2020 18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt – CCDWN Workshop)

Impact of Popular Content Relational Structure on Joint Caching and Recommendation Policies

Marina Costantini, Thrasyvoulos Spyropoulos
EURECOM, Sophia-Antipolis, France

***Abstract*—Recent work has shown that the performance of caching systems can be boosted by the delivery of alternative but related content. In this setup, recommendation systems are exploited to offer the user appealing alternatives if the original request is not found in the cache. This framework relies on the assumption that a content can partially or completely substitute another if they are sufficiently similar.**

In this work we model these similarity relations as a graph, where each content is a node and related contents are linked by an edge. We then study how the characteristics of this *content graph* constrain the gains of designing jointly the caching and the recommendations with respect to just using a simple baseline in the soft cache hits setup. We start by selecting a number of descriptive graph features that we expect to play a role in the performance of related content delivery policies. We then analyze the effect of each of these features on the policies' performance through extensive simulations with synthetic data. Our results confirm that features such as the degree distribution and clustering coefficient of the graph are crucial to decide whether the close-to-optimal algorithm will perform significantly better than the simple baseline. Our experiments with four real-world distinct datasets further support these observations. Motivated by these clear dependencies, we conclude by showing that we can train a classifier to predict the gains attainable by a "smart" policy with respect to a low-complexity baseline using only a few macroscopic graph features as predictor variables.

I. Introduction

Storing popular items at the edge of the wireless network (i.e. *caching*) is one of the main solutions that have been proposed to deal with the large, ever-increasing demand of content stored in the cloud [1], [2]. Caching at the network edge is beneficial for both the users, who can get content at a higher bitrate and reduced latency [3], and the network operators, who can thus alleviate the congestion during peak traffic periods by having a large fraction of requests being served by the local caches [4]. The cached content can then be updated during off-peak hours.

Recently, a number of works have considered using recommendation systems as a means to improve caching efficiency [5]–[12] and demonstrated interesting performance benefits in various scenarios. These contributions propose to use recommendation systems to encourage the user to pick up contents that are both of their interest and can be found in the local cache, thus improving the number of cache hits while still satisfying the user's preferences.

The idea of *related content* naturally motivates the modeling of the set of potentially requested contents (the *catalog*) as a graph where each content is a node and related contents are linked by a (possibly weighted) edge. The structure of this graph can have a significant impact on the performance of policies for related content delivery: if e.g. there exist tight communities of many interconnected contents, any of these could be recommended as an alternative for any of the other, potentially increasing the gains of joint recommendation and caching algorithms.

However, the problem of the joint design of these variables is NP-hard [8], [13], and even polynomial approximation algorithms can become prohibitively slow for moderate catalogue sizes. What is more, such algorithms provide the optimal values of the control variables but no insights as to what key parameters lead to these choices, or why the gains are sometimes moderate and others large, depending on the catalog considered.

Thus, a number of intriguing questions arise: *(1) What is the added value of doing the joint optimization when considering e.g. the Netflix catalog of movies? (2) How does this value depend on key properties of the graph? Will it change if we consider a catalogue of Amazon Prime movies instead?* and *(3) Could we use these properties to predict performance on a new catalog, without actually running the optimization algorithm?*

In this paper, we attempt a first-of-its-kind preliminary investigation of these questions, by identifying *which* characteristics of the content graph affect the policies for caching and recommendation and *how* they do so. To better illustrate our methodology and findings, we focus on the specific setup of [13], where *soft cache hits* and a limited number of recommendations are introduced in the system.[1] To test the benefits of the joint design we compare the performance of (i) a close-to-optimal policy for the joint problem and (ii) a baseline obtained by a simple decomposition of the problem that loosely corresponds to the current state-of-the-art. Extensive simulations with both synthetic and real-world data confirm the tight relation between the graph structural properties of the content catalog and the policies' performance.

Our main contributions, each tackling one of the questions posed before, can be summarized as follows:

1) We formalize the model of a content catalog as a graph and identify key content graph and network setup parameters[2] (e.g. degree distribution, clustering coefficient for

[1]Our analysis can be readily applied to other related problem setups, as we will discuss later.

[2]We remark that we will use the word "network" to refer to the communication and content distribution network, while we will reserve the word "graph" for the relational model of the content catalog.

ISBN 978-3-903176-29-4 © 2020 IFIP

the former and cache size, number of recommendations for the latter) that have an impact on the performance of the caching and recommendation algorithms. We isolate their individual effect by running experiments with synthetic data and changing one parameter at a time, and observe their impact on the absolute and relative performances of the two policies considered.

2) We validate our findings by running the policies on four real-world datasets and attempt an analysis of the outcome based on the graph features of each dataset. Furthermore, we show that modifications of the network setup parameters affect the joint policy's performance in each trace differently, and these differences highly depend on the traces' graph structure.
3) Inspired by the dependencies observed between the graph structural parameters and policy performance, we do a preliminary test on training a classifier to predict whether the relative performance between the policies will exceed a given threshold using only a handful of content graph and network setup parameters as predictor variables, with very encouraging results.

