popular content objects agreed by the M ISPs sharing it. Each ISP chooses a subset of the N objects to be *demanded* from the cache, and the remaining subset it makes *available* in its own local storage. The partition of the objects to available and demanded sets is done independently by each ISP, according to the popularity of the objects its clients exhibit. We will mostly assume that the *size* of the demanded set is fixed and equal for all ISPs. For example, an ISP may choose to put its K most popular objects in the available set, and the remaining $N-K$ objects (which are also popular, but less than the first K) in the demanded set. Note that throughout this paper we assume that objects have the same size. We do so for clarity, noting that in a real implementation variable-size objects can be handled by simple techniques like chunking and padding. After the demanded sets of the M ISPs have been specified, the system invokes a *placement phase* for populating the cache with objects. Then in the *request phase* the cache is used to fulfill requests from the ISPs for content in their demanded set they need to deliver upon their clients' requests. Note that the placement phase can be a recurring event and an opportunity to change the agreed upon universe of N objects.

Coded shared cache: Without coding, the cache simply stores the content objects in the union of the demanded sets of the M ISPs. To reduce the storage cost, we allow the cache to store *coded objects*, which herein means bit-wise XOR combinations of content objects. The main operational change when moving to a coded cache is in the processing of ISP requests by the cache, and in processing the cache responses by the ISP. We discuss these in detail later in this section, paying special attention to reducing the complexity of these operations.

C. Definitions and notations

We now put the cache model in more formal terms. The demanded sets of the ISPs are represented by the $M \times N$ matrix $\mathbf{D}$, which we call *demand matrix*. Entry i, j of $\mathbf{D}$ (denoted $d_{i,j}$) is 1 if object j is in the demanded set of ISP i, and 0 otherwise. Recall from Section II-B that a 0 entry means that object j is available locally to ISP i. Let $\mathbf{d}_i$ be the i-th row of $\mathbf{D}$. The number of objects demanded by ISP i equals the number of 1s in row $\mathbf{d}_i$, which we call the *weight* of $\mathbf{d}_i$ and denote d_i. It will be useful to define the weight W of the demand matrix $\mathbf{D}$ as its maximal row weight: $W = \max_i d_i$. An important sub-class of demand matrices is *balanced* matrices, for which $d_i = W$ for all i.

When the M ISPs share the cost of the cache equally, it is natural to assume that demand matrices are balanced (each ISP is allocated the same "budget" W of demanded objects). In addition, any $\mathbf{D}$ can be made balanced *without increasing* W by changes of 0s to 1s in low-weight rows. We now turn to formulate the algebraic setup for the coded cache. Let $\{v_1, \ldots, v_N\}$ be the N content objects. In a coded cache the stored objects are *coded objects* of the form $r = \sum_{j=1}^{N} c_j v_j$, where c_j are scalar coefficients over some finite field $\mathbb{F}$. In all the coding schemes proposed in this paper, the coefficients c_j are 0 or 1 ($\mathbb{F}$ is the *binary field*, GF(2)), and summation of objects is performed as bit-wise exclusive-or (XOR) operations between objects. Overall the cache stores N_c coded objects $\{r_1, \ldots, r_{N_C}\}$, where we refer to N_c as the *cache size*, or *storage cost* (in general $N_c \leq N$). A convenient representation of the coded objects is through coding vectors and the coding matrix.

Definition 1. *A cache* coding matrix $\mathbf{C}$ *is an* $N_c \times N$ *matrix holding the coefficients of the stored coded objects. Each row* *of* $\mathbf{C}$ *is a* coding vector $\mathbf{r}_l$ *holding the coefficients of the coded object* r_l*. That is, the element* $c_{l,j}$ *of* $\mathbf{C}$ *is the* j*-th coefficient of* r_l*.*

To properly design the coding matrix $\mathbf{C}$ to have low storage cost and low per-request complexity, we need to define the request service model of the cache. The operation of the cache is done via content requests from ISPs and responses by the cache as follows. **Request ISP→cache**: ISP i sends to the cache a request for object j in its demanded set. **Response cache→ISP**: the cache sends to ISP i a packet with the size of a single object, such that after receiving it ISP i can recover object j. The task of the cache is to store sufficient and minimal information such that it can deliver, at low complexity, successful responses to any pair i, j of ISP and demanded object. The following is a simple characterization of the coded objects sent by the cache in response to ISP requests, which applies when ISP storage is uncoded. Let $E = \{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$ be the set of unit vectors in the vector space $\mathbb{F}^N$. Define E_i to be the subset of E indexed by the objects in the available set of ISP i.

Proposition 1. *A coded object* $u_{i,j}$ *sent by the cache in response to a request of* v_j *by ISP* i *has a coding vector* $\mathbf{u}_{i,j}$ *that lies in the affine space* $\mathbf{e}_j + span(E_i)$*.*

The affine space $\mathbf{e}_j + span(E_i)$ contains all translations of the unit vector $\mathbf{e}_j$ by vectors of the linear subspace spanned by the vectors of E_i. This means that for ISP i to recover v_j it needs to receive from the cache a coded object that is a linear combination of v_j and (possibly) other content objects in its available set. Since there is at least one ISP i demanding W objects not in $span(E_i)$, Proposition 1 implies the following size lower bound.

Proposition 2. *The number* N_c *of coded objects stored by the cache is at least* W*.*

When a coding matrix $\mathbf{C}$ has $N_c = W$, we refer to it as an *optimal-storage* coding matrix. In general, for $\mathbf{C}$ to fulfill all requests, a necessary and sufficient condition is that $\forall i$: $rank\left(\mathbf{C}; E_i\right) = N$, where ; represents vertical concatenation and $rank\left(\cdot\right)$ is the rank of the matrix over the binary field.

Our main concern in this work is to find matrices $\mathbf{C}$ that imply low complexity of serving content requests, where low complexity means that both the cache and the ISPs need to perform few operations to recover the requested object. We formalize this in the following. Enumeration of complexity is done using a basic unit called *operation*, which amounts to the complexity of bit-wise XORing a pair of objects. In these definitions we ignore the complexity of finding which coding objects need to be used to decode a request (this complexity is negligible when the objects are large, and can also be calculated once and stored in a lookup table). Then the cache and total complexity are given in the following two definitions.

