Optimal Content Caching and Recommendation with Age of Information

Ghafour Ahani and Di Yuan, *Senior Member, IEEE*

Abstract—Content caching at the network edge has been considered an effective way of mitigating backhaul load and improving user experience. Caching efficiency can be enhanced by content recommendation and by keeping the information fresh. To the best of our knowledge, there is no work that jointly takes into account these aspects. By content recommendation, a requested content that is not in the cache can be alternatively satisfied by a related cached content recommended by the system. Information freshness can be quantified by age of information (AoI). We address, optimal scheduling of cache updates for a time-slotted system accounting for content recommendation and AoI. For each content, there are requests that need to be satisfied, and there is a cost function capturing the freshness of information. We present the following contributions. First, we prove that the problem is NP-hard. Second, we derive an integer linear formulation, by which the optimal solution can be obtained for small-scale scenarios. Third, we develop an algorithm based on Lagrangian decomposition. Fourth, we develop efficient algorithms for solving the resulting subproblems. Our algorithm computes a bound that can be used to evaluate the performance of any suboptimal solution. Finally, we conduct simulations to show the effectiveness of our algorithm in comparison to a greedy schedule.

Index Terms—Age of information, caching, content recommendation, Scheduling

◆

1 INTRODUCTION

Content caching at network edge, such as base stations, is a promising solution to deal with the explosively increasing traffic demand and to improve user experience [1]. This approach is beneficial for both the users and the network operators as the former can access the content at a reduced latency, and latter can alleviate the load on backhaul links. The performance of edge caching, however, can be further improved by utilizing content recommendation and optimizing information freshness.

Originally, recommender systems have been used for presenting content items that best match user interests and preferences. In fact, the reports in [2], [3] show that 80% of requests on content distribution platforms are due to content recommendations. Recently, a number of studies have proposed to utilize content recommendation for improving caching efficiency. In [4], [5], recommendation is utilized to steer user requests toward the contents that are both stored in the cache and of interest to users. More recently, content recommendation is employed to satisfy content requests using alternative and related contents. Namely, instead of the initially requested content that is absent from the cache, some other related contents are recommended [6], [7], [8]. This approach is of interest to many applications such as video and image retrieval, and entertainment-based ones [6].

Another important aspect that arises naturally in the context of content caching is the freshness of information [9]. As cached contents may become obsolete with time, we need to also account for updating the content items. Information freshness is quantified by age of information (AoI) which is defined as the amount of time elapsed with respect to the time stamp of the information in the most recent update [10]. The AoI grows linearly between two successive updates.

In this study, we address optimal scheduling for updating the cache for a time-slotted system where content recommendation and AoI are jointly accounted for. The cache has a capacity limit, and the content items vary by size. Moreover, updating the cache in a time slot is subject to a network capacity limit. For a content request, if the content is available in the cache, the request is served using the stored content. Otherwise, a set of related and cached contents will be recommended. If one of the recommended contents is accepted, then the request will be again served from the cache with the accepted content. If not, the request will be served by the remote server with a higher cost. It is worth noting that incentive mechanisms may be utilized to motivate users to accept the recommended contents (e.g., zero-rating services) [6]. For each content item, there is a cost function that is monotonically increasing in the AoI. Thus, caching a content with higher AoI results in a higher cost.

The optimization decision consists of, the selection of the content items for updating the cache, and a recommendation set for each non-cached content. The objective is to find the schedule minimizing the total cost over the scheduling horizon.

Our work consists in the following contributions for the outlined cache optimization problem with recommendation and AoI (COPRA):

- We rigorously prove the NP-hardness of the problem, even when contents are of uniform size, based on a reduction from 3-satisfiability (3-SAT).
- We derive an integer linear programming (ILP) formulation for the problem in its general form, enabling the use of general-purpose optimization solvers to approach the problem. This is particularly useful for solving small-scale problem instances to optimality, and to enable to accurately evaluate low-complexity though sub-optimal

- *G. Ahani and D. Yuan are with the Department of Information Technology, Uppsala University, 751 05 Uppsala, Sweden (e-mails: {ghafour.ahani, di.yuan}@it.uu.se).*

algorithms.

- For problem solving, we apply Lagrangian decomposition to the ILP, allowing for decomposing the problem into two subproblems, each with special structures. The first subproblem itself further decomposes into smaller problems, each of which can be mapped to finding a shortest path in a graph. The second subproblem also decomposes to smaller problems. However, the problem size remains exponential, and therefore we propose column generation for gaining optimality. Moreover, we demonstrate that the pricing problem of column generation can be solved via dynamic programming (DP). It is also worth noting that, decomposition enables parallel computation. In addition, our algorithm computes a lower bound (LBD) that can be used to evaluate the quality of any given solution.

- Finally, we conduct extensive simulations to evaluate the performance of our algorithm by comparing its solution to global optimum for small-scale scenarios, and to the LBD otherwise. The evaluations show that our algorithm provides solutions within 8% of global optimality.

2 RELATED WORK

To the best of our knowledge there is no work that jointly study content caching, recommendation, and AoI. In the following, we first review the works that have studied content caching and AoI, and then those on caching and recommendation.

The works in [9], [11], [12], [13], [14], [15], [16], [17] have studied content caching when AoI is accounted for. The general problem setup in these works is what contents to cache and when to update them with an objective function based on AoI. In [9], [11] the objective is minimizing the expected AoI when inter-update intervals of each item or the total number of updates are known. In [12], [13], [14] content caching is studied where both popularity and AoI are considered. In [12], cache miss is minimized, and in [13] the load of backhaul link is minimized via balancing the AoI and cache updates. In [14], partially updating a content, which depends on the type of content and its AoI, is enough to completely update the content. In [15], the overall utility of a cache defined based on AoI of contents is maximized, subject to limited cache and backhaul link capacities. For a given origin, a set of users, and a (set of) cache between them, the AoI at the cache(s) and users is analyzed in [17]. As an extension of [17], the work in [16] studied the trade-off between obtaining a content from the origin with longer transmission time and from the cache with higher AoI. A recent survey of AoI can be found in [18].

In general, the works that studied content caching and recommendation can be classified into two categories. In the first category, content recommendation is utilized to shape the requests and steer the content demand toward the contents that are both stored in the cache and interesting to the users [4], [5], [19], [20], [21], [22], [23], [24], [25]. In [4], [5] a preference "distortion" tolerance measure is used to quantify how much the engineered recommendations distort the original user content preferences. In [19], an experiment is conducted to demonstrate the effect of content recommendation on caching efficiency in practice. In [20], the objective is to maximize both the quality of recommendation and streaming rate, and the authors proposed a polynomial-time algorithm with approximation guarantee. In [21], [22], caching and recommendation decisions are optimized based on the preference distribution of individual users. In [23], content caching and

recommendation are optimized taking into account the temporal-spatial variability of user requests. In [24], the authors study the fairness issues of recommendation where some contents get more visible than others by recommendation. In [25], reinforcement learning is utilized for learning user behavior and optimizing caching and recommendation.

In the second category of studies, recommendation is utilized to satisfy a request when the requested content is not available in the cache, by recommending some other cached and related contents [6], [7], [8], [26], [27], [28]. The idea of recommending related contents in case of a cache miss is formally introduced in [6] where the authors referred to the scenario as "soft cache hit". In this reference, the authors illustrate how "soft cache hit" is able to improve the caching performance. They also consider a caching problem with the objective of maximizing the cache hit rate where all contents in the cache can be recommended. Using the submodularity property of the objective function, they propose algorithms with performance guarantee. Later, in [8], the authors consider a more realistic system model in which only a limited number of contents can be recommended. Then, they propose a polynomial-time algorithm based on first solving the caching problem, and then finding the recommendations sets. In [26], the authors model the relation among contents as a graph, and then studied the characteristics of this graph to predict whether it is worth to find the optimal solution or a low complexity heuristic will be sufficient. In [27], the authors try to find the best caching policy for a sequence of requests where recommendation is accounted for. In [28] a multi-hop cache network is studied where soft cache hit is allowed in one of the caches along the path to the end node that stores the initially requested content.

The closest works to our study are [6], [8] in the sense that they also considered soft cache hits. However, there are significant differences. To the best our knowledge, it is novel that that caching decision, content recommendation, and information freshness are jointly optimized. Moreover, in our work we account for cache update costs, as well as the capacities of cache and backhaul links.