II. Problem Setup and Policies

We consider the soft cache hits setup with limited number of recommendations [13], where users request files to a local server. The server is equipped with a cache that can store a fraction of the complete catalog of contents. The requests can either be served by the local cache if it contains the content requested ("direct" hit) or with a related content otherwise (which will give a "soft" hit if accepted by the user). Our model assumes that the substitutes offered to the user are provided by a recommendation system that looks for related content stored in the cache.[3] This notion of content relation naturally motivates the modelling of the catalog as a graph. In this work, we analyze how the structural characteristics of this *content graph* constrain the gains attainable by a close-to-optimal policy against a simple baseline. This analysis, as we show in section IV, may allow us to state *a priori* whether, for given a content graph, it is worth to find the optimal cache configuration and content recommendations or a low-complexity heuristic will perform similarly enough, only based on the graph features and without having to actually run the complex algorithm.

In this section we give the details about each of the system components introduced above. The soft cache hits setup has been already studied in detail in [9], [13], but we give a brief summary here to set the goals of this paper and interpret the results of the experiments shown in sections III and IV. Table I summarizes some key network setup notation.

A. Content Model

We denote with $\mathcal{K}$ the catalog of contents from which the users make their requests ($|\mathcal{K}| = K$). We will reuse the adjacency matrix from [9]:

$$U = \{u_{ij} \in [0,1]\},\ i,j = 1,\dots,K$$

to denote this *content graph*, where u_{ij} denotes the probability that a user accepts content j when they had initially requested content i.[4] For example, if $\mathcal{K}$ is a catalog of music videos, two songs of the same artist will be linked with a high weight and two songs from different artists but of the same genre will be linked with a low weight. Thus, unrelated contents i and j will have $u_{ij} = 0$ (no link in the content graph), and $u_{ii} = 1\ \forall i$.

TABLE I: Network Setup Notation

C	Storage capacity of the cache
x_i	Content i is stored in the cache ($x_i = 1$) or not ($x_i = 0$)
$\mathcal{K}$	Set of contents ($\lvert\mathcal{K}\rvert = K$)
u_{ij}	Probability of accepting substitute j when i is requested, $U = \{u_{ij}\}$
p_i	Probability of content i being requested
y_{ij}	Recommend j to substitute i ($y_{ij} = 1$) or not ($y_{ij} = 0$)

B. Recommendation Model

Requests for contents in $\mathcal{K}$ arriving to the server are random and i.i.d, where content i is requested with probability p_i. In our simulations the p_i follow a Zipf distribution as shown in related literature [14].

Our model assumes that given a request for a content i that is not locally available, the user will be offered N alternative contents j that they might accept with probability u_{ij}. We use the variable

$$y_{ij} \in \{0,1\},\ i,j = 1,\dots,K$$

to indicate whether content j is recommended if a requested content i is not found in the cache, and we require $\sum_{j=1}^{K} y_{ij} \leq N\ \forall i$.

C. Caching Model

In order to isolate our analysis from the additional gains from femto-caching, we focus on a single cache of size $C \ll K$.[5] We use variables

$$x_i \in \{0,1\},\ i = 1,\dots,K$$

to indicate whether content i is stored in the cache. These values must satisfy the cache capacity constraint $\sum_{i=1}^{K} x_i = C$.

When a new request for a content i arrives to the server three things can happen:

1) Direct hit: The content is found in the cache.

[3]This is assumed to be the respective app recommender (e.g., YouTube recommender), which already knows and utilizes these relations for baseline (i.e., non cache-aware) recommendations.

[4]In practice, a recommender will not have these probabilities but the content relations through, e.g., some collaborative filtering algorithm. The actual probabilities will be a function of such relations (e.g., linearly dependent, or other more complex function), which might furthermore depend on additional incentive mechanisms that can also be learned by the recommender. Without loss of generality, we assume that the u_{ij} probabilities are (normalized) linear functions of the baseline relations of files.

[5]The gains attainable in the multi-cache, multi-user scenario for the soft cache hits setup have been studied in [9], [13] and are orthogonal to the gains analyzed here. In such scenarios, the u_{ij} and p_i quantities could differ between users. Here we assume that the values given to these parameters are aggregated values, i.e. they correspond to the "average user".

2) Soft hit: The cache does not contain the request but the user accepts an alternative content offered by the recommender.

3) Cache miss: Neither the requested item nor any related content is found in the cache. There are no hits accrued independently of the decisions for y_{ij} taken by the recommender.

D. Performance Metric

Here we will evaluate the performance of each policy by its attained Cache Hit Ratio (CHR), which measures the expected quotient of the number of hits to the number of requests.