Definition 2. *The number of operations the* cache *needs to perform to compute a coded object for ISP* i *requesting content object* v_j *is denoted* $CC_{i,j}\left(\mathbf{C}\right)$*.*

Definition 3. *The total number of operations the* cache and ISP *need to perform to recover a content object* v_j *at ISP* i *is denoted* $TC_{i,j}\left(\mathbf{C}\right)$*.*

The number of operations performed by the ISP is the difference $TC_{i,j}\left(\mathbf{C}\right) - CC_{i,j}\left(\mathbf{C}\right)$. Note that CC and TC depend on the coding matrix $\mathbf{C}$, and may differ between i, j pairs. In the sequel, we will look for coding matrices $\mathbf{C}$ that given the demand matrix $\mathbf{D}$ guarantee low CC and TC for all i, j pairs.

Example 1. *Consider the following demand matrix*

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

The weight of this matrix is $W = 2$*, and one optimal-storage coding matrix for it is*

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

When $i = 2, j = 4$ *is the request, the cache needs to compute the coded object* $r_1 + r_2$ *that has the coding vector* $\mathbf{r}_1 + \mathbf{r}_2 = [1, 0, 1, 1, 1, 1]$*. When ISP* $i = 2$ *receives this coded object, it computes* $r_1 + r_2 - v_1 - v_3 - v_5 - v_6 = v_4$*. Thus this coding matrix has* $CC_{2,4}\left(\mathbf{C}\right) = 1$ *and* $TC_{2,4}\left(\mathbf{C}\right) = 5$*. In contrast, another optimal-storage matrix*

$$\mathbf{C}' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

has $CC_{i,j}\left(\mathbf{C}'\right) = 0$ *and* $TC_{i,j}\left(\mathbf{C}'\right) = 2$ *for any* i, j*.*

D. The model as multicast network/index coding

When not considering the CC and TC complexities at the request phase, the problem of designing an optimal-storage coded cache can be posed as an *index coding* problem [6],[7]: the ISPs are clients demanding content objects, and having their available objects as side information. Moreover, because for each ISP the union of the available and demanded sets is the entire N-set, the index coding problem is a special case of *multicast network coding* [8],[9] with N sources and M sinks. Known results in multicast network coding give that achieving the lower bound $N_c = W$ is possible by linear coding over finite fields of size $q \geq M$ [10]. While there are many results reducing the required field sizes for special networks and parameters (see for example [11],[12]), no existing result implies our constructive result showing that the finite field of size $q = 2$ is sufficient for $M \leq 4$ and any N. Even more crucial than lowering the field size, our code-design algorithms offer significant reduction in the CC and TC complexities over using known code-design tools in index and multicast network coding. Since both index and network coding address the problem of recovering *all* N source objects simultaneously, no care is taken to reduce the CC and TC complexities for *individual* object requests. That means algorithms like the *linear information flow (LIF)* [10] may return solutions with CC and TC complexities that are significant fractions of N, which are clearly not applicable for designing a scalable cache for large numbers of objects.

The particular branch of network coding with the highest relevance to this work consists of schemes that consider the decoding *delay* in the code design. Both in low-delay communications and here in caching one needs to mitigate the adverse effect coding has on access to small subsets of the N objects. One coding approach for low-delay is *immediately-decodable network coding* (IDNC) [13],[14], where all coded objects are XORs of full and disjoint subsets of the N content objects. Restricting the coding solution to be IDNC results in sub-optimality, because even without this property the cache can fulfill individual object requests by sending a single coded object to the requesting ISP. A less restrictive approach is *generation-based network coding* [15],[16], where the N objects are partitioned to small subsets (generations), but a coded object is not necessarily a XOR of the full subset. Results in [16] imply that for general M (which grows with N) our code-design problem is NP-hard; but luckily, for fixed values of M and for demand matrices coming from a distribution our algorithms can still be efficient and practical (more details on the relation of the results to [15],[16] are provided in the next sections).

While discussing prior work we also mention the recent line of work on *wireless coded caching* that started in [17],[18]. Despite the similar name, the results there are not directly applicable to our problem because these results divide the content objects into small pieces, and distribute them among multiple caching nodes.

III. Code Design for $M = 3, 4$

A. Formal problem definition

Given a demand matrix $\mathbf{D}$, we now define formally the problem of designing a binary coding matrix $\mathbf{C}$ which, in addition to being optimal-storage, has low CC and TC complexities. The key step is decomposing the matrix $\mathbf{D}$ into smaller submatrices with the property that each submatrix has a low-complexity binary coding matrix, while the decomposed code is still optimal-storage.

Let $S \subset \{1, \ldots, N\}$ be a size-t subset of the N content-object indices. Define $\mathbf{D}_S$ to be the $M \times t$ submatrix of $\mathbf{D}$ taken as the subset S of the columns of $\mathbf{D}$. $\mathbf{D}_S$ specifies the ISP demands for content objects in the set S. We denote the weight of $\mathbf{D}_S$ by W_S (recall the definition of weight from Section II-C). If coding of content objects in S is done separately from content objects outside S, then we have the following.

Proposition 3. *For a request of v_j with $j \in S$ by ISP i we have $CC_{i,j}(\mathbf{C}_S) \leq W_S - 1$ and $TC_{i,j}(\mathbf{C}_S) \leq t - 1$.*

Proof: Follows from elementary linear algebra. ISP i has $w \leq W_S$ demanded content objects in S. Thus w stored coding vectors are sufficient for the cache to produce a coding vector with contribution of v_j and only objects in the available set of ISP i. A set of w vectors requires at most $w - 1 \leq W_S - 1$ operations, hence $CC_{i,j}(\mathbf{C}_S) \leq W_S - 1$. To get the bound on the total complexity, we observe that the coding vector in the cache response has contributions from at most $t - w$ objects in ISP i's available set, hence this many operations are needed to cancel them from the response coding vector. Summing the cache and ISP operations gives $t - w + w - 1 = t - 1$, and the bound on $TC_{i,j}(\mathbf{C}_S)$ follows. ■

In addition to the bounds of Proposition 3, the proof reveals that a larger demand-set size w of an ISP means higher complexity to the cache, while a smaller demand set means higher complexity to the ISP, with the total complexity being bounded by $t - 1$ operations regardless of w. We can now formulate the problem of optimal-storage, low-complexity coded caching.

Problem 1. *Given $\mathbf{D}$, find a decomposition of $\{1, \ldots, N\}$ to a collection $\mathcal{S}$ of disjoint subsets such that $\cup(S \in \mathcal{S}) = \{1, \ldots, N\}$, with the resulting submatrices $\mathbf{D}_S$ having the following properties*

1) Low total complexity: $\forall S \in \mathcal{S}$, $|S| \leq t$ for some given parameter t.
2) Low cache complexity: $\forall S \in \mathcal{S}$, $W_S \leq \ell$ for some given parameter ℓ.
3) Optimal storage cost: $\sum_{S \in \mathcal{S}} W_S = W$.
4) XOR-only coding: $\forall S \in \mathcal{S}$, $\mathbf{D}_S$ has a *binary* coding matrix $\mathbf{C}_S$ with W_S coding vectors.