3 SYSTEM SCENARIO AND COMPLEXITY ANALYSIS

3.1 System Scenario

The system scenario consists of a content server, a base station (BS), and a set of content items $\mathcal{I} = \{1, 2, \ldots, I\}$. The server has all the contents, and the BS is equipped with a cache of capacity S. The BS is connected to the content server with a communication link of capacity L via which the cache contents can be updated. The size of content item $i \in \mathcal{I}$ is denoted by s_i.

We consider a time-slotted system with a time period of T time slots, denoted by $\mathcal{T} = \{1, 2, \ldots, T\}$. At the beginning of each time slot, the contents of the cache are subject to updates. Namely, some stored contents may be removed from the cache, some new contents may be added to the cache by downloading from the server, and some existing contents may be refreshed.

The AoI of an item in the cache is the time difference between the current slot and the time slot in which the item was most recently downloaded to the cache. Each time an item is downloaded to the cache, the item's AoI is zero, i.e., maximum information freshness. The AoI then increases by one for each time slot, until the next update. In other words, the AoI of any cached content item is linear in time, if the content is not refreshed. For content $i \in \mathcal{I}$, the relevant AoI has a limit A_i. The content is considered

obsolete if the AoI exceeds A_i. Hence, a cached content i in time slot t can take one of the AoIs in $\mathcal{A}_{ti} = \{0, ..., \min(A_i, t-1)\}$. The cost associated with content item i with AoI a in time slot t is characterized by a cost function f_{tia} that is monotonically increasing in AoI a.

For a request of content i, if the content is stored in the cache and the AoI is no more than A_i, the request is satisfied from the cache. Otherwise, a set of related cached contents, hereinafter referred to as a recommendation set, is recommended to the user. If the user accepts any element of the recommendation set, the request is satisfied by the cache. If not, the request needs to be satisfied from the server. Note that since a user may not be interested in getting a long list of recommended contents, we limit the size of recommendation set to be at most N [8], [29]. Denote by $\mathcal{R}_i = \{1, 2, ..., R_i\}$ the index set of all contents related to content i. This set can be determined from past statistics and/or learning algorithms [6]. Obviously, the index set of any recommendation set for content i is a subset of $\mathcal{R}_i$. Note that the recommendation set may change from a time slot to another.

Denote by h_{ti} the number of requests for content $i \in \mathcal{I}$ in time slot $t \in \mathcal{T}$. The value of h_{ti} can be estimated via recent requests of the contents, popularity of the contents, and/or machine learning algorithms [6], [30]. In this study, for the ease of exposition, we consider the total number of requests for a content instead of individual user requests. Similarly, the acceptance probability of a content does not vary from a user to another. Note that individual user requests and acceptance probability can be easily accommodated in our formulations and algorithms. Denote by c_s and c_b the costs for downloading one unit of data from the server and from the cache to a user, respectively. Downloading cost from server to cache is $c_s - c_b$. Intuitively, $c_s > c_b$ to encourage downloading from the cache.

The cache optimization problem with recommendation and AoI, or COPRA in short, is to determine which content items to store, update, and recommend in each time slot, such that the total cost of content requests over time horizon $1, 2, ..., T$ is minimized, subject to cache and backhaul link capacities.

3.2 Cost Model

Denote by x_{tia} a binary optimization variable that equals one if and only if content i with AoI a is stored in the cache at time slot t. Hence, $x_{ti0} = 1$ means that the content i at time slot t is just downloaded from the server to the cache with AoI zero. Then, the overall downloading cost is shown in (1). In (1), the first term is the downloading cost from server to the cache due to cache updates and the second term is the downloading cost of requests that are delivered using cached contents.

$$\Delta_{download} = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} \left((c_s - c_b) s_i x_{ti0} + \sum_{a \in \mathcal{A}_{ti}} c_b s_i h_{ti} f_{tia} x_{tia} \right) \tag{1}$$

Next, we calculate the downloading cost related to content recommendation. Denote by p_{ija} the probability of accepting content $j \in \mathcal{R}_i$ with AoI a instead of content i. This probability depends both on the correlation between the two contents as well as the AoI of content j. The value of p_{ija} can be calculated based on historical statistics [6], item-item recommendation [31], and/or collaborative filtering techniques [32]. Denote by c a generic candidate set of contents for recommendation. Because of AoI, each element of c is a tuple of a recommended content and its

AoI. We refer to c as the recommendation set. Denote by $\mathcal{C}_{ti}$ the set of all such recommendation sets for content $i \in \mathcal{I}$ in time slot t. Denote by v_{tic} a binary optimization variable that takes value one if and only if (some content) in recommendation set $c \in \mathcal{C}_{ti}$ is accepted instead of content i in time slot t. The probability of not accepting any of the contents in c is $\tilde{P}_{ic} = \prod_{(j,a) \in c} (1 - p_{ija})$. Thus, the probability of accepting at least one of them is $1 - \tilde{P}_{ic}$, and hence the expected cost[1] is:

$$\Delta_{recom} = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} \sum_{c \in \mathcal{C}_{ti}} \left(c_b(1 - \tilde{P}_{ic}) + c_s \tilde{P}_{ic} \right) s_i h_{ti} v_{tic} \tag{2}$$

Finally, the total cost of system is the sum of $\Delta_{download}$ and Δ_{recom}.

3.3 Problem Formulation

COPRA can be formulated using integer-linear programming (ILP), as shown in (3). In (3), we use y_{ti} as an auxiliary binary variable that equals one if and only if content item i is cached in time slot t. Constraints (7b) state that if content i is cached in time slot t, then it should exactly take one of the possible AoIs a in $\mathcal{A}_{ti}$. Constraints (7c) and (7d) together ensure that content i in time slot t has AoI a (i.e., $x_{ita} = 1$) if and only if three conditions hold: Item i is in the cache ($y_{ti} = 1$), it has AoI $a - 1$ in time slot $t - 1$ ($x_{i(t-1)(a-1)} = 1$), and it is not refreshed again in slot t ($x_{it0} = 0$). Constraints (3e) indicate that either content i is cached in time slot t or some set $c \in \mathcal{C}_{ti}$ is recommended. Constraints (3f) ensure that the contents in recommendation set c are indeed cached. Constraints (3h) and (3g) formulate the cache and backhaul capacities. Finally, Constraints (3i)-(3k) state the variable domain.

$$\text{ILP} : \min_{\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{v}} \quad \Delta_{download} + \Delta_{recom} \tag{3a}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}_{ti}} x_{tia} = y_{ti}, t \in \mathcal{T}, i \in \mathcal{I} \tag{3b}$$

$$x_{tia} \geq y_{ti} + x_{(t-1)i(a-1)} - x_{ti0} - 1, \\ t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \tag{3c}$$

$$x_{tia} \leq x_{(t-1)i(a-1)}, \\ t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \tag{3d}$$

$$\sum_{c \in \mathcal{C}_{ti}} v_{tic} + y_{ti} = 1, t \in \mathcal{T}, i \in \mathcal{I} \tag{3e}$$

$$\sum_{c \in \mathcal{C}_{ti} : (j,a) \in c} v_{tic} \leq x_{tja}, t \in \mathcal{T}, i \in \mathcal{I}, j \in \mathcal{R}_i \tag{3f}$$

$$\sum_{i \in \mathcal{I}} s_i y_{ti} \leq S, t \in \mathcal{T} \tag{3g}$$

$$\sum_{i \in \mathcal{I}} s_i x_{ti0} \leq L, t \in \mathcal{T} \tag{3h}$$

$$y_{ti} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I} \tag{3i}$$

$$x_{tia} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \tag{3j}$$

$$v_{tic} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I}, c \in \mathcal{C}_{ti} \tag{3k}$$

As the number of recommendations set are exponentially many, the ILP is exponential in size. However, the ILP is of

---

1. In this cost, $1 - \tilde{P}_{ic}$ is multiplied by the size of initially requested content i.e., s_i. The reason is that we consider recommending only contents with similar size to s_i.

interest for solving small-scale problem instances for gauging the performance of other suboptimal solutions.

4 COMPLEXITY ANALYSIS

In this section, we rigorously prove the NP-hardness of COPRA based on a reduction from the 3-SAT. Next, we show the tractability of the problem for a single time slot when the contents are partitioned into subcategories with uniform probabilities.