In our setup, where we consider soft hits and a limited number of recommendations, any suitable policy has to choose the variables x_i and y_{ij} trying to maximize the CHR while satisfying the constraints presented above. This problem was introduced and solved in [13], and can be summarized as:

$$\max_{x_i, y_{ij}} \quad \text{CHR} = \sum_{i=1}^{K} p_i \left[1 - \prod_{j=1}^{K} (1 - x_j \cdot u_{ij} \cdot y_{ij}) \right] \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{K} x_i \leq C \tag{2}$$

$$\sum_{j=1}^{K} y_{ij} \leq N, \forall\, i \tag{3}$$

$$x_i, y_{ij} \in \{0, 1\} \tag{4}$$

In the problem above the objective function (1) computes the expectation of a cache hit over all contents in the catalog. To see this, let us assume for a moment that the $u_{ij} \in \{0, 1\}$. Then, for each i the product term in (1) will be 1 iff neither i or any related content j are cached. In such case requesting i leads to a miss. Conversely, if either i or a related content j are cached *and* recommended, then the product term is 0, and the CHR increases an amount p_i. Thus, (1) computes the expected CHR over all contents i, where the hit can come either from getting the request i directly or from a related content that is both cached and recommended.[6] In the general case where $u_{ij} \in [0, 1]$, eq. (1) still computes the expected CHR, as shown in Lemma 1 in [9]. Constraint (2) states that we cannot cache more contents than the capacity of the cache, and constraint (3) limits the number of recommended items to N. Altogether, the general objective could be loosely described as "maximizing the cache hit rate for the operator while making sure that the recommendations shown to the user are relevant".

E. Caching and Recommendation Policies

In order to test whether we can predict the performance benefits of optimal joint caching and recommendation based solely on a few content graph and network setup parameters, we considered two different policies:

[6]This formulation assumes that a maximum of N alternative contents are presented one by one to the user, and the next content is presented only if all the previous were rejected. We plan to consider alternative user models in future work.

TABLE II: Graph Parameter Notation

R	Node degree
$E[R]/K$	Density
α	Zipf popularity exponent
p_{link}	Link probability in an ER graph
ℓ_{new}	Number of links added per node in a BA graph
$\gamma(X)$	Skewness of random variable X
κ	Clustering coefficient
n	Newman's modularity

Popularity Caching (POP): This approach decomposes the joint problem of finding the x_i and the y_{ij} into two sequential choices: it first caches the most popular contents (i.e. highest p_i) until the cache is full, and then for each item i it recommends the N contents j with highest u_{ij} out of the ones cached (more details in [13]). This simple strategy could be used in the cases where the caching and recommendation decisions are done by different entities (e.g. by the network operator and the content provider, respectively), but the recommender can know what is cached. This policy is our baseline.

Joint Caching and Recommendation (JCR): This policy approximates the optimal solution of the *joint* problem of Eqs. (1)-(4) with a polynomial algorithm based on a primal decomposition of the problem. It provides a theoretical worst-case guarantee of 63% and in practice it achieves almost optimal performance in most scenarios considered (see [13] for more details). Thus, we will consider the performance of JCR our reference of the optimal performance achievable in each scenario analyzed.

III. Impact of Content Graph Structure

In this section we concentrate on a few aspects of the content graph structure and study their effect on the performance of the policies introduced before. We consider these properties with different content graphs, but also different network setups (e.g. cache sizes) for the same graph; as we will see later, such parameters can further affect the (absolute or relative) impact of content graph properties.

Understanding the impact of these parameters can be helpful for, given a content graph of interest, deciding whether it is worth running a computationally expensive but close-to-optimal algorithm over a quick and simple heuristic (see section IV on performance prediction), for designing the network setup to make the most of a particular policy (see Fig. 3 on the impact of N and C on different real-world traces), or to define the limits of performance achievable by any approach.

A fair question to ask at this point is *how do we characterize the structure of a graph?* The answer is not straightforward, since graphs are complex structures over which many metrics can be computed (see e.g. [15]). Furthermore, an exhaustive characterization (through the adjacency matrix, for example) is not insightful and impractical for large graphs. Here we have concentrated on a few macroscopic parameters that can be easily measured in any graph and that we could expect to have a significant impact on the performance of the joint caching and recommendation problem.

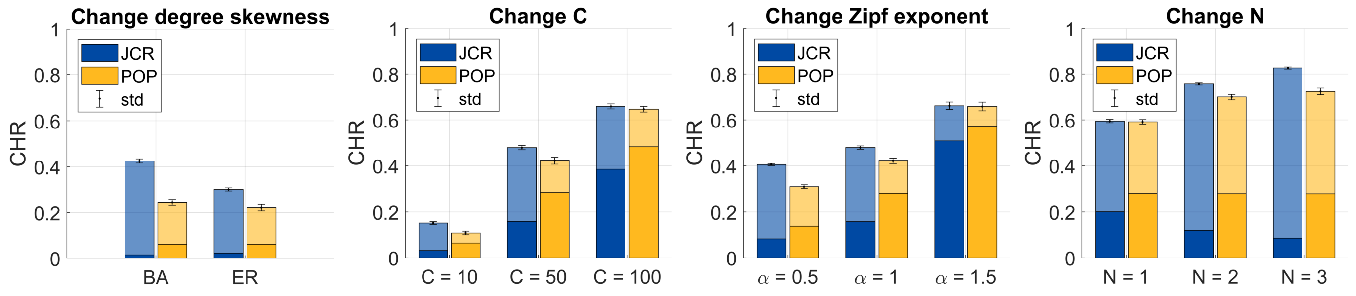

Fig. 1: Effect of degree skewness, cache size, popularity Zipf exponent and number of recommendations on performance.