The challenge of Problem 1 is that in general for a decomposition $\mathcal{S}$, property 3 is only true as a lower bound: $\sum_{S \in \mathcal{S}} W_S \geq W$. In particular, when the input demand matrix $\mathbf{D}$ is balanced, equality in property 3 requires that *all* the submatrices $\mathbf{D}_S$ are balanced as well. Solving Problem 1 in general is difficult even with only requiring properties 1-3. Hence we restrict our attention to small values of $M = 3, 4$, which turn out to be efficiently solvable with all properties 1-4.

B. Optimal-storage low-complexity coding for M=3 ISPs

In this subsection we prove the existence of a solution to Problem 1 for any $\mathbf{D}$ with $M = 3$. After proving existence in the following theorem, an efficient greedy algorithm to find the solution is explained.

Theorem 4. *For any demand matrix $\mathbf{D}$ with $M = 3$ and weight W, there exists a decomposition $\mathcal{S}$ that satisfies properties 1-4 of Problem 1 with $t = 3$ and $\ell = 2$.*

The main ingredient in the proof of Theorem 4 is the following lemma.

Lemma 5. *Any balanced demand matrix $\mathbf{D}$ with $M = 3$ and weight W can be decomposed to balanced matrices $\mathbf{D}_S$, each with at most $t = 3$ columns and weight at most $\ell = 2$.*

First note that assuming a balanced $\mathbf{D}$ is without loss of generality, since we can always add 1s to balance a non-balanced $\mathbf{D}$ without changing W. Before proving the lemma, let us explore the structure of $\mathbf{D}$ for the special case of $M = 3$. Since there are 3 rows in $\mathbf{D}$, the demand of each content object can be represented by one of 8 length-3 column vectors. An object with the vector $(0, 0, 0)^T$ is not demanded by any ISP, and at the other extreme an object with the vector $(1,1,1)^T$ is demanded by all 3 ISPs. Objects with $(0,0,0)^T$ can be discarded from $\mathbf{D}$ because they are not needed in the cache. Objects with $(1,1,1)^T$ are themselves balanced $\mathbf{D}_S$ matrices with 1 column and weight 1, satisfying the t and ℓ constraints of Lemma 5. Hence they can be extracted from the input $\mathbf{D}$ and leave us with the same problem only on a smaller balanced matrix and with smaller weight. In between there are 6 intermediate options, where the type of each vector is set as the decimal value of the vector in binary representation: $(0,0,1)^T$ is type 1, $(0,1,0)^T$ is type 2, $(0,1,1)^T$ is type 3, and so on. We denote by b_i the number of type i objects in $\mathbf{D}$.

Proof: (of Lemma 5) From the weight of the bottom row of $\mathbf{D}$ (the least significant bit of the vector) we get the following equation

$$b_1 + b_3 + b_5 + b_7 = W. \tag{1}$$

Similar equations for the other two rows give

$$b_2 + b_3 + b_6 + b_7 = b_4 + b_5 + b_6 + b_7 = W, \tag{2}$$

and rearranging the equations, one gets

$$b_1 - b_6 = b_2 - b_5 = b_4 - b_3 = \sigma, \tag{3}$$

for some integer σ. If $\sigma = 0$ and not all b_i are 0, it means that we can find a pair of vectors with types 1,6 or types 2,5 or types 4,3. In either case we can extract from $\mathbf{D}$ a balanced $\mathbf{D}_S$ matrix with 2 columns and weight 1. If $\sigma > 0$, there is a triple of vectors with types 1,2,4. In this case we can extract from $\mathbf{D}$ a balanced $\mathbf{D}_S$ matrix with 3 columns and weight 1. Similarly, if $\sigma < 0$, there is a triple of vectors with types 6,5,3. In this case we can extract from $\mathbf{D}$ a balanced $\mathbf{D}_S$ matrix with 3 columns and weight 2. After extracting the $\mathbf{D}_S$ as above, we iterate with a smaller (balanced) $\mathbf{D}$. It is readily observed that the process is guaranteed to fully decompose $\mathbf{D}$ with each $\mathbf{D}_S$ having at most $t = 3$ columns and weight at most $\ell = 2$. ∎

The balanced decomposition shown in Lemma 5 guarantees property 3, because the weight of the demand matrix decreases by W_S after extracting $\mathbf{D}_S$ from it. To complete the proof of Theorem 4, we need to show property 4, i.e., that each matrix $\mathbf{D}_S$ in the decomposition of Lemma 5 has a *binary* coding matrix with W_S coding vectors. For $S = \{j_1, j_2\}$ combining types 1,6, or types 2,5, or types 4,3, the cache stores $v_{j_1} + v_{j_2}$; for $S = \{j_1, j_2, j_3\}$ combining types 1,2,4 the cache stores $v_{j_1} + v_{j_2} + v_{j_3}$; for $S = \{j_1, j_2, j_3\}$ combining types 6,5,3 the cache stores $v_{j_1} + v_{j_2}$ and $v_{j_2} + v_{j_3}$. Also, for $S = \{j_1\}$ with type 7 the cache stores v_{j_1} uncoded. In summary, Theorem 4 implies that for $M = 3$ and arbitrary demands, one can implement a XOR-only coded cache with optimal storage, cache complexity at most 1, and total complexity at most 2 (for any request).

Decomposing $\mathbf{D}$ *efficiently* to the type combinations in the proof of Lemma 5 is simple, and we skip its formal algorithm specification. Each balanced matrix $\mathbf{D}_S$ can be found in time at most linear in N (usually much less), and for any order of extracting these $\mathbf{D}_S$ the algorithm is guaranteed to complete with all-balanced matrices. This implies a polynomial-time decomposition algorithm, and close to linear average complexity when $\mathbf{D}$ is random.

C. Optimal-storage low-complexity coding for M=4 ISPs

Our main result in this section is extending optimal-storage low-complexity coded caching to $M = 4$ ISPs. This case is more involved, since now demand column vectors fall into 16 types, twice as many as in the case $M = 3$. In the following we develop a storage-optimal decomposition algorithm, and bound the per-request cache and total complexities it guarantees.