Theorem 1. *COPRA is NP-hard.*

Proof. We adopt a polynomial-time reduction from the 3-SAT problem that is NP-complete [33]. Consider any 3-SAT instance with k clauses and n Boolean variables $u_1, u_2, ..., u_n$. A variable or its negation is called a literal. Denote by $\hat{u}_i$ the negation of u_i, $i = 1, 2, ..., n$. Each clause consists of a disjunction of exactly three different literals, for example, $\hat{u}_1 \vee u_5 \vee u_7$. The task is to determine if there is an assignment of true/false values to the variables, such that all clauses are satisfied (i.e., at least one literal has value true in every clause).

We construct a reduction from 3-SAT as follows. Each literal or clause represents a content, referred to as literal and clause contents, respectively. Moreover, n auxiliary contents are defined, one for each pair of variable and its negation. Hence, there are in total $3n + k$ contents, and $\mathcal{I} = \{1, 2, ..., 3n + k\}$. All contents have unit size, i.e., $s_i = 1$ for $i \in \mathcal{I}$. Each variable, its negation, and the corresponding auxiliary content are related mutually with acceptance probability of 1. Thus, the requests made for any of them can be fully satisfied by any of the other two contents. The number of time slots is one, i.e., $\mathcal{T} = \{1\}$, and the size of cache is n, i.e., $S = n$. The number of requests for each clause and literal content is 1, i.e., $h_{1i} = 1$ if i is a literal or a clause content. Each clause content is related to the corresponding three literal contents with acceptance probability of 1. Hence for a request made for a clause content, the system can recommend the three literals if some or all of them are cached. There are $n + k + 1$ requests for each auxiliary content, i.e., $h_{1i} = n + k + 1$ if i is an auxiliary content. No relation is present between contents other than those specified above. Note that the acceptance probability is symmetric between any two related contents.

Parameters c_b and c_s are set to 1 and 2, respectively. We now show there is an optimal solution such that the cache stores exactly either a variable or its negation. Suppose an auxiliary and/or a clause content is cached. In the former case, swapping this auxiliary content with a non-cached literal content of the corresponding pair will not increase but possibly improve the cost. Because, by swapping, more clause contents may also be satisfied from the cache. Now, suppose a clause content is cached. Then, at least one auxiliary content must be served using the server with cost $2(n+k+1)$. For the other contents, the best possible outcome is $(n - 1)(n + k + 1) + 2n + k + 1$. Hence, the total cost is $\Delta_1 = 2(n + k + 1) + (n - 1)(n + k + 1) + 2n + k + 1$. The cost when exactly one literal of each literal pair is cached is no more than $\Delta_2 = 3n + n(n + k + 1) + 2k$, assuming all clause contents are served using the server. It can be verified easily that $\Delta_1 > \Delta_2$. Therefore, at an optimum, the cache stores exactly either a variable or its negation. Thus the optimal total cost for the literal and auxiliary contents is known.

Clearly the construction above is polynomial in size. If there is no solution to the 3-SAT, then at least one clause content need to be downloaded from server with cost $c_s = 2$, and each of the

other clause contents has at least the cost of $c_b = 1$. Thus, the total cost is at least $\delta_1 = k + 1$. If there is a solution to 3-SAT, then the cost for all clause contents is at most $\delta_2 = k$. As can be seen $\delta_1 > \delta_2$. Thus, whether or not there exists a caching strategy with a total cost of no more than δ_2 gives the correct answer to 3-SAT. Therefore, the recognition versions of COPRA is NP-complete and its optimization version is NP-hard. $\square$

In practice, the content items may naturally fall into different subcategories based on the type of the content, e.g., video contents can be catagorized based on if it is science fiction, drama, or comedy, etc., [21], [22]. If all items in a subcategory are related with the same acceptance probability, then we show the optimal solution of the problem with uniform size and one time slot can be computed in polynomial time via DP. Note that the probability from a subcategory to another may still differ. We refer to this special case as COPRA-CAT.

Theorem 2. *COPRA-CAT can be solved in polynomial time.*

Proof. We compute a matrix, called cost matrix and denote it by $\boldsymbol{g}$, in which entry $g(k, i)$ represents the total cost by caching i content items of category k. This value is computed simply from the first i contents with the highest requests. Below, a recursive function is introduced to derive the optimal caching solution over all categories. We define a second matrix, called the optimal cost matrix, and denote it by $\boldsymbol{w}$, in which $w(k, s')$ represents the cost of the optimal solution by considering the first k categories using a cache size of s', $s' = 0, 1, ..., S$. The value of $w(q, s')$ is computed by the following recursion:

$$w(k, s') = \min_{r = 0, 1, ..., s'} \{g(k, r) + w(k - 1, s' - r)\} \quad (4)$$

Using Equation (4), the optimal cost for the first k categories is computed given the optimal cost of the first $k - 1$ categories. For the overall solution, the optimal cost can be computed using the above recursion for cache size S and K categories. We prove it by induction. First, when $k = 1$, i.e., we have only one category, We have $w(1, s') = \min_r g(1, r)$ for all $r = 0, 1, ..., s'$. Obviously $r^* = s'$, that is, to allocate the whole capacity to this category. Now, assume $w(l, s')$ is optimal for some l. We prove that $w(l + 1, s')$ is optimal. According to the recursive function:

$$w(l + 1, s') = \min_r \{g(l + 1, r) + w(l, s' - r)\} \quad (5)$$

The possible values for $r = 0, 1, ..., s'$, and for each of the possible values of r, $w(l, s' - r)$ is optimal. This together gives the conclusion that the minimum will be obtained indeed by the min operation. Thus, $w(l + 1, s')$ is optimal.

Finally, we show that $w(K, S)$ can be computed in polynomial time. The complexity of computing g is of $O(KI)$. By the above, the computational complexity of $\boldsymbol{w}$ is of $O(KS^2)$ where S is up to the number of contents. $\square$

5 GREEDY ALGORITHM

A commonly considered strategy for fast but suboptimal solution is a greedy approach (GA) that builds up a solution incrementally. GA tries to minimize the total cost of each time slot by considering the items one by one. The algorithm is shown in Algorithm 1.

For each time slot and each item, GA calculates an overall score based on the number of requests, the relations to other contents, and the size of the content, see Line 7. Then, GA treats

items based on their scores in descending order. For a content under processing, it is downloaded from server to the cache if there is enough cache and backhaul capacities, see Lines 11-13. Otherwise, GA checks if the content is cached in the previous time slot, and if there is enough cache capacity to store the content, see Lines 14-17. When all contents are processed, GA finds recommendation sets for the non-cached items. For each non-cached item, GA looks at the cached and related items, and pick the ones of highest acceptance probabilities, see Lines 18-21. GA is simple but it turns out the performance is not satisfactory, and therefore there is a need of developing a better algorithm.

Algorithm 1 Greedy Algorithm

Input: $\mathcal{T}, S, L, \boldsymbol{p}, \boldsymbol{h}, A_i, s_i, i \in \mathcal{I}$
Output: $\boldsymbol{y}^*, \boldsymbol{x}^*, \boldsymbol{v}^*$

1: $\boldsymbol{y} \leftarrow \{0, t \in \mathcal{T} \cup \{0\}, i \in \mathcal{I}\}$
2: $\boldsymbol{x} \leftarrow \{0, t \in \mathcal{T}, i \in \mathcal{I}, a \in \mathcal{A}_{ti}\}$
3: $\mathcal{C}'_{ti} \leftarrow \emptyset, t \in \mathcal{T}, i \in \mathcal{I}$
4: $\boldsymbol{v} \leftarrow \{0, t \in \mathcal{T}, i \in \mathcal{I}, c \in \mathcal{C}'_{ti}\}$
5: **for** $t \in \mathcal{T}$ **do**
6:    $\mathcal{I}' \leftarrow \mathcal{I}, L' \leftarrow L, S' \leftarrow S$
7:    $\theta_{ti} \leftarrow (\sum_{j \in \mathcal{I}} p_{ji0} h_{tj})/s_i, i \in \mathcal{I}$
8:    **while** $\mathcal{I}' \neq \emptyset$ and $S' > 0$ **do**
9:       $i^* \leftarrow \arg\max_{i \in \mathcal{I}'} \theta_{ti}$
10:      $\mathcal{I}' \leftarrow \mathcal{I}' \setminus \{i^*\}$
11:      **if** $(s_{i^*} \leq L'$ and $s_{i^*} \leq S')$ **then**
12:         $L' \leftarrow L' - s_{i^*}, S' \leftarrow S' - s_{i^*}$
13:         $y_{ti^*} \leftarrow 1, x_{ti^*0} \leftarrow 1$
14:      **else if** $(y_{(t-1)i^*} = 1$ and $s_{i^*} \leq S'$ and $x_{ti^* A_i^*} \neq 1)$
       then
15:         $S' \leftarrow S' - s_{i^*}$
16:         $a^* \leftarrow \arg\max_a x_{(t-1)i^* a}$
17:         $y_{ti^*} \leftarrow 1, x_{ti^*(a^*+1)} \leftarrow 1$
18:    **for** $i \in \mathcal{I}'$ **do**
19:      $c \leftarrow \{(j, a) : x_{tja} = 1, j \in \mathcal{R}_i, a \in \mathcal{A}_{tj}\}$
20:      $\mathcal{C}'_{ti} \leftarrow$ {the first N elements in c with the highest
       probabilities
       with respect to i}
21:      $v_{tic} \leftarrow 1$

---

6 ALGORITHM DESIGN

We propose an algorithm by applying Lagrangian decomposition (LD) to ILP (3). In LD, some variables are duplicated, with equalities constraints requiring that the duplicates are equal to the original variables. Next, these constraints are relaxed using Lagrangian relaxation and some method (often a subgradient method [34], [35]) is applied to solve resulting Lagrangian dual.