In the first part of this section we perform experiments with synthetic data changing one parameter at a time to isolate their effect on the policies' performance as much as possible. In the second part we measure the graph parameter values of real-world traces and look at the performance of the two considered policies on them. We then attempt to interpret the latter results in light of the trends identified with synthetic data. Table II summarizes the notation used in this section, where the skewness $\gamma(X)$ of a random variable X is measured with Pearson's moment coefficient $\gamma(X) = E[((X - \mu_X)/\sigma_X)^3]$, and the clustering coefficient κ is measured as the number of triangles over the number of triples in the graph.

A. Synthetic Graphs

We tested the effect of both (i) content graph parameters: degree skewness, popularity skewness and community structure, and (ii) network setup parameters: cache size and number of recommendations. For our simulations with synthetic data we used three probabilistic models of graphs:

Erdős–Rényi (ER): a link between each pair of nodes is generated at random and independently with probability p_{link}. The resulting node degree is a random variable with binomial distribution, i.e. $R \sim B(K, p_{link})$.

Barabási–Albert (BA): starting from an initial graph of K_0 nodes, a new node is added by connecting it with $\ell_{new} \leq K_0$ new links to the nodes already in the graph. The probability of connecting to a node in the graph is proportional to its degree. The degree distribution of the resulting graph follows a power law distribution with exponent 3 [15].

Community graph: this is a disconnected graph where each community is connected and there are no links between different communities.

Next we describe the experiments done to test the impact of each graph parameter on the policies' performance and discuss the results. The parameter values used in each experiment are shown in Table III. In all cases w.l.o.g. we set $u_{ij} = 0.5$ if contents $i \neq j$ are related.[7] For all graphs, we assume a Zipf popularity distribution with tunable exponent α, and assign the popularities p_i to the graph nodes randomly. For each experiment we perform 32 repetitions of the graph generation and policy testing. We then report the average and standard deviation of the CHR accrued by each policy.

[7]For example, in the case of the traces where we only know which files are related but we do not know "how much", this 0.5 factor captures that the "value" of a soft cache hit (i.e. the click probability for the related item) is lower than a direct hit. As explained in Section II, the exact value of u_{ij} might depend on a number of factors, which go beyond the scope of this work. We observed empirically that its precise value (barred from the two extremes of 0 and 1) only affects the total CHR but not the qualitative impact of different graph parameters.

TABLE III: Parameter values used in each experiment with synthetic data

Feature tested	Degree skewness	C	Zipf exponent	N	Community structure
Graph	{ER,BA}	ER	ER	ER	{ER,comm}
K	1000	1000	1000	1000	1000
C	10	{10,50,100}	50	50	50
N	1	1	1	{1,2,3}	1
α	1	1	{0.5,1,1.5}	1	1
p_{link}	0.04	0.01	0.01	0.04	0.02
ℓ_{new}	10	-	-	-	-

1) Degree skewness: The first panel of figure 1 shows the performance of the two policies for both BA and ER graphs with $\ell_{new} = 10$ and $p_{link} = 0.04$, respectively. These values were selected so that the density $E[R]/K$ would be the same for both types of graphs, since a higher density would automatically increase the CHR thanks to more soft hits, and the comparison in that case would be unfair. The bars show the total CHR accrued, and the fraction obtained from direct and soft hits are shown in dark and light color respectively.

The CHR achieved by JCR on the BA graph is much larger than that of the ER graph. This happens because JCR is capable of picking the high-degree nodes in the BA graph to earn many soft hits. The ER, however, does not provide such a possibility, since all nodes have approximately the same degree and there is no clearly winning strategy that JCR can take. POP, on the other hand, performs almost identically for both graph types. This is because the popularity values were assigned to the nodes randomly, thus caching the most popular nodes is equivalent to picking up randomly and uniformly C nodes in the graph. Therefore the performance of POP depends on the average connectivity, and since $E[R]/K$ is the same for both graph types, POP achieves approximately the same CHR in both cases.

2) Cache size: The second panel shows the performance for different cache sizes for an ER graph. As expected, as the cache size increases both algorithms achieve a higher CHR. However, the high popularity skewness helps POP to boost its performance more than it helps JCR: by making the cache size only a 10% of the total catalog size, POP can achieve with its simple criterion a CHR ≈ 0.5 coming from direct hits and a total CHR comparable to that achieved by JCR. Note that if the popularity distribution was uniform ($\alpha = 0$) the amount of direct hits accrued by POP would equal C/K. The effect of the skewed distribution of p_i is discussed next.

3) Popularity skewness: The third panel shows the performance for different Zipf popularity exponents. Again, increasing this parameter helps both algorithms, but the boost is larger for POP. The reason is analogous to that of increasing C for fixed α: a larger percentage of requests can be served with direct hits, which is what POP goes after by caching the most popular contents. Note that as α increases, JCR tends to "copy" the strategy of POP and go for more direct hits.