Theorem 6. *For any demand matrix $\mathbf{D}$ with $M = 4$ and weight W, there exists a decomposition $\mathcal{S}$ that satisfies properties 1-4 of Problem 1 with $t = 5$ and $\ell = 3$.*

Here too the main proof ingredient is a decomposition lemma.

Lemma 7. *Any balanced demand matrix $\mathbf{D}$ with $M = 4$ and weight W can be decomposed to balanced matrices $\mathbf{D}_S$, each with at most $t = 5$ columns and weight at most $\ell = 3$.*

Proof: As in Section III-B, $b_1, ..., b_{14}$ are the counts of objects in $\mathbf{D}$ of types 1-14, respectively. To simplify the enumeration, we pair together types whose corresponding column vectors sum to $(1,1,1,1)^T$. For example type 1 $(0,0,0,1)^T$ is paired with type 14 $(1,1,1,0)^T$. The 7 pairs define 7 variables as follows: $x_1 = b_1 - b_{14}$, $x_2 = b_2 - b_{13}$, $x_3 = b_4 - b_{11}$ and $x_4 = b_8 - b_7$ are variables tracking pairs with 1 and 3 ones in the binary representation, and $y_1 = b_3 - b_{12}$, $y_2 = b_5 - b_{10}$ and $y_3 = b_6 - b_9$ are variables tracking pairs with 2 ones each in the binary representation. We choose to differentiate between x and y variables to better accommodate for symmetries in the sequel. Note that all the x and y variables are integers (possibly negative or zero). From the fact that $\mathbf{D}$ is balanced we can obtain the following equations

$$\begin{aligned} x_1 + y_2 - x_2 - y_3 &= 0 \\ x_1 + y_1 - x_3 - y_3 &= 0 \\ x_1 + y_1 + y_2 - x_4 &= 0 \end{aligned} \tag{4}$$

Since there are 7 variables and only 3 equations, we write the general solution for the y variables depending on the x variables

$$y_1 = \frac{x_3 + x_4 - x_1 - x_2}{2}, \; y_2 = \frac{x_2 + x_4 - x_1 - x_3}{2},$$
$$y_3 = \frac{x_1 + x_4 - x_3 - x_2}{2}. \tag{5}$$

We now show that for any values of x_1, x_2, x_3, x_4 that correspond to a $\mathbf{D}$ matrix, an invariant is maintained that we can extract a balanced $\mathbf{D}_S$ matrix with at most $t = 5$ columns. In the following we use the symmetry between the different x_i's to reduce the number of cases. *Case 1*: a single non-zero x_i (wlog[1] x_1) gives a variable vector $(x_1, x_2, x_3, x_4, y_1, y_2, y_3)$ of the form $(2X, 0, 0, 0, -X, -X, X)$, from which we can extract a balanced $\mathbf{D}_S$ with 5 vectors of types 1,1,12,10,6

[1]without loss of generality.

(if $X > 0$), and types 14,14,3,5,9 (if $X < 0$). *Case 2*: two non-zero x_i (wlog x_1, x_2) give a variable vector of the form $(X_1, X_2, 0, 0, Y_1, Y_2, Y_3)$. If both X_1, X_2 are positive then Y_1 is negative and we can extract $\mathbf{D}_S$ with 3 vectors of types 1,2,12. Other balanced $\mathbf{D}_S$ matrices can be extracted when X_1, X_2 are both negative (types 14,13,3) and when they have opposite signs (types 1,13,10,6 or types 14,2,5,9). We summarize all cases (up to symmetries) in Table I. *Case 3*: three or more non-zero x_i. Consider the two x_i variables with the largest absolute values (wlog x_1, x_2). If the sum of absolute values of x_1, x_2 is *strictly* larger than the sum of absolute values of x_3, x_4, then we are back to Case 2, because the y_i that were non-zero in Case 2 remain non-zero here and with the same signs. If the sum of absolute values of x_3, x_4 is the same, we need to consider (up to symmetries) the following options on the signs of the x_i's: $(X, X, X, X, 0, 0, 0)$, $(X, X, X, -X, -X, -X, -X)$,$(X, X, -X, -X, -2X, 0, 0)$. The first option can extract a balanced $\mathbf{D}_S$ with 4 vectors with types 1,2,4,8 (if $X > 0$) or types 14,13,11,7 (if $X < 0$). We observe that the other two options subsume Case 2 with y_1 having the correct sign to balance x_1,x_2. We complete the proof by pointing to the $\mathbf{D}_S$ matrix with the largest weight: types 14,14,3,5,9 with $W_S = 3 = \ell$. After getting to the state where all x_i, y_i are zero, we are left with pairs of matching vectors that can be coded together separately with $t = 2$, $l = 1$. ■

To complete the proof of Theorem 6, we need to show property 4. Table I presents the binary coding vectors that solve each of the type combinations extracted from $\mathbf{D}$ in Lemma 7. One can check that the binary coding vectors in the center column indeed solve the demand matrix represented by the types in the left column. The indices of the object numbers v_j are ordered to match the type order on the left. The entries of the table are given up to symmetries. For other type combinations not listed in the table, one should look for the numbers of ones in the $|S|$ columns of $\mathbf{D}_S$, find the table row with these numbers on the right column, and then map the object numbers v_j to the types to match the numbers of ones. For example, the type combination 2,14,9,5 is symmetric with 1,13,10,6 (row 5 in the table) in the numbers of ones in each column from left to right, and thus will have the same coding vectors. Not showing in the table are coding vectors for 2-column balanced matrices composed of paired types, e.g. types 1,14 and types 3,12. These are solved simply by storing $v_1 + v_2$.

In summary, Theorem 6 implies that for $M = 4$ and arbitrary demands, one can implement a XOR-only coded cache with optimal storage, cache complexity at most 2, and total complexity at most 4 (for any request). As in Section III-B for $M = 3$, the successive decomposition of $\mathbf{D}$ can be done efficiently: in polynomial time in the worst case, and close to linear when $\mathbf{D}$ is random.