6.1 Lagrangian Decomposition

In our LD-based algorithm (LDA), we duplicate the $\boldsymbol{x}$ variables. Specifically, we replace $\boldsymbol{x}$ variables in AoI constraints (3b)-(3d) by $\boldsymbol{x}'$ and add a set of constraints requiring that $\boldsymbol{x} = \boldsymbol{x}'$. Next, we relax constraints $\boldsymbol{x} = \boldsymbol{x}'$ with multipliers $\boldsymbol{\lambda}$, and the resulting Lagrangian relaxation is given in (6). Note that $\Delta'_{download}$ is the same as $\Delta_{download}$ but the x variables are replaced by x'.

As can be seen ILP (6) is decomposed into to two subproblems, one consists of all terms containing $\boldsymbol{x}'$, and the other all terms with $\boldsymbol{x}$. Below, we formally state each of them.

$$\min_{\boldsymbol{y},\boldsymbol{x},\boldsymbol{x}',\boldsymbol{v} \in \{0,1\}} \Delta'_{download} + \Delta_{recom} +$$
$$\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}_{ti}} \lambda_{tia}(x'_{tia} - x_{tia}) \quad (6a)$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}_{ti}}^{t-1} x'_{tia} = y_{ti}, t \in \mathcal{T}, i \in \mathcal{I} \quad (6b)$$

$$x'_{tia} \geq y_{ti} + x'_{(t-1)i(a-1)} - x'_{ti0} - 1,$$
$$t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \quad (6c)$$

$$x'_{tia} \leq x'_{(t-1)i(a-1)}, t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \quad (6d)$$

$$(3e) - (3h)$$

6.2 Subproblem One

Subproblem 1, hereinafter referred to as SP_1, is shown in (7). The SP_1 consists of all terms having $\boldsymbol{x}'$, namely the downloading cost and Lagrangian multiplier terms as the objective function, and constraints related to AoI.

We exploit the structure of SP_1 as follows. First, as there is no constraint bundling the content items together, SP_1 decomposes by content, leading to I smaller problems. The optimization problem corresponding to content $i \in \mathcal{I}$ is denoted by $\text{SP}_1^{(i)}$ and consists of the terms of SP_1 for content i. Second, we show that $\text{SP}_1^{(i)}, i \in \mathcal{I}$, can be solved as a shortest path problem.

Theorem 3. $\text{SP}_1^{(i)}, i \in \mathcal{I}$, can be solved in polynomial time as a shortest path problem.

Proof. Consider content $i \in \mathcal{I}$. We construct an acyclic directed graph such that finding the shortest path from the origin to distention is equivalent to solving $\text{SP}_1^{(i)}$. The graph is shown in Figure 1. The objective function of $\text{SP}_1^{(i)}$ is:

$$\sum_{t \in \mathcal{T}} \left((c_s - c_b) s_i x'_{ti0} + \sum_{a \in \mathcal{A}_{ti}} c_b s_i h_{ti} f_{tia} x'_{tia} \right) + \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}_{ti}} \lambda_{tia} x'_{tia}$$
$$= \sum_{t \in \mathcal{T}} (c_s - c_b) s_i x'_{ti0} + \sum_{t \in \mathcal{T}} \sum_{a \in \mathcal{A}_{ti}} (c_b s_i h_{ti} f_{tia} + \lambda_{tia}) x'_{tia}$$
$$(8)$$

The graph is constructed as follows. Nodes O and D are used to represent the origin and destination respectively. For time slot t, there are $1 + \min\{A_i, t - 1\}$ vertically aligned nodes. A path passing node V_{ti}^0 and V_{tia}^1 corresponds to the following two scenarios, respectively:

1) The content is not in the cache.
2) The content is in the cache and has AoI $a, a \in \mathcal{A}_{ti}$.

For each node V_{ti}^0 there are two outgoing arcs, one to $V_{(t+1)i}^0$ which corresponds to that the content is not stored in the next time slot and the arc hence has weight zero, and the other to $V_{(t+1)i0}^1$ which has weight $d_{(t+1)i0} = (c_s - c_b)s_i + c_b s_i h_{(t+1)i} f_{(t+1)i0} + \lambda_{(t+1)i0}$ corresponding to the case that the content is downloaded to the cache in the next time slot and has AoI zero. For each node V_{tia}^1 there three outgoing arcs to $V_{(t+1)i}^0$, $V_{(t+1)i0}^1$, and $V_{(t+1)i(a+1)}^0$, respectively. A path passing through the first, second, and the third arcs corresponds to the following three scenarios, respectively:

1) Content is deleted for the next time slot with arc weight zero.

Fig. 1. Graph of the shortest path for $\mathrm{SP}_1^{(i)}$ corresponding to content item i.

$$\mathrm{SP}_1 : \min_{\boldsymbol{y}, \boldsymbol{x}' \in \{0,1\}} \Delta'_{download} + \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}_{ti}} \lambda_{tia} x'_{tia} \tag{7a}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}_{ti}} x'_{tia} = y_{ti}, t \in \mathcal{T}, i \in \mathcal{I} \tag{7b}$$

$$x'_{tia} \geq y_{ti} + x'_{(t-1)i(a-1)} - x'_{ti0} - 1,$$
$$t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \tag{7c}$$

$$x'_{tia} \leq x'_{(t-1)i(a-1)},$$
$$t \in \mathcal{T} \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \tag{7d}$$

2) The content is re-downloaded from the cache and has AoI zero with weight $d_{(t+1)i0}$.
3) The content is kept and its AoI increases with one time unit and has weight $d_{(t+1)i(a+1)} = c_b s_i h_{(t+1)i} f_{(t+1)i(a+1)} + \lambda_{(t+1)i(a+1)}$.

Finally, there are T arcs from V_{Ti0}^0 and V_{Tia}^1 to D, each with weight zero.

Given any solution of $\mathrm{SP}_i^{(i)}$, by construction of the graph, the solution directly maps to a path from the origin to the destination with the same objective function. Conversely, given a path we construct an ILP solution. For time slot t, if the path contain node V_{ti}^0 and V_{tia}^1, we set $y_{ti} = 0$. If the path passes through node V_{tia}^1, we set $y_{ti} = 1$ and $x'_{tia} = 1$. The resulting ILP solution has the same objective function value as the path length in terms of the arc weights. Hence the conclusion. $\square$

6.3 Subproblem Two

Subproblem 2, hereinafter referred to as SP_2, consists of all those terms of (6) containing $\boldsymbol{x}$. SP_2 decomposes by time slot, leading to T smaller problems. Denote by $\mathrm{SP}_2^{(t)}$ the problem corresponding to time slot t, shown in (9).