4) Number of recommendations: The last panel shows the performance when we change N. The gap between JCR and POP increases with N in favor of the former. This is due to POP getting more soft hits just by chance, while JCR adjusts its strategy to exploit the larger number of recommendations. Note that the fractions of soft and direct hits of JCR increase and decrease respectively as N gets larger, and added together they achieve a higher total CHR each time.

5) Community structure: Figure 2 shows the performance for different cache sizes and two types of graphs: a community graph that contains 50 communities of 20 nodes all connected to their neighbors ("Cliques"), and an ER graph with $K = 1000$ and $E[R] = 20$. The strong community structure enhances the performance of JCR with respect to the ER case, and the effect is larger when the cache size matches the number of communities. This is because JCR can recognize that choosing one content per community is the best strategy. When C is smaller than the number of communities, the effectiveness of this strategy is limited by the cache size. When it is larger, JCR has to pick more than one node per community and the gains with respect to ER are more moderate. Again there are no significant effects on POP when the graph type changes, since popularity values are assigned at random and the density of both graphs is the same.

In the following experiments we measured the degree of community structure with two parameters: the clustering coefficient κ and Newman's modularity n [16]. For the latter we used the implementation of [17], which applies the method of [18] to find communities in the graph before computing n.

Overall, the results with synthetic data seem to indicate that there is indeed a direct relation between graph parameters and policy performance. But *can we observe similar tendencies in real-world data?* We address this question next.

B. Real-world traces

We tested the performance of JCR and POP in four real datasets from different applications and sources:

Amazon for Android applications (AznApp) [19]: here we considered that a pair of items were related when they were bought together.

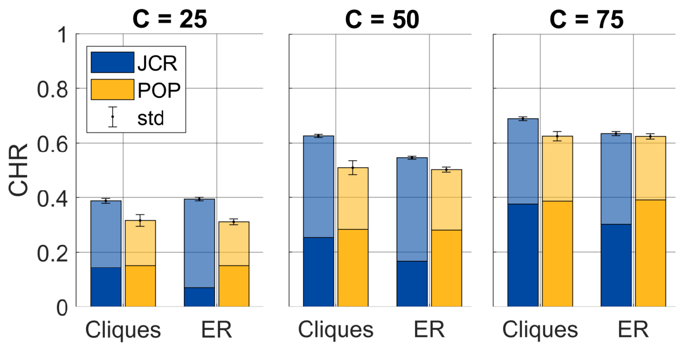

Fig. 2: Effect of community structure. Community structure boosts the performance of JCR respect to POP, provided that C approximately matches the number of communities.

LastFM [20]: the dataset contains for each song a list of similar songs, thus we considered songs i and j related when song i was in the list of j or vice-versa.

MovieLens (MovLens) [21]: for this dataset we built the content graph from the user ratings using collaborative filtering (see details of preprocessing in [9]).

YouTube [22]: as for LastFM, we set a link between two videos if either of them is in the recommended list of the other.

Like in the synthetic data, we set $u_{ij} = 0.5$. Since the Amazon and LastFM datasets did not contain the content popularity distribution, we generated random popularity values following a Zipf distribution with α = 1. Table IV shows the graph parameter values of all traces.

TABLE IV: Graph parameter values of the real-world traces. Notation is explained in Tables I and II.

	AznApp	LastFM	MovLens	YouTube
K	8229	3506	3306	2098
$E[R]$	15.97	4.25	25.49	5.38
$E[R]/K$	0.0019	0.0012	0.0077	0.0026
$\gamma(R)$	11.95	3.6	2.41	1.42
$\gamma(p)$	1.94	1.89	2.07	7.57
κ	0.08	0.13	0.55	0.38
n	0.59	0.81	0.78	0.84

Figure 3 shows the performance of both algorithms in the traces for different C/K ratios and values of N. Note that there is no standard deviation specified in these plots, since the traces are taken from real-world data and are thus unique.

As observed in synthetic data, increasing either C or N improves the performance of both algorithms. In the case of POP, increasing C is consistently better than increasing N, as could be expected. For JCR, however, *which* of the two parameters has the greatest impact to boost the performance in a trace *depends heavily on the characteristics of the content graph* (compare panels 2 and 3 against panel 1 in Fig. 3):

When the graph structure favors the accrual of soft hits, it is more beneficial to increase N***:*** This is true for AznApp, which has very high degree skewness, thus confirming the observations done with synthetic data. The MovLens dataset has high density and clustering coefficient, which also favors