IV. Randomized Code Design for Larger M

Continuing to solve Problem 1 beyond $M = 4$ becomes increasingly difficult, as the number of types needed in a decomposition lemma (like Lemmas 5,7) increases exponentially

types in $\mathbf{D}_S$	coding vectors	symmetries
1,1,12,10,6	$v_1 + v_3 + v_4$, $v_2 + v_3 + v_5$	v_1,v_2: 1 one, v_3,v_4,v_5: 2 ones
14,14,3,5,9	$v_1 + v_3$, $v_2 + v_4$, $v_1 + v_2 + v_5$	v_1,v_2: 3 ones, v_3,v_4,v_5: 2 ones
1,2,12	$v_1 + v_2 + v_3$	v_1,v_2: 1 one, v_3: 2 ones
14,13,3	$v_1 + v_3$, $v_2 + v_3$	v_1,v_2: 3 ones, v_3: 2 ones
1,13,10,6	$v_2 + v_3$, $v_1 + v_3 + v_4$	v_1: 1 one, v_2: 3 ones, v_3,v_4: 2 ones
1,2,4,8	$v_1 + v_2 + v_3 + v_4$	v_1,v_2, v_3, v_4: 1 one
14,13,11,7	$v_1 + v_2$, $v_1 + v_3$, $v_1 + v_4$	v_1,v_2, v_3, v_4: 3 ones

TABLE I: Translation between binary coding vectors and type combinations.

with M. In fact, when M is not bounded (grows with N), a problem similar to Problem 1 properties 1-3 has been shown to be NP-hard [16] (in their problem only the bound $W_S \le \ell$ was given and the number of the matrices $\mathbf{D}_S$ minimized, while in this paper the additional bound $|S| \le t$ also yields a bound on total complexity). Another issue with increasing M is the requirement to have binary (XOR-only) coding: already when $M = 6$ there exist demand matrices that do not admit optimal-storage binary coding matrices. Our approach for general M is thus the following:

1) Partition the matrix $\mathbf{D}$ to balanced sub-matrices $\mathbf{D}_S$ of weight $W_S \le 2$.
2) For each $\mathbf{D}_S$ find a binary coding matrix (when one exists) using a simple colorability condition.

In the first item we fix $\ell = 2$ to have low cache complexity and a more tractable partition problem, and in the second item we characterize a condition that will conveniently find a binary coding matrix $\mathbf{C}_S$ for the $\mathbf{D}_S$ found in item 1, when a binary coding matrix exists. Note that with this approach to general M we are still seeking optimal-storage coding matrices, up to having some $\mathbf{D}_S$ matrices that do not admit a binary coding matrix, and thus need to be repartitioned or encoded with more than 2 code symbols. We start our discussion of general M with a condition on binary-codable weight-2 matrices.

A. A colorability condition for binary-coded weight-2 matrices

Given a demand sub-matrix $\mathbf{D}_S$ of weight 2 and arbitrary number of columns, we look for a condition to determine if $\mathbf{D}_S$ can be binary coded using 2 coding vectors. For simplicity we assume that all M rows of $\mathbf{D}_S$ have weight 2 (exactly 2 ones in each row), but an immediate extension addresses the case where some rows have smaller weight. Toward such a condition, we define the induced graph of the matrix $\mathbf{D}_S$.

Definition 4. *Given a balanced weight-2 matrix, the* induced graph *of the matrix is one whose vertices correspond to the matrix columns, and each row defines an edge between the two vertices (columns) where this row has ones.*

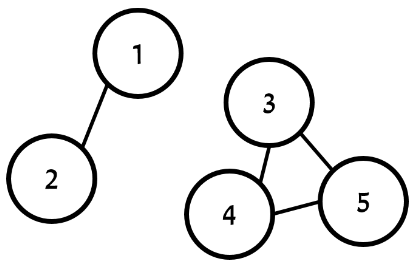

Figure 2: The induced graph of a sample demand matrix. Vertices represent the columns, and edges the (weight-2) rows.

Example 2.

Consider the demand matrix

$$\mathbf{D}' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

There are $n = 5$ objects (columns) and $M = 4$ ISPs (rows) mapped to $|V| = 5$ vertices and $|E| = 4$ edges in the induced graph. Numbering the objects from left to right we draw an edge between vertices 1 and 2 (row 1), and a clique of 3 vertices $3, 4, 5$ (rows $2, 3, 4$). Fig. 2 shows the resulting graph.

A necessary and sufficient condition for binary coding is given in the following proposition.

Proposition 8. *A balanced weight-2 demand matrix has a 2-vector binary coding matrix if and only if its induced graph is 3-colorable.*

Recall from classical graph theory that a graph is 3-colorable if we can assign to each vertex one of 3 colors such that no pair of vertices with an incident edge shares the same color.

Proof: Positive direction (3-coloring $\rightarrow$ binary coding): assume the induced graph has a coloring with 3 colors (named A, B, C). Given such a coloring we explicitly construct the 2 coding vectors $\mathbf{r}_1, \mathbf{r}_2$: a vertex j colored A has 1 in the j-th coordinate of $\mathbf{r}_1$ and 0 in the j-th coordinate of $\mathbf{r}_2$. Similarly, a vertex colored B has 0 in $\mathbf{r}_1$ and 1 in $\mathbf{r}_2$, and a vertex colored C has 1 in both $\mathbf{r}_1$ and $\mathbf{r}_2$. It can be seen that for any combination of two distinct colors incident on a demand edge, the two demanded objects can be recovered: from r_1, r_2 (if A and B), or from $r_1, r_1 + r_2$ (if B and C), or from $r_2, r_1 + r_2$ (if A and C).

Converse direction (binary coding $\rightarrow$ 3-coloring): we apply essentially the same mapping in reverse from $\mathbf{r}_1, \mathbf{r}_2$ to A, B, C. If $\mathbf{r}_1, \mathbf{r}_2$ are working coding vectors then it cannot happen that the j-th coordinate of both is 0, unless vertex j has no edges (object not demanded). A vertex with no edges can be colored arbitrarily without affecting the coloring constraint. Thus we can apply the reverse mapping of the one in the positive direction. Then a similar argument to the first half of the proof shows that no pair of vertices with an edge can have the same color, otherwise the corresponding 2 demands cannot be recovered by $\mathbf{r}_1, \mathbf{r}_2$. ∎

While determining 3-colorability is a hard problem for general graphs (3-coloring is a well-known NP-complete problem), our graphs here are quite small – only M edges and the number of vertices that cannot be much larger than M, unless the problem is trivialized by a very sparse demand matrix. In practice, in the empirical study shown in Section IV-C the coloring instances were indeed easy to solve.