The number of v variables in $\mathrm{SP}_2^{(t)}$ is exponentially many, as there are exponential number of recommendation sets. Hence, having all v variables in the ILP is impractical. To deal with this issue, we apply column generation to the $\boldsymbol{v}$ variables in the LP relaxation of (9), to generate only the promising recommendation sets. Column generation is a powerful method to obtain the global optimum of some structured linear programs with exponential number of variables. In a column generation algorithm, the most promising variables are generated in a iterative process by solving alternatively a master problem (MP) and a pricing problem (PP). Each time PP is solved, a new variable that possibly improves the objective function is generated. The benefit of column generation

is to exploit the fact that at optimum only a few variables are positive. Below we define the MP and PP for solving $\mathrm{SP}_2^{(t)}$. In the following, to ease the presentation, we consider a generic time slot and drop the index t.

$$\mathrm{SP}_2^{(t)} : \min \sum_{i \in \mathcal{I}} \sum_{c \in \mathcal{C}_i} \left(c_b (1 - \tilde{P}_{ic}) + c_s \tilde{P}_{ic} \right) s_i h_{ti} v_{tic} \tag{9a}$$

$$- \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}_{ti}} \lambda_{tia} x_{tia} \tag{9b}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}_{ti}} x_{tia} = y_{ti}, i \in \mathcal{I} \tag{9c}$$

$$\sum_{c \in \mathcal{C}_i} v_{tic} + y_{ti} = 1, i \in \mathcal{I} \tag{9d}$$

$$\sum_{c \in \mathcal{C}_i : (j,a) \in c} v_{tic} \leq x_{tja}, , i \in \mathcal{I}, j \in \mathcal{R}_i \tag{9e}$$

$$\sum_{i \in \mathcal{I}} s_i y_{ti} \leq C \tag{9f}$$

$$\sum_{i \in \mathcal{I}} s_i x_{ti0} \leq L \tag{9g}$$

$$y_{ti} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I} \tag{9h}$$

$$x_{tia} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \tag{9i}$$

$$v_{tic} \in \{0, 1\}, t \in \mathcal{T}, i \in \mathcal{I}, c \in \mathcal{C}_{ti} \tag{9j}$$

6.3.1 MP and RMP

MP is the continuous version of (9). Restricted MP (RMP) is the MP but with a small subset $\mathcal{C}'_i \subseteq \mathcal{C}_i$ for any content $i \in \mathcal{I}$. Denote by C'_i the cardinality of $\mathcal{C}'_i$.

6.3.2 Pricing problem

The PP uses the dual information to generate new variables/columns. Denote by $\boldsymbol{v}^*$ the optimal solution of RMP. After obtaining $\boldsymbol{v}^*$, we need to check whether $\boldsymbol{v}^*$ is the global optimum of RMP. This can be determined by finding a column with the minimum reduced cost for each content $i \in \mathcal{I}$. This means the PP decomposes to I smaller problems, one corresponding to each content i. If all these minimum reduced cost values are nonnegative, the current solution is optimal. Otherwise, we add the columns with negative reduced costs to their recommendation sets.

Consider content $i \in \mathcal{I}$. Denote by π_i^* and $\boldsymbol{\beta}_i^* = \{\beta_{ija} | j \in \mathcal{R}_i, a \in \mathcal{A}_i\}$ the optimal dual values of the counterpart constraints

of (9d) and (9e) in the RMP, respectively. Hence, the reduced cost of the v-variable of content i and recommendation set c is:

$$\left(c_b(1-\tilde{P}_{ic})+c_s\tilde{P}_{ic}\right)s_ih_i-\pi_i^*+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\beta_{ija}^*=$$
$$\left(s_ih_i(c_s-c_b)\tilde{P}_{ic}\right)+c_bs_ih_i-\pi_i^*+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\beta_{ija}^* \quad (10)$$

in which $\tilde{P}_{ic}=\prod_{(j,a)\in c}(1-p_{ija})$. This reduced cost is nonlinear due to the term $\tilde{P}_{ic}$. But, we can linearize it using logarithm. Let

$$p'=\log\left(h_is_i(c_s-c_b)\prod_{(j,a)\in c}(1-p_{ija})\right)$$
$$=\log\left(h_is_i(c_s-c_b)\right)+\sum_{(j,a)\in c}\log(1-p_{ija}) \quad (11)$$

Now, the reduced cost can expressed as:

$$10^{p'}+c_bs_ih_i-\pi_i^*+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\beta_{ija}^* \quad (12)$$

where $p'=\log\left(h_is_i(c_s-c_b)\right)+\sum_{(j,a)\in c}\log(1-p_{ija})$. As $p_{ija}\in(0,1)$, $\sum_{(j,a)\in c}\log(1-p_{ija})$ is zero or a negative value. Thus, the minimum and maximum values that p' can take are $p'_{min}=\log\left(h_is_i(c_s-c_b)\right)+\sum_{(j,a)\in c}\log(1-p_{ija})$. and $p'_{max}=\log\left(h_i(c_s-c_b)\right)$, respectively. Hence $p'\in[p_{min},p_{max}]$. The above expression is for a given v-variable. In the following, we define PP, that is an auxiliary problem, of which the optimum will tell us the not-yet-present variable with minimum reduced cost.

Denote by $\mathrm{PP}^{(i)}$ the PP corresponding to content i. Let z_{ja} be a binary optimization variable that takes value one if and only if content j with AoI a is in the set to be generated. Then $\mathrm{PP}^{(i)}$ can be expressed as (13). Note that the terms $c_bs_ih_i$ and π_i^* are constants here, and hence can be dropped in the optimization process. Constraints (13c) ensure that for each content in the recommendation set, exactly one AoI value is selected. Constraint (13d) states that the total number of contents in the recommendation set can not exceed the given upper bound. In the following, we show that $\mathrm{PP}^{(i)}$ can be solved via DP.

$$\mathrm{PP}^{(i)}:\min 10^{p'}+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\beta_{ija}^*z_{ja} \quad (13a)$$

$$\text{s.t.}\quad p'=\log\left(h_is_i(c_s-c_b)\right)+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\log(1-p_{ija})z_{ja} \quad (13b)$$

$$\sum_{a\in\mathcal{A}_j}z_{ja}\leq 1, j\in\mathcal{R}_i \quad (13c)$$

$$\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}z_{ja}\leq N \quad (13d)$$

$$z_{ja}\in\{0,1\}, j\in\mathcal{R}_i, a\in\mathcal{A}_j \quad (13e)$$

$$p'\in[p_{min},p_{max}] \quad (13f)$$

Theorem 4. *$PP^{(i)}$ can be solved to any desired accuracy via DP.*

Proof. We first perform two prepossessing steps, and then apply DP to the resulting problem. First, as the objective function is minimization and p' is a continuous variable, constraint (13b) can

be stated equivalently as a greater-than-or-equal constraint. We re-express the constraint as:

$$\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}-\log(1-p_{ija})z_{ja}\geq\log\left(h_is_i(c_s-c_b)\right)-p' \quad (14)$$

Since $p'\in[p_{min},p_{max}]$, the minimum and maximum values that the right-hand-side of the constraint can take are zero and $\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}-\log(1-p_{ija})$, respectively.

Second, the problem can be solved to any desired accuracy (though not exactly the optimum), by quantizing the interval of p' into W steps; this corresponds to multiplying the coefficients with some number M and rounding. Let $W=M\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}-\log(1-p_{ija})$ denote the maximum value of the right-hand-side of (14) after multiplying it by M. Similarly let $q_{ja}=-M\log(1-p_{ija})$ for $j\in\mathcal{R}_i$, $a\in\mathcal{A}_j$. Note that the minimum value that the right-hand-side can take is still zero. Hence, $p'\in[0,W]$.

After these two steps, (13) can be re-expressed as (15). Formulation (15) resembles an inversed multiple-choice knapsack problem with an upper bound (15d) on the number of items. The difference is that we have p' as one additional variable with a term in the objective function. The selection of p' affects the right-hand-side, corresponding to changing the knapsack capacity. Knapsack problem is solved via DP efficiently. The interesting point is that DP provides not only the optimal function value of z with the given capacity, but also those for all intermediate values starting from zero, implying that one computation is enough to examine the effect of all p' values. Then the optimum can be obtained by post processing considering the function term with p'. $\square$

$$\min 10^{p'}+\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}\beta_{ija}^*z_{ja} \quad (15a)$$

$$\text{s.t.}\quad \sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}q_{ja}z_{ja}\geq W \quad (15b)$$

$$\sum_{a\in\mathcal{A}_j}z_{ja}\leq 1, j\in\mathcal{R}_i \quad (15c)$$

$$\sum_{j\in\mathcal{R}_i}\sum_{a\in\mathcal{A}_j}z_{ja}\leq N \quad (15d)$$

$$z_{ja}\in\{0,1\}, j\in\mathcal{R}_i, a\in\mathcal{A}_j \quad (15e)$$

$$p'\in[0,W] \quad (15f)$$

The DP algorithm for solving $\mathrm{PP}^{(i)}$ is shown in Algorithm 3. Lines 1-5 are the initialization steps. Lines 6-13 solves (15) with maximum capacity W in which matrix $\boldsymbol{B}^*$ is the optimal cost matrix. Entry $B^*(w,j,n)$ represents the cost of the optimal solution when up to n contents of the first j contents can be in the recommendation set with a knapsack capacity $w\in[0,W]$. Matrix $\boldsymbol{A}^*$ is an auxiliary matrix that stores the AoI corresponding to optimum for each tuple (w,j,n) where $w\in[0,W], j=1,..,R_i$, and $n=1,2,...,\min\{j,N\}$. Lines 14-32 perform the post processing step. Namely for each intermediate value $p'\in[0,W]$, the corresponding objective function value is calculated and compared to the minimum value found so for, in order to find the global minimum of the problem. The complexity of this algorithm is of $O(WR_iNA_i)$. The column generation algorithm for solving $\mathrm{SP}_2^{(t)}$ is shown in Algorithm 2 in which Algorithm 3 is used for solving $\mathrm{PP}^{(i)}$, $i\in\mathcal{I}$.