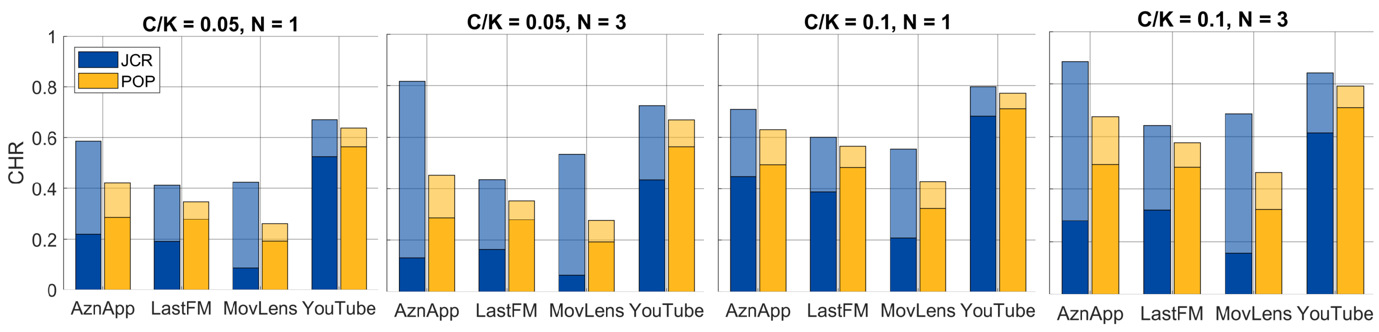

Fig. 3: Results on traces for different cache sizes and number of recommendations.

the accrual of soft hits. However, the structure of this trace seems not to be "as convenient" for JCR as that of AznApp to exploit soft hits, and increasing C achieves a similar total CHR as increasing N.

When the graph is not particularly well-connected or the popularity skewness is high, it is better to increase C: This happens for YouTube, which has a very high popularity skewness, and for LastFM, whose parameters are similar to those of MovLens but is more sparsely connected (it has significantly lower average dergee $E[R]$, density $E[R]/K$ and clustering coefficient κ). In these cases JCR can do better by accruing direct hits rather than soft ones, and thus increasing C is preferable over increasing N.

These observations suggest that, as observed with synthetic data, qualitative and quantitative performance differences in these much more complex traces can be, to some extent, attributed to structural differences of their respective content graphs. This motivates our next and final step in this work: to investigate whether these features can be used to predict the expected performance of the policies in a given dataset.

IV. Performance prediction

The results of the previous section with both synthetic and real-world data show that the performance of JCR and POP is heavily affected by the content graph structure and the network setup parameters. This suggests that looking at the parameters alone might suffice to design an automated performance predictor that, given a new graph (dataset), will be able to decide the benefits of exploiting the soft cache hits over just caching the most popular items, without actually running the optimization algorithm. Given the limited amount of training data (data catalogs and related content graphs), we choose in this work to perform a simple classification task: "will joint optimization (JCR) provide relative gains that exceed a threshold T, compared to the baseline (POP)?". We will attempt to answer this question using only the graph and network-related features introduced before, and Support Vector Machine (SVM) based classification.[8] We will also demonstrate how to generate a reasonably sized training set, through the use the collected data traces. Finally, we stress that the goal of this section is *not* to derive a state-of-the-art machine learning algorithm for this task, but rather to carry out a preliminary investigation of the feasibility of such a task.

[8]We also tried performing this binary classification using Logistic Regression, but SVM provided better results.

A. Dataset

We have chosen SVM methods as they can achieve reasonable performance even with smaller amounts of training data (compared e.g. to modern Deep Neural Network based methods). Nevertheless, our baseline consists of 6 collected traces, and 2 synthetic graph types. It is thus important to devise a methodology to come up with a proper training set out of these, that has diversity in the graph structure and not too many samples of a particular type of graph that could generate bias. Our complete dataset was constructed from:

Splitting traces: We randomly split the traces in groups of $500 \leq K \leq 700$ nodes (the average number of nodes in our traces was 4257), each of which constitutes a new graph of our dataset. Apart from the four traces used in Section III, here we also considered the Amazon Virtual Games (AznVG, $K = 5614$) and Amazon Movies & TV data (AznTV, $K = 2789$) datasets [19]. The adjacency values u_{ij} were obtained as those for AznApp.

Synthetic graphs: To add variability to the dataset and potentially improve generalization we generated six additional cases: three ER graphs with $p_{link} = 0.002, 0.005$ and 0.01 respectively, and three BA graphs with ℓ_{new} = 1, 4 and 8. The number of nodes of these graphs was chosen randomly and uniformly in [500, 700].

This procedure provided a total of 47 graphs (41 from real traces, 6 from synthetic data). We will refer to the set of graphs generated from a particular trace or synthetic graph as their "child set", e.g. out of the YouTube dataset we generated 36 YouTube child traces with this subsampling method. For every such child graph, we also create different network setups, as combinations of the following parameters: (i) 3 values of of Zipf popularity α = 0.5, 1 and 1.5, (ii) 3 values of C/K = 0.01, 0.05 and 0.1, and (iii) $N = 3$. This gives a total of 9 network setups per child graph, and thus a total of 423 different scenarios to consider for training (and testing).

B. Experiment design

Each case i in the dataset was represented by a vector $x_i \in \mathbb{R}^7$ with the values of parameters $\{K, E[R], \gamma(R), \kappa, n, \alpha, C\}$.

TABLE V: Distribution of labels per set ($y = 1/y = -1$).