B. Partitioning the demand matrix to weight-2 sub-matrices

Once we know how to find optimal-storage binary coding vectors for weight-2 demand sub-matrices of arbitrary number of columns, we are motivated to partition the matrix $\mathbf{D}$ to such sub-matrices. We recall from [16] that for general M it is NP-hard to partition $\mathbf{D}$ to a minimal number of matrices $\mathbf{D}_S$ with bounded weight ℓ, even if $\ell = 2$. Furthermore, since typical N (number of objects) is extremely large, even polynomial-time complexity is not sufficient, and we need near-linear complexity. Our approach to optimal partitioning will thus be based on randomized algorithms, whose low running times will be shown assuming that $\mathbf{D}$ comes from a known distribution. The distribution we assume here for $\mathbf{D}$ is the *i.i.d. Bernoulli distribution* on its elements; but the same algorithms can be applied for other distributions (in fact, due to the large size of $\mathbf{D}$, the algorithms are seen empirically to work well even when $\mathbf{D}$ does not follow any synthetic distribution). In the remainder of the section we treat $\mathbf{D}$ as a *random demand matrix*, according to the following definition.

Definition 5. *A demand matrix $\mathbf{D}$ is called* random *with* load p *if each entry $d_{i,j}$ in it is an i.i.d. Bernoulli random variable with* $\Pr(d_{i,j} = 1) = 1 - \Pr(d_{i,j} = 0) = p$.

The basic method of randomized partitioning algorithms is to randomly draw columns from $\mathbf{D}$ until balanced matrices $\mathbf{D}_S$ are found with weight 1 or 2. This gives low-complexity partition algorithms because the number of iterations needed on average to find a balanced sub-matrix is a constant independent of N (but dependent on M, p, and the sophistication of the algorithm), which results in overall *linear average complexity*. The simplest manifestation of this method is given in Algorithm 1.

Algorithm 1. ***Full-stroke balanced weight 1 or 2***

```
1) initialize an empty column set $S = \emptyset$
2) while the columns in $S$ do not form a balanced weight $1$ or $2$ matrix
3)     choose a random column index and add to $S$
4)     go to 1 if some row exceeded weight $2$
5) return balanced weight 1 or 2 matrix $\mathbf{D}_S$
```

After each completed run of Algorithm 1 we re-run it on the matrix remaining after removing $\mathbf{D}_S$. Algorithm 1 finds only balanced sub-matrices, so it guarantees optimal storage (subject to weight-2 $\mathbf{D}_S$ being 3-colorable for binary coding). We next want to analyze the running time of Algorithm 1. Toward that we give the following two definitions.

Definition 6. *Given p and M (kept implicit for convenience), denote by $Q(m)$, $m \leq M$, the probability that starting from a* *matrix corresponding to a column set with m rows of weight 0 and $M-m$ rows of weight 1, successive column drawing will reach a balanced weight-1 matrix $\mathbf{D}_S$ without any row exceeding weight 1 before.*

Clearly $Q(M)$ is the probability that Algorithm 1 returns a weight-1 matrix $\mathbf{D}_S$ in a single execution of the while loop. A similar definition for weight-2 now follows.

Definition 7. *Given p and M (kept implicit for convenience), denote by $Q(m_1,m_2)$, $m_1 \le m_2 \le M$, the probability that starting from a matrix corresponding to a column set with m_1 rows of weight 0, m_2 rows of weight < 2, and $M-m_2$ rows of weight 2, successive column drawing will reach a balanced weight-2 matrix $\mathbf{D}_S$ without before reaching a balanced weight-1 matrix or exceeding weight 2 in any row.*

Note the dependence between m_1 and m_2 requiring $m_1 \le m_2$. Clearly $Q(M,M)$ is the probability that Algorithm 1 returns a weight-2 matrix $\mathbf{D}_S$ in a single execution of the while loop. Moreover, $Q(M)+Q(M,M)$ is the probability that a single while loop of Algorithm 1 is successful, i.e., it returns a weight 1 or 2 matrix without re-initializing S to the empty set. Knowing $Q(M)$,$Q(M,M)$ is thus crucial for obtaining the expected runtime of Algorithm 1, which is the reciprocal of $Q(M)+Q(M,M)$ by the geometric distribution. The following propositions derive $Q(m)$ and $Q(m_1,m_2)$ and give $Q(M)$ and $Q(M,M)$ as special cases.

Proposition 9. *Given p and M, $Q(m)$ from Definition 6 can be calculated by the following recursive expression, with base case $Q(0)=1$.*

$$Q(m) = \left[1-P_0(M)\right]^{-1} P_0(M-m) \sum_{i=1}^{m} P_1(m,i)\, Q(m-i)\,, \tag{6}$$

where $P_0(n) \triangleq (1-p)^n$ and $P_1(n,i) \triangleq \binom{n}{i} p^i (1-p)^{n-i}$.

Proof: For the base case $m=0$, $Q(0)=1$ because by definition the matrix is balanced weight 1. To succeed in reaching balanced weight 1 from $m>0$ weight-0 rows, we need a column that has ones in i of these m rows and zeros elsewhere (happens with probability $P_0(M-m)\,P_1(m,i)$), and then succeed from $m-i$ weight-0 rows (happens with probability $Q(m-i)$). The sum in (6) adds products of these probabilities for $i=1,\ldots,m$. The first term in the right-hand side accounts for the possibility to have $i=0$ (drawing the all-zero column). ■

Proposition 10. *Given p and M, $Q(m_1,m_2)$ from Definition 7 can be calculated by the following recursive expression, with base cases $Q(0,0)=1$ and $Q(0,M)=0$.*

$$Q(m_1,m_2) = \left[1-P_0(M)\right]^{-1} P_0(M-m_2)\cdot \sum_{i=0}^{m_1} \sum_{\substack{j=0 \\ i,j\neq 0,0}}^{m_2-m_1} P_1(m_1,i)\, P_1(m_2-m_1,j)\, Q(m_1-i,m_2-j)\,, \tag{7}$$

where as before $P_0(n) \triangleq (1-p)^n$ and $P_1(n,i) \triangleq \binom{n}{i} p^i (1-p)^{n-i}$.

Proof: For the base case $m_1=m_2=0$, $Q(0,0)=1$ because by definition the matrix is balanced weight 2. For the base case $m_1=0$,$m_2=M$, $Q(0,M)=0$ because by definition the matrix is balanced weight 1, and the algorithm returns a weight-1 and not weight-2 matrix. Similarly to Proposition 9, the probability to succeed from state m_1,m_2 is a sum over i,j of the recurrence while hitting with the new column i rows of weight 0, j rows of weight 1, and no rows with weight 2. The sum in (7) adds all these probabilities, and the first term in the right-hand side accounts for the possibility to have $i=j=0$ (drawing the all-zero column). ■

Having the analytical tools of $Q(m)$ and $Q(m_1,m_2)$ now allows us to define a better partition algorithm which, instead of starting from an empty set S after failure, "back tracks" to the optimal previous subset in terms of success probability. We present this as Algorithm 2.