Algorithm 2 Column generation for $\text{SP}_2^{(t)}$

1: Initialize $\mathcal{C}_i'$ for $i \in \mathcal{I}$
2: Stop $\leftarrow$ False
3: **while** Stop = False **do**
4:     Solve RMP (9) and obtain dual optimum values $\boldsymbol{\pi}$ and $\boldsymbol{\beta}$
5:     Stop $\leftarrow$ True
6:     **for** $i \in \mathcal{I}$ **do**
7:         Solve $\text{PP}^{(i)}$ by Algorithm 3
8:         **if** the reduced cost< 0 **then**
9:             Stop $\leftarrow$ False
10:             Add the column to $\mathcal{C}_i'$

Algorithm 3 Dynamic programming for $\text{PP}^{(i)}$

1: Create matrix B^* of size $(1+W) \times (1+R_i) \times (1+N)$
2: Create matrix A^* of size $(1+W) \times (1+R_i) \times (1+N)$
3: $B^*[0,j,n] \leftarrow 0$ for any j and n
4: $B^*[w,0,n] \leftarrow \infty$ for any $w > 0$ and any n
5: $w \leftarrow 1$, Stop $\leftarrow$ False
6: **while** Stop = False **do**
7:     **for** $j = 1, ..., R_i$ **do**
8:         **for** $n = 1, ..., \min\{j, N\}$ **do**
9:             $B^*(w,j,n) \leftarrow \min_{a \in \mathcal{A}_j}\{\beta_{ija} + B^*(w', j-1, n-1), B^*(w, j-1, n')\}$, $A^*(w,j,n) \leftarrow \arg\min_{a \in \mathcal{A}_j}\{\beta_{ija} + B^*(w', j-1, n-1), B^*(w, j-1, n')\}$,
            where
            $w' = \max\{0, w - q_{ja}\}$ and $n' = \min\{j-1, n\}$
10:     **if** $B^*(w, R_i, N) = \infty$ or $w = W$ **then**
11:         Stop $\leftarrow$ True
12:     **else**
13:         $w \leftarrow w + 1$
14: OPT $\leftarrow \infty$, $w \leftarrow 1$, Stop $\leftarrow$ False
15: **while** Stop = False **do**
16:     $q^* \leftarrow 0$, $w' \leftarrow w$, $j' \leftarrow R_i$, $n' \leftarrow N$
17:     **if** $B^*(w', j', n') = \infty$ or $w > W$ **then**
18:         Stop $\leftarrow$ True
19:     **else**
20:         **while** $j' \geq 1$ and $n' \geq 1$ **do**
21:             **if** $B^*(w', j', n') < B^*(w', j'-1, n')$ **then**
22:                 $a^* \leftarrow A^*(w', j', n')$
23:                 $w' \leftarrow \min\{0, w' - q_{j'a^*}\}$
24:                 $q^* \leftarrow q^* + q_{j'a^*}$
25:                 $n' \leftarrow n' - 1$
26:             **else**
27:                 $n' \leftarrow \min\{j'-1, N\}$
28:             $j' \leftarrow j' - 1$
29:         $p \leftarrow \log(h_i s_i(c_s - c_b)) - q^*/M$
30:         **if** $10^p + c_b s_i h_i - \pi_i^* + B^*(w, R_i, N) <$ OPT **then**
31:             OPT $\leftarrow 10^p + c_b s_i h_i - \pi_i^* + B^*(w, R_i, N)$
32:     $w \leftarrow w + 1$

6.4 Attaining Integer Feasible Solutions

The solutions of the two subproblems will likely violate some original constraints, and we present an approach to generate feasible solutions based on SP_2. We take the solutions of $\text{SP}_2^{(t)}$, $t \in \mathcal{T}$, and "repair" them in order to construct an integer solution for COPRA. The reason of using $\text{SP}_2^{(t)}$, $t \in \mathcal{T}$ is that its solution

contains the information of recommendation sets, and hence it resembles more a solution to the original problem. However, these solutions do not respect the AoI evolution of contents across the time slots as each $\text{SP}_2^{(t)}$, $t \in \mathcal{T}$, is solved independently from the others. The repairing algorithm (RA) is shown in Algorithm 4, which consists of three main steps. In the algorithm, symbol $\leftarrow$ is used to indicate the assignment of a value. Symbol $\Leftarrow$ is used to indicate that an assigned value of an optimization variable is kept fixed subsequently.

In the first step, we take the solution of $\text{SP}_2^{(t)}$, $t \in \mathcal{T}$, and perform an iterative rounding process on the y-variables to obtain an integer solution. More specifically, we first fix the current y-variables having value one, followed by fixing the variable with the largest fractional value to one if there is enough capacity and zero otherwise. We then solve $\text{SP}_2^{(t)}$ again. Now, if the solution is integer, we stop. Otherwise, this process is repeated until an integer solution is obtained. Obviously, in the worst case, I iterations are needed to obtain an integer solution. Denote by $\hat{\boldsymbol{y}} = \{\hat{y}_{ti} : t \in \mathcal{T} \text{ and } i \in \mathcal{I}\}$ the obtained values of y-variables of each $\text{SP}_2^{(t)}$ for $t \in \mathcal{T}$. This step corresponds to Lines 1-9 in Algorithm 4.

In the second step, we utilize $\hat{\boldsymbol{y}}$ as input to the optimization problem stated in (16). Therein, the y-variables have the same meaning as defined earlier in Section 3.3. Solving (16) provides a caching solution in which the AoI evolution of contents across time slots are respected. The objective function is maximization, to encourage setting the $\boldsymbol{y}$-variables to be as similar to $\hat{\boldsymbol{y}}$ as possible. Here, ϵ is a small positive number, to encourage caching contents even if $\hat{y}$ is zero. This step corresponds to Line 10 in Algorithm 4.

$$\max_{\boldsymbol{y},\boldsymbol{x} \in \{0,1\}} \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}} (\epsilon + h_{ti} s_i \hat{y}_{ti}) y_{ti} \tag{16a}$$
$$\text{s.t.} \quad (3\text{b}), (3\text{c}), (3\text{d}), (3\text{g}), (3\text{h})$$

After these two steps, we have a complete caching solution over time slots. Finally, for each non-cached content item, we choose the N highest related cached contents as its recommendation set. This step corresponds to Lines 11-13 in Algorithm 4. We remark that formulation (16) is an integer program. However in practice this is solved rapidly. Moreover, the repairing operation does not need to be done in every iteration of subgradient optimization.

6.5 Algorithm Summary

The main steps of LDA is shown in Algorithm 5. Line 1 initialize the total number of iterations K to perform, and tolerance parameters ϵ_1 and ϵ_2. Lines 2 and 3 initialize the vector of Lagrangian multiplier $\boldsymbol{\lambda}$ to 0, the iteration counter $k = 1$, the lower bound LBD to zero, and the best found solution $\bar{w}$ to ∞. Lines 5 and 6 solve the $\text{SP}_1^{(i)}$ for $i \in \mathcal{I}$ and $\text{SP}_2^{(t)}$ for $t \in \mathcal{T}$, respectively. Lines 7 and 8 calculate the Lagrangian function value, and update the LBD if a higher lower bound is found. Lines 9 finds a solution for the problem, and then Line 10 updates the current upper bound if a solution with lower objective function value is obtained. Line 11 updates the Lagrange multipliers, and Line 12 increases the iteration counter by one. Finally, Line 13 checks whether a stopping criterion is met.