Child set	$T = 0.1$	$T = 0.15$	**Child set**	$T = 0.1$	$T = 0.15$
AznVG	44 / 37	38 / 43	MovLens	28 / 17	20 / 24
AznApp	66 / 42	57 / 51	YouTube	13 / 23	10 / 26
AznTV	20 / 25	15 / 30	ER	12 / 15	10 / 17
LastFM	13 / 41	11 / 43	BA	21 / 6	18 / 9

We assigned a binary label y_i to each case according to:

$$y_i = \begin{cases} 1 & \text{if } \frac{\mathrm{CHR}_{\mathrm{JCR}}(i) - \mathrm{CHR}_{\mathrm{POP}}(i)}{\mathrm{CHR}_{\mathrm{JCR}}(i)} > T \\ -1 & \text{otherwise,} \end{cases}$$

where $T \in [0, 1]$ is a threshold for relative performance that we chose arbitrarily and the subindex of $\mathrm{CHR}_{\mathrm{P}}(i)$ indicates the CHR obtained by policy P when applied to case i. The choice of the threshold T defines the number of cases with each label. Table V shows the distribution of labels for each child set and the two thresholds $T = 0.1, 0.15$ used in our experiments. Note that the addition of the two numbers gives the total number of cases in the child set.

To test the generalization power of the obtained model, we trained using the child sets of 7 out of the 8 graph types considered, and used the remaining child set for testing. This resulted in 8 different training and testing experiments, each using a different child set as testing data. Such a splitting allowed us to make sure that the test set was *new unseen data, completely unrelated* from that used for training. Furthermore, to test the robustness of the results we repeated each of the 8 experiments 10 times. We report the mean and standard deviation of the training and testing accuracies in each experiment.

C. Support Vector Machines

The SVM is a non-probabilistic binary linear classifier method that finds the hyperplane in the feature space that better separates the two classes. The optimization problem solved by SVM (using L1 regularization) can be formulated as

$$\begin{aligned} \min_{\beta, \beta_0, \xi} \quad & \frac{1}{2} \|\beta\|^2 + \delta \sum_{i=1}^{m} \xi_i \\ \text{s.t.} \quad & y_i(\beta^T x_i + \beta_0) \geq 1 - \xi_i, \ i = 1, \ldots, m \\ & \xi_i \geq 0, \ i = 1, \ldots, m \end{aligned}$$

where m is the number of samples in the data (in our case graph-network setup combinations), $f(x) = \beta^T x_i + \beta_0$ defines the separating hypeplane between the two classes, ξ_i are slack variables that allow for having points on the wrong side of the boundary and δ is a cost parameter that penalizes having a large number of misclassified points [23].

For our experiments we normalized the features x_i by subtracting their mean and dividing by their standard deviation. For the training of the SVM we used the Matlab function `fitcsvm` and let the software optimize the hyperparameter δ through the option `OptimizeHyperparameters`.

D. Results of automatic performance prediction

Table VI shows the mean and standard deviation (between parenthesis) of both the training and testing accuracies over the 10 repetitions of the experiments and for two values of T. The distribution of labels ($y = 1/y = -1$) is shown next to each value of T. Next we make some remarks on these results.

TABLE VI: Accuracy of the SVM classifier for relative performance prediction.

Set used for testing	$T = 0.1$ (217/206)		$T = 0.15$ (180/243)	
	Test	**Train**	**Test**	**Train**
AznVG	0.93 (0.006)	0.94 (0.002)	0.96 (0.020)	0.93 (0.003)
AznApp	0.93 (0.012)	0.95 (0.001)	0.89 (0.035)	0.93 (0.005)
AznTV	0.96 (0.000)	0.95 (0.004)	0.86 (0.015)	0.93 (0.002)
LastFM	0.83 (0.020)	0.95 (0.004)	0.86 (0.049)	0.94 (0.007)
MovLens	0.84 (0.007)	0.95 (0.001)	0.91 (0.000)	0.93 (0.003)
YouTube	0.87 (0.037)	0.95 (0.002)	0.94 (0.000)	0.92 (0.003)
ER	0.96 (0.000)	0.95 (0.001)	0.83 (0.031)	0.93 (0.001)
BA	0.91 (0.031)	0.94 (0.001)	0.83 (0.019)	0.93 (0.001)
Mean	**0.90 (0.051)**	**0.95 (0.005)**	**0.89 (0.049)**	**0.93 (0.006)**

The features considered are good indicators to distinguish between high- and low-gain scenarios for the joint approach with respect to the baseline. This observation is based on the high training and testing accuracies observed in all experiments.[9] However, in some cases there is a large gap between the training and testing accuracies (differences larger than 5% in Table VI have been highlighted in purple), which leaves room for further improvement, discussed below.

More features or non-linear combinations of the current features might be needed when the generalization power changes with T. This comes from the fact that for some traces (AznTV, MovLens, YouTube, ER and BA) the training and testing accuracies are very similar for one value of T and quite different for the other. Thus, for one T the "separation rule" learned from the training data applies well to the test set (i.e. the hyperplane found makes a good split of the testing points), but when the labels change for the new T the rule learned at training does not apply to the test set anymore. For the former case a potential solution is enlarging the training set (discussed in the next point). Another alternative is considering more complex, non-linear interactions between the features: an example of such interplay was the link between the cache size and the number of communities identified in the experiment of section III-A5, where the gains of JCR due to strong community structure where maximized when both quantities were equal. Thus, adding new combinations of the features already considered may allow for capturing effects not well represented yet. Adding completely new features could help to this end as well, but contrary to what we want they might broaden the accuracy gap by introducing overfitting.