Algorithm 2. ***Back-tracking balanced weight 1 or 2***

```
1) initialize an empty column set $S=\emptyset$
2) while the columns in $S$ do not form a balanced weight $1$ or $2$ matrix
3)     choose a random column index and add to $S$
4)     if some row exceeded weight $2$, remove the $l \ge 1$ last added columns such that the resulting matrix has maximal $P(m_1,m_2)$, where
           $P(m_1,m_2) = \begin{cases} Q(m_1)+Q(m_1,M) & \text{if } m_2 = M \\ Q(m_1,m_2) & \text{otherwise} \end{cases}$
5) return balanced weight 1 or 2 matrix $\mathbf{D}_S$
```

Algorithm 2 is a generalization of Algorithm 1 in the sense that instead of always re-initializing S to the empty set, it removes from S only the last l added columns, where l is determined to maximize the probability to reach a balanced matrix from the state corresponding to the columns remaining in S. It is shown in the next subsection that Algorithm 2 offers a significant improvement over its special case Algorithm 1.

Next we want to again improve over the basic Algorithm 1, but this time without the search required for back-tracking in Algorithm 2. Observe from the definitions of $Q(m)$ and $Q(m_1,m_2)$ that we are so far fixed to the special case of an algorithm aiming at reaching a balanced-matrix state $m_1=m_2=0$ (weight 2) or $m_1=0,m_2=M$ (weight 1). For many values of M and p, the algorithm will perform better if first aiming to some intermediate state m_1,m_2, and only from there to the balanced states (returning to m_1,m_2 if failing to reach a balanced matrix). We call this the *2-stroke approach*, where our objective is to find analytically the intermediate state m_1,m_2 that gives the lowest expected number of iterations before reaching a balanced matrix, and in particular lower than the 1-stroke approach of Algorithm 1. For this task we will need the following definition.

Definition 8. *Given p and M (kept implicit for convenience), denote by $F(m_1,m_2)$, $m_1 \le m_2 \le M$, the probability that starting from a matrix corresponding to an empty column set S, successive column drawing will reach a column set with m_1 rows of weight 0, m_2 rows of weight < 2, and $M-m_2$ rows* *of weight* 2 *before reaching a column set with either 1)* $< m_1$ *rows of weight* 0*, or 2)* $< m_2$ *rows of weight* < 2*, or 3) any row with weight* > 2.

The interpretation of Definition 8 is that an algorithm that starts from an empty S and aims at state m_1, m_2 will succeed to reach it with probability $F(m_1, m_2)$. Such 2-stroke algorithm will re-initialize S to empty when clear that m_1, m_2 cannot be reached, happening in the three cases specified in Definition 8. The formal definition of the 2-stroke algorithm is given as Algorithm 3. An important step of the algorithm is to find the optimal intermediate state aimed in the first stroke. For this step we need the following proposition.

Proposition 11. *Given* p *and* M*,* $F(m_1, m_2)$ *from Definition 8 can be calculated by the following recursive expression, with base case* $F(M, M) = 1$.

$$F(m_1, m_2) = \sum_{\substack{i=0 \\ i,j \neq 0,0}}^{m_2-m_1} \sum_{j=0}^{M-m_2} P_0(M - m_2 - j) P_1(m_1 + i, i) \cdot$$

$$P_1(m_2 - m_1 + j - i, j) \frac{F(m_1 + i, m_2 + j)}{1 - P_0(M)}, \quad (8)$$

where as before $P_0(n) \triangleq (1-p)^n$ *and* $P_1(n,i) \triangleq \binom{n}{i} p^i (1-p)^{n-i}$.

Proof: For the base case $m_1 = m_2 = M$, $F(M, M) = 1$ because by definition the initial empty matrix satisfies $m_1 = m_2 = M$. The recursion step works similarly to Proposition 10 but in the reverse direction of decreasing m_1, m_2. The probability to reach state m_1, m_2 is a sum over i, j of reaching $m_1 + i, m_2 + j$ first, and then hitting with the new column i rows of weight 0 (out of $m_1 + i$), j rows of weight 1 (out of $m_2 + j - (m_1 + i)$), and no rows of weight 2 (out of $M - (m_2 + j)$). The sum in (8) adds all these probabilities, and the denominator accounts for the possibility to have an arbitrary number of all-zero columns drawn before moving from $m_1 + i, m_2 + j$ to m_1, m_2. ∎

In the 2-stroke algorithm the expected running time will consist of the iterations needed to reach m_1, m_2 plus the iterations needed to reach a balanced matrix when starting from m_1, m_2. Assuming independence of drawing as before we get from the geometric distribution that the expected number of iterations in 2-stroke is $1/F(m_1, m_2) + 1/Q(m_1, m_2)$. Thus the intermediate state sought in the first stroke will be chosen as the m_1, m_2 that minimizes this sum.

Algorithm 3. ***2-stroke balanced weight 1 or 2***

```
1) Find $m_1^*, m_2^*$ that minimize $1/F(m_1, m_2) + 1/Q(m_1, m_2)$
2) initialize an empty column set $S' = \emptyset$
3) while the columns in $S'$ do not form a matrix with state $m_1^*, m_2^*$
4)     choose a random column index and add to $S'$
5)     go to 2 if for new $S'$: $m_1 < m_1^*$, or $m_2 < m_2^*$, or some row exceeds weight $2$
6) initialize $S = S'$
7) while the columns in $S$ do not form a balanced weight 1 or 2 matrix
8)     choose a random column index and add to $S$
9)     go to 6 if some row exceeded weight $2$
10) return balanced weight 1 or 2 matrix $\mathbf{D}_S$
```

Note that Algorithm 3 does not require a search in the drawing phase as is the case for Algorithm 2. Rather, the optimal m_1^*, m_2^* is calculated once and remains static so long that p does not change significantly. Optimal first-stroke states can also be stored in a lookup table for different values of p. An important consideration for Algorithm 3 is to compare the best expected number of 2-stroke iterations $1/F(m_1^*, m_2^*) + 1/Q(m_1^*, m_2^*)$ to the best expected number of 1-stroke iterations $1/(Q(M) + Q(M, M))$; if the latter is smaller, one should prefer Algorithm 1 over Algorithm 3.