Algorithm 4 RA for constructing integer solutions

Input: $\text{SP}_2^{(t)}$ for $t \in \mathcal{T}$
Output: An integer solution for COPRA

1: **for** $t \in \mathcal{T}$ **do**
2:    **while** (exists y_{ti} with fractional value) **do**
3:       $y_{ti} \Leftarrow 1$ if $y_{ti} = 1$
4:       $g = \max\{y_{ti} : 0 < y_{ti} < 1\}$
5:       $j = \arg\max\{y_{ti} : 0 < y_{ti} < 1\}$
6:       $\Phi \leftarrow L$ **if** $(t = 1)$ **else** $\Phi \leftarrow S$
7:       $y_{tj} \Leftarrow 1$ **if** $(s_j + \sum_{\substack{i \in \mathcal{I} \\ y_{ti}=1}} s_i \le \Phi)$ **else** $y_{tj} \Leftarrow 0$
8:       Solve $\text{SP}_2^{(t)}$
9: $\hat{\boldsymbol{y}} \leftarrow \{y_{ti} : t \in \mathcal{T}, i \in \mathcal{I}\}$
10: Solve formulation (16) and obtain the values of $\boldsymbol{y}$
11: **for** $t \in \mathcal{T}$ **do**
12:    **for** $i \in \mathcal{I} : y_{ti} = 0$ **do**
13:       $c \leftarrow$ the first N elements in $\{(j, a) : x_{tja} = 1, j \in \mathcal{R}_i, a \in \mathcal{A}_{tj}\}$ with the highest relations to i

---

Algorithm 5 The main steps of LDA

1: Initialize K, ϵ_1, and ϵ_2
2: $\boldsymbol{\lambda} \leftarrow \mathbf{0}$, $k \leftarrow 1$
3: $\text{LBD} \leftarrow 0$, $\bar{w} \leftarrow \infty$
4: **repeat**
5:    Solve $\text{SP}_1^{(i)}$ for $i \in \mathcal{I}$ and obtain $\boldsymbol{x}^{(k)}$
6:    Solve $\text{SP}_2^{(t)}$ for $t \in \mathcal{T}$ by Algorithm 2 and obtain $\boldsymbol{x}'^{(k)}$
7:    Calculate $L(\boldsymbol{\lambda}^{(k)})$ which is (6a)
8:    **if** $L(\boldsymbol{\lambda}^{(k)}) > \text{LBD}$ **then** $\text{LBD} \leftarrow L(\boldsymbol{\lambda}^{(k)})$
9:    Apply Algorithm 4 to obtain an integer solution and its objective function value U
10:    **if** $U < \bar{w}$ **then** $\bar{w} \leftarrow U$
11:    Calculate $\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda}^{(k)} + t^{(k)}\boldsymbol{d}^{(k)}$ where $t^{(k)} = \eta \frac{\bar{w} - L(\boldsymbol{\lambda}^{(k)})}{||\boldsymbol{d}^{(k)}||^2}$, $\boldsymbol{d}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}'^{(k)}$
12:    $k \leftarrow k + 1$
13: **until** $||\boldsymbol{d}^{(k)}|| > \epsilon_1$ and $||\boldsymbol{\lambda}^{(k+1)} - \boldsymbol{\lambda}^{(k)}|| > \epsilon_2$ and $k > K$

---

7 PERFORMANCE RESULTS

In this section, we present performance evaluation results of LDA and GA. We first consider small-size problem instances, and evaluate the performances of LDA and GA by comparing them to the global optimum obtained from ILP (3). We report the (relative) deviation from the optimum, referred to as the optimality gap. For large-size problem instances, it is computationally difficult to obtain global optimum. Instead, we use the LBD derived from LDA as the reference value. This is a valid comparison because the deviation with respect to the global optimum will never exceed the deviation from the LBD. We will see that, numerically, using the LBD remains accurate in evaluating optimality.

The content popularity is modeled by a ZipF distribution, i.e., the probability where the i-th content is requested is $\frac{i^{-\gamma}}{\sum_{i \in \mathcal{I}} i^{-\gamma}}$ [36], [37]. Here γ is the shape parameter and it is set to $\gamma = 0.56$ [36]. The sizes of content items are generated within interval $[1, 10]$. We have set the the cache capacity to 50% of the total size of content items, i.e., $C = 0.5 \sum_{i \in \mathcal{I}} s_i$. The capacity of backhaul link is set to $L = \rho \sum_{i \in \mathcal{I}} s_i$ where parameter ρ steers the backhaul capacity in relation to the total size of content

items. The probability of accepting a related content is generated in interval $[0.6, 1)$. The maximum AoI that a content can take is set to two. We use content-specific and time-specific functions including linear and nonlinear ones from the literature [38], [39] to model the AoI cost of content items. Specifically, for each content, one of the following functions is randomly selected: $f_{tia} = 1 + \alpha_{ti}a$, $f_{tia} = \frac{1}{1 - \alpha_{ti}a}$, and $f_{tia} = e^{\alpha_{ti}a}$. The functions are made content-specific and time-specific by varying parameter α_{ti}. We remark that the performance of LDA remains largely the same if only one type of function is used for all contents. The use of multiple functions is to show that the algorithm works in general with diverse functions. We will vary parameters I, T, and ρ, and study their impact on the overall cost and algorithm performance. For each input setup, we have generated 10 problem instances and we report the average cost.

Figure 2 shows the total cost returned by LDA when recommendation is utilized, and LDA with no recommendation (denoted by LDC-NC). The figure shows that, interestingly, the total cost decreases by more than 50% with recommendation. Another interesting point is that the reduction is even more when the number of content items increases. From this result, the consideration of recommendation optimization is relevant.

Fig. 2. Impact of I on total cost when $T = 12$, $S = 0.5 \sum_{i \in \mathcal{I}} s_i$, $L = 0.3 \sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$. The blue and pink lines show the total cost with and without recommendation, respectively.

Figures 3-5 and Figures 6-8 show the performance results for the small-size and large-size problem instances, respectively. In Figures 3-5, the green line represents the global optimum computed using ILP (3). In Figures 6-8, the black line represents the LBD obtained from LDA. In all figures, the blue and red lines represent the overall cost returned by LDA and GA, respectively. The deviation from global optimum for LDA is within a few percent, while for GA it is significantly larger. Moreover, the results for both small-size and large-size problem instances are consistent.

Figure 3 shows the impact of content items on the total cost for small-size problem instances. The overall cost slightly decreases with the number of contents. This is due to the fact that the capacity of cache is set relatively to the total. Namely, with larger number of contents, more capacity is available, and hence more opportunity to serve content requests from the cache. This effect, however, can not be seen for large problem instances due to a saturation effect, see Figure 6. As can be seen the cost has fluctuations due to instable solutions of GA. For small-size problems, the optimality gap of GA is about 57%, while for LDA it is about 7% from global optimum. For large-size problems, the performance of LDA remains the same, while that of GA increases

Fig. 3. Impact of I on total cost when $T = 6$, $S = 0.5 \sum_{i \in \mathcal{I}} s_i$, $L = 0.3 \sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

Fig. 5. Impact of ρ on total cost when $I = 20$, $T = 6$, $S = 0.5 \sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

Fig. 4. Impact of T on total cost when $I = 20$, $S = 0.5 \sum_{i \in \mathcal{I}} s_i$, $L = 0.3 \sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

Fig. 6. Impact of I on total cost when $T = 12$, $S = 0.5 \sum_{i \in \mathcal{I}} s_i$, $L = 0.3 \sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

to 70%. Intuitively, the reason is that with larger number of items, the problem becomes too difficult for a simple algorithm such as GA.

Figures 4 shows the impact of time slots for small-size problem instances. As can be seen, the cost increases with number of time slots. Apparently, this is because with more time slots, there are more requests to serve, and hence higher cost. GA has an optimality gap around 60%, while for LDA the gap is only 8%. The results for large-size problems are shown in Figure 7. LDA consistently shows good performance, whereas the results of GA are very sub-optimal. It is worth noting that the optimality gaps of both LDA and GA slightly increase with the number of time slots.

Figure 5 shows the impact of ρ on the total cost. Larger ρ means higher backhaul capacity. The costs of both LDA and GA decrease sharply when ρ increases from 10% to 20%, then the decrease slows down due to a saturation effect. The optimality gap of LDA is 17.5% when $\rho = 10\%$. This is because when the backhaul capacity is extremely limited, very few content items can be updated in a time slot, and as a result even one or two sub-optimal choices would largely impact the performance. When ρ increases to 20%, the cost significantly decreases and the optimality gap decreases as well to 7.8%. For higher value of ρ, the gap slightly decreases further and stays around 7%. For GA the deviation from optimality is high no matter ρ is small or not. Similar trends can be seen for large-size problems, see Figure 8. Note that in the figure it may not be clear that the result of LDA and LBD both decrease with ρ. To show this, we have plotted a subfigure in the middle-right section of the figure.