Enlarging the dataset with new, different and diverse graphs that enrich the feature space would probably improve generalization significantly. This would most likely help particularly the experiment using the LastFM child set for testing,

[9]In some cases the test accuracy is even slightly higher than the training accuracy. This can happen if the test data has relatively few samples and is not particularly challenging for the given threshold.

apart from being beneficial also for the cases mentioned in the previous paragraph. When testing with the LastFM child set the gap between training and testing accuracies is large for both values of T. This suggests that this test set might have specific characteristics not accounted for in the training set. In such case using a larger dataset with more varied content graphs that are potentially more similar to that of the LastFM child set could help to improve testing accuracy.

V. Discussion and Future Work

We have shown through our experiments with both synthetic and real-world data that we can associate specific graph properties to clear changes in the outcomes of the two policies considered. Such connection between graph parameters and policy performance reveal that there is a *fundamental limit* on the gains that *any* smart policy can achieve, compared to the baseline, that is intricately tied to the properties of the dataset (graph) itself, rather than the algorithm.

This observation motivated the exploration of whether we could do automatic performance prediction from a few graph structural descriptors. Our results with SVM classification support the hypothesis that with careful training of the classifier and a sufficiently representative dataset it is indeed possible to get good predictions. This not only gives further proof of the tight relation between graph properties and policy performance, but it also opens the possibility of using automatic classification for deciding whether applying a "smart" but computationally costly policy is worthy with respect to just using a simple baseline.

Finally, we remark that the analysis made here can also be applied to other settings considering the interplay between caching and recommendations. This can be done straightforwardly e.g. for [5]–[7], [9], [10], where the content catalog is also modeled as a graph. However, other cases may need an extra step to map their setup to our content graph formulation: in [8], [11], for example, the cosine distance used to measure user-content interest could instead be used to build the adjacency matrix of the content graph (which would be user-specific). In the case of [12], we could build U by setting each component u_{ij} to the inverse of the cost $C_a(i,j)$ of approximating content i with content j. In sum, our analysis does not only concern the JCR and POP policies, but could be readily applied to any other similar setting.

Future work will address the challenge of predicting the actual values of the individual and relative gains in CHR, instead of setting a hard threshold and perform binary classification as we did here. We will also consider adding features that could help devise effects not yet well captured by the ones analyzed in this work, and study more deeply the relevance of each parameter for predicting policy performance with sensitivity analysis and feature selection techniques.

Acknowledgements

This work was partially funded by the IMT F&R program and by the ANR-17-CE25-0001 Jeunes Chercheuses et Jeunes Chercheurs project 5C-for-5G.

References

[1] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1111–1125, 2018.

[2] P. Lyman and H. R. Varian, "How Much Information 2003." http://groups.ischool.berkeley.edu/archive/how-much-info-2003/. Retrieved on February 2020.

[3] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, "Tracing the path to youtube: A quantification of path lengths and latencies toward content caches," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 80–86, 2018.

[4] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[5] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, "A metric cache for similarity search," in *Proceedings of the 2008 ACM workshop on Large-Scale distributed systems for information retrieval*, pp. 43–50, 2008.

[6] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii, "Nearest-neighbor caching for content-match applications," in *Proceedings of the 18th international conference on World wide web*, pp. 441–450, 2009.

[7] F. Chierichetti, R. Kumar, and S. Vassilvitskii, "Similarity caching," in *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 127–136, 2009.

[8] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 125–138, 2018.

[9] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1300–1313, 2018.

[10] T. Giannakas, P. Sermpezis, and T. Spyropoulos, "Show me the cache: Optimizing cache-friendly recommendations for sequential content access," in *2018 IEEE 19th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 14–22, IEEE, 2018.

[11] Z. Lin and W. Chen, "Joint pushing and recommendation for susceptible users with time-varying connectivity," in *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2018.

[12] M. Garetto, E. Leonardi, and G. Neglia, "Similarity caching: Theory and algorithms," in *Proceedings of the IEEE Conference on Computer Communications (Infocom) 2020*, 2020.

[13] M. Costantini, T. Spyropoulos, T. Giannakas, and P. Sermpezis, "Approximation guarantees for the joint optimization of caching and recommendation," in *2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020.

[14] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet.," *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[15] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[16] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[17] A. Kehagias, "Community detection toolbox." https://www.mathworks.com/matlabcentral/fleexchange/45867-community-detection-toolbox), MATLAB Central File Exchange. Retrieved on November 2018.

[18] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[19] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, ACM, 2015.

[20] "Million song dataset." http://millionsongdataset.com/lastfm/.

[21] "Movielens dataset." https://grouplens.org/datasets/movielens/.

[22] "Dataset for statistics and social network of youtube videos." http://netsg.cs.sfu.ca/youtubedata/, 2008.

[23] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.