C. *Evaluation of randomized partitioning algorithms*

We now move to examine and compare the performance of Algorithms 1-3. Recall that all three algorithms reach a balanced weight 1 or 2 matrix in each round of drawings. Hence they all maintain the optimal-storage property, and only differing in their running times due to the different expected numbers of iterations in their main loop. Fig. 3 shows the results. It can be seen that as expected Algorithm 2 (backtracking) gives the best performance, and that Algorithm 3 improves significantly over the basic Algorithm 1. When the load gets to about 0.4, Algorithms 1,3 give essentially the same performance. Only close to load 0.5 does Algorithm 2 lose its advantage over the other searchless two. The overall average complexity of the randomized partition algorithm is the value in the y-axis of Fig. 3 times the number of objects N (by elementary counting the average number of times a column is chosen by the algorithm equals the average number of while iterations per matrix $\mathbf{D}_S$). Our results also show the percentage of balanced weight-2 matrices that correspond to 3-colorable induced graphs and thus have binary coding with optimal storage. Fig. 4 plots the percentage of successful coloring (3 or less colors) as a function of load, and for different numbers of ISPs (i.e. rows in $\mathbf{D}$). 3-colorability was tested by a greedy (sub-optimal) coloring algorithm on submatrices obtained by Algorithm 1. Even with this simple algorithm the percentage of 3-colorable induced graphs is shown to be very close to 1 (the numbers in the plot are in general only lower bounds on the 3-colorability success probability, and the exact values may be even higher).

V. Conclusions and Future Work

This paper studies the code-design problem for shared caches with a strong emphasis on low per-request complexity. It shows that when the number of ISPs is much smaller than the number of objects, efficient coded caches can be constructed either deterministically or probabilistically. In the following we suggest interesting directions for future work.

1) **Limits of binary coding**. The most direct theoretical open problem from this paper is to find the limit of binary coding in the setup of caching with uncoded ISP

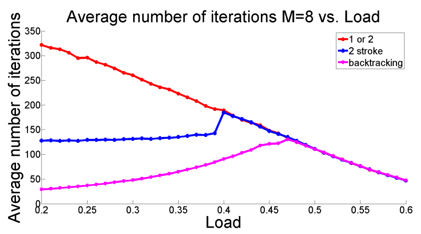

Figure 3: The average number of iterations per matrix $\mathbf{D}_S$ of the three randomized partitioning algorithms.

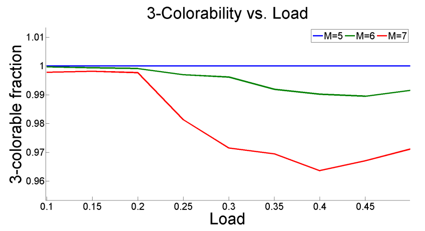

Figure 4: A lower bound on the probability that a submatrix from a randomized partition algorithm (specifically, Algorithm 1) be 3-colorable.

storage. This paper shows that binary coding is optimal-storage sufficient for any demand matrix with $M = 4$ ISPs, and for $M = 6$ there are examples of demand matrices that cannot be solved with binary coding. Answering whether (and how) $M = 5$ is solvable with binary coding is an open problem of prime interest.

2) **Real-valued demand matrices**. The assumption made in this paper is that the ISPs are those that make the decisions of which objects the cache needs to store for them. In reality it may be the case that ISPs prefer to submit "softer" preferences to the cache, not necessarily hard 0/1 decisions. Such softer preferences can greatly improve the cache efficiency, as they will add flexibility to populate the cache for better joint performance. It is an interesting future direction to adapt the present algorithms and propose new ones for soft demands given in real-valued $\mathbf{D}$ matrices.
3) **Game theory and mechanism design**. Continuing on item 2 above, it will be interesting to develop coding schemes that ensure good caching performance under different models of (non) cooperation between the ISPs. Since ISPs are simultaneously in cooperation and in competition, solving such problems is necessary to ensure robust joint operation of the cache.
4) **Real-system implementation**. An extremely important future work is to see the performance of coded-caching schemes within a real system serving real video content. In that environment, many new issues will have to be studied. For example, objects of variable size, priorities between objects and between ISPs, richer network architectures, and many more.

VI. Acknowledgement

We thank Tuvi Etzion for useful comments on the manuscript. This work was supported in part by the Israel Science Foundation (ISF), by the US-Israel Binational Science Foundation (BSF), and in part by the US National Science Foundation (NSF) under Grants 1642983 and 1718658.

References

[1] X. Tang and S. Chanson, "Coordinated en-route web caching," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595–607, 2002.
[2] A. Jiang and J. Bruck, "Optimal content placement for en-route web caching," in *Second IEEE International Symposium on Network Computing and Applications, NCA 2003*.
[3] E. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," in *Proceedings of IEEE INFOCOM 2009*.
[4] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proceedings of IEEE INFOCOM 2010*.
[5] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *Proceedings of IEEE INFOCOM 2013*.
[6] Z. Bar-Yossef, Y. Birk, T. Jayram, and T. Kol, "Index coding with side information," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1479–1494, 2011.
[7] S. El Rouayheb, A. Sprintson, and C. Georghiades, "On the index coding problem and its relation to network coding and matroid theory," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3187–3195, 2010.
[8] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.
[9] T. Ho and D. Lun, *Network coding: an introduction*, Cambridge University Press, 2008.
[10] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain and L. Tolhuizen "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, 2005.
[11] M. Feder, D. Ron, and A. Tavory, "Bounds on linear codes for network multicast," in *Electronic Colloquium on Computational Complexity*, 2003, Rep. 33.
[12] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 829–848, 2006.
[13] D. Nguyen, T. Tran, T. Nguyen, and B. Bose, "Wireless broadcast using network coding," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 2, pp. 914–925, 2009.
[14] X. Li, C.-C. Wang, and X. Lin, "On the capacity of immediately-decodable coding schemes for wireless stored-video broadcast with hard deadline constraints," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 5, pp. 1094–1105, 2011.
[15] M. Yu, P. Sadeghi, and N. Aboutorab, "On deterministic linear network coded broadcast and its relation to matroid theory," *IEEE Information Theory Workshop, ITW 2014*.
[16] M. Yu, P. Sadeghi, and A. Sprintson, "The benefit of limited feedback to generation-based random linear network coding in wireless broadcast," *arXiv preprint arXiv:1506.01150, 2015*, 2015. [Online]. Available: http://arxiv.org/abs/1506.01150
[17] M. Maddah-Ali and U. Niesen, "Fundamental limits of caching," in *IEEE International Symposium on Information Theory, ISIT 2013*.
[18] R. Pedarsani, M. Maddah-Ali, and U. Niesen, "Online coded caching," in *IEEE International Conference on Communications, ICC 2014*.