8 CONCLUSIONS

We have studied optimal scheduling of cache updates where AoI of contents and recommendation are jointly taken into account. With both AoI and recommendation, the problem is hard even for one single time slot. We formulated the problem as an integer liner program (ILP). The ILP provides optimal solutions, but it is not practical to large problem instances. Simple algorithms are not likely to be effective, and this finding is obtained via the poor performance of a greedy algorithm (GA). To arrive at good solutions efficiently, one has to analyze and exploit the structure of this optimization problem. We achieve this by the Lagrangian decomposition algorithm (LDA) that allows for decomposition for handling large-scale problem instances. LDA decomposes the problem into several subproblems where each of them can be solved efficiently. The algorithm provides solutions within a few percentage from global optimality.

REFERENCES

[1] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, 2016.

[2] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Transactions on Management Information Systems*, vol. 6, no. 4, pp. 1–19, 2016.

[3] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 404–410.

[4] 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, 2019.

Fig. 7. Impact of T on total cost when $I = 50$, $S = 0.5\sum_{i \in \mathcal{I}} s_i$, $L = 0.3\sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

Fig. 8. Impact of ρ on total cost when $I = 50$, $T = 12$, $S = 0.5\sum_{i \in \mathcal{I}} s_i$, and $\gamma = 0.56$.

[5] ——, "Caching-aware recommendations: Nudging user preferences towards better caching performance," in *IEEE Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.

[6] 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.

[7] L. Song and C. Fragouli, "Making recommendations bandwidth aware," *IEEE Transactions on Information Theory*, vol. 64, no. 11, pp. 7031–7050, 2018.

[8] M. Costantini, T. Spyropoulos, T. Giannakas, and P. Sermpezis, "Approximation guarantees for the joint optimization of caching and recommendation," in *IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.

[9] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, "Age-optimal constrained cache updating," in *IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 141–145.

[10] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *IEEE Conference on Computer Communications (INFOCOM)*, 2012, pp. 2731–2735.

[11] S. Zhang, J. Li, H. Luo, J. Gao, L. Zhao, and X. S. Shen, "Towards fresh and low-latency content delivery in vehicular networks: An edge caching aspect," in *10th International Conference on Wireless Communications and Signal Processing (WCSP)*, 2018, pp. 1–6.

[12] C. Kam, S. Kompella, G. D. Nguyen, J. E. Wieselthier, and A. Ephremides, "Information freshness and popularity in mobile caching," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 136–140.

[13] G. Ahani and D. Yuan, "Accounting for information freshness in scheduling of content caching," in *IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[14] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. Yates, "Age of information aware cache updating with file- and age-dependent update durations," in *18th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, 2020, pp. 1–6.

[15] G. Ahani, D. Yuan, and S. Sun, "Optimal scheduling of age-centric caching: Tractability and computation," *IEEE Transactions on Mobile Computing*, 2020, DOI:10.1109/TMC.2020.3045104.

[16] M. Bastopcu and S. Ulukus, "Maximizing information freshness in caching systems with limited cache storage capacity," in *54th Asilomar Conference on Signals, Systems, and Computers*, 2020, pp. 423–427.

[17] ——, "Information freshness in cache updating systems," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1861–1874, 2021.

[18] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1183–1210, 2021.

[19] S. Kastanakis, P. Sermpezis, V. Kotronis, D. S. Menasche, and T. Spyropoulos, "Network-aware recommendations in the wild: Methodology, realistic evaluations, experiments," *IEEE Transactions on Mobile Computing*, 2020, DOI:10.1109/TMC.2020.3042606.

[20] D. Tsigkari and T. Spyropoulos, "User-centric optimization of caching and recommendations in edge cache networks," in *IEEE 21st International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2020, pp. 244–253.

[21] Y. Fu, K. N. Doan, and T. Q. Quek, "On recommendation-aware content caching for 6g: An artificial intelligence and optimization empowered paradigm," *Digital Communications and Networks*, vol. 6, no. 3, pp. 304–311, 2020.

[22] Y. Fu, Z. Yang, T. Q. S. Quek, and H. H. Yang, "Towards cost minimization for wireless caching networks with recommendation and uncharted users' feature information," *IEEE Transactions on Wireless Communications*, 2021, DOI:10.1109/TWC.2021.3076495.

[23] K. Guo and C. Yang, "Temporal-spatial recommendation for caching at base stations via deep reinforcement learning," *IEEE Access*, vol. 7, pp. 58 519–58 532, 2019.

[24] T. Giannakas, P. Sermpezis, A. Giovanidis, T. Spyropoulos, and G. Arvanitakis, "Fairness in network-friendly recommendations," in *IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2021, pp. 71–80.

[25] D. Liu and C. Yang, "A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users," *IEEE Access*, vol. 7, pp. 83 120–83 136, 2019.

[26] M. Costantini and T. Spyropoulos, "Impact of popular content relational structure on joint caching and recommendation policies," in *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOPT)*, 2020, pp. 1–8.

[27] M. Garetto, E. Leonardi, and G. Neglia, "Similarity caching: Theory and algorithms," in *IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 526–535.

[28] J. Zhou, O. Simeone, X. Zhang, and W. Wang, "Adaptive offline and online similarity-based caching," *IEEE Networking Letters*, vol. 2, no. 4, pp. 175–179, 2020.

[29] D. Liu and C. Yang, "A learning-based approach to joint content caching and recommendation at base stations," in *IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.

[30] N. Zhang, K. Zheng, and M. Tao, "Using grouped linear prediction and accelerated reinforcement learning for online content caching," in *IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018, pp. 1–6.

[31] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[32] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, 2009.

[33] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[34] J. L. Goffin, "On convergence rates of subgradient optimization methods," *Mathematical Programming*, vol. 13, pp. 329–347, 1977.

[35] M. S. Bazaraa and H. D. Sherali, "On the choice of step size in subgradient optimization," *European Journal of Operational Research*, vol. 7, no. 4, pp. 380–388, 1981.

[36] 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.

[37] G. Ahani and D. Yuan, "Optimal scheduling of content caching subject to deadline," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 293–307, 2020.

[38] Y. Sun, E. Uysal-Biyikoglu, R. D. Yates, C. E. Koksal, and N. B. Shroff, "Update or wait: How to keep your data fresh," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7492–7508, 2017.

[39] Y. Sun and B. Cyr, "Sampling for data freshness optimization: Non-linear age functions," *Journal of Communications and Networks*, vol. 21, no. 3, pp. 204–219, 2019.

Ghafour Ahani received the B.Sc. and the M.Sc. degrees in applied mathematics from the University of Kurdistan, Sanandaj, Iran, in 2008 and 2010 respectively. He is currently pursuing the Ph.D. degree with the IT department, Uppsala University, Sweden. His research interests include mathematical optimization applied to networks and resource allocation in communication systems.

Di Yuan (SM'16) received his MSc degree in Computer Science and Engineering, and PhD degree in Optimization at Linköping Institute of Technology in 1996 and 2001, respectively. After his PhD, he has been associate professor and then full professor at the Department of Science and Technology, Linköping University, Sweden. In 2016 he joined Uppsala University, Sweden, as chair professor. His current research mainly addresses network optimization of 4G and 5G systems, and capacity optimization of wireless networks. Dr Yuan has been guest professor at the Technical University of Milan (Politecnico di Milano), Italy, in 2008, and senior visiting scientist at Ranplan Wireless Network Design Ltd, United Kingdom, in 2009 and 2012. In 2011 and 2013 he has been part time with Ericsson Research, Sweden. In 2014 and 2015 he has been visiting professor at the University of Maryland, College Park, MD, USA. He is an area editor of the Computer Networks journal. He has been in the management committee of four European Cooperation in field of Scientific and Technical Research (COST) actions, invited lecturer of European Network of Excellence EuroNF, and Principal Investigator of several European FP7 and Horizon 2020 projects. He is a co-recipient of IEEE ICC'12 Best Paper Award, and supervisor of the Best Student Journal Paper Award by the IEEE Sweden Joint VT-COM-IT Chapter in 2014.