Optimal Content Caching and Recommendation With Age of Information

Ghafour Ahani and Di Yuan, Senior Member, IEEE

Abstract—Content caching at the network edge is 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. 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). This article has the following contributions. First, we address optimal scheduling of cache updates for a time-slotted system accounting for content recommendation and AoI, and to the best of our knowledge, there is no work that has jointly taken into account these aspects. Next, we rigorously prove the problem’s NP-hardness. Then, we derive an integer linear formulation, by which the optimal solution can be obtained for small-scale scenarios. On the algorithmic side, our contributions include the development of an effective algorithm based on Lagrangian decomposition, and efficient algorithms for solving the resulting subproblems. Our algorithm computes a bound that can be used to evaluate the performance of any suboptimal solution. We conduct simulations to show the effectiveness of our algorithm.

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

1 INTRODUCTION

1.1 Background and Motivation

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 with reduced latency, and the 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, recommendation systems have been used for presenting content items that best match user interests and preferences. In fact, the studies in [2], [3] show that 80% of requests on content distribution platforms are due to content recommendation. 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 here 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 any of the recommended contents is accepted, 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, the cost of serving a request with a content with larger AoI is larger.

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.

1.2 Contributions

We contribute to optimal edge caching by addressing recommendation and AoI. Specifically, for the resulting cache
optimization problem with recommendation and AoI (COPRA), the contributions are as follows.

- 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).
- For mathematical modeling, our contribution is 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 algorithms.
- For problem solving, our contribution is a Lagrangian decomposition scheme based on 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 further. However, the problem size is exponential. To deal with this issue, we propose the use of column generation that solves the continuous relaxation of the subproblem, with the advantage of waiving the need of explicitly considering all decision variables in the solution process. 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 9% of global optimality.

2 RELATED WORK

To the best of our knowledge, there is no work that jointly studies 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 where AoI is accounted for. The general problem setup in these works is to determine 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 update interval of each item or the total number of updates is known. In [12], [13], [14] content caching is studied considering both content popularity and AoI. 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] the authors consider AoI with partial content updates. In [15], the overall utility of a cache that is defined based on AoI of contents is maximized, subject to limited cache and backhaul link capacities. 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. For a given origin, a set of caches, and a set of users accessing the caches, the AoI of the cache(s) and users is analyzed in [17].

AoI is of importance to scenarios where content items are continuously updated. One specific example is news distribution platforms where the news are continuously updated and the cache holds local copies [18]. Another relevant scenario is caching of user-generated contents (UGC) where the contents become less useful as time elapses [19]. The works in [20], [21], [22] studied problems in which information are updated with known rates. In these works, the term age of version is used instead of AoI to indicate that in fact the content is the same but the information gets updated leading to a different version of the content. The works in [20] and [21] studied maximizing the freshness of information in gossip networks. In [20] a source node directly forwards updates to a set of monitoring nodes, while in [22] the monitoring nodes are grouped into clusters and the updates are conveyed via cluster heads. In [21] the authors considered a local cache that holds a set of content items. The cost increases linearly with the AoI of contents. The authors derived caching and eviction strategies in order to balance the tradeoff between the cost and AoI. The work in [23], unlike all other works, considered age of incorrect information in real-time remote monitoring applications. More specifically, a remote node monitors in real-time a source node. The objective is to minimize the cost of the monitor node being in an erroneous state over time. To this end, age of incorrect information is used to model the cost. The authors formulated the problem as a Markov decision process, and computed the optimal solution via dynamic programming. A recent survey of AoI can be found in [24].

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 them toward the contents that are both stored in the cache and interesting to the users [4], [5], [25], [26], [27], [28], [29], [30], [31]. In [4], [5] a preference "distortion" tolerance measure is used to quantify how much the engineered recommendations distort the original user content preferences. In [25], an experiment is conducted to demonstrate the effect of content recommendation on caching efficiency in practice. In [26], 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 [27], [28], caching and recommendation decisions are optimized based on the preference distribution of individual users. In [29], content caching and recommendation are optimized taking into account the temporal-spatial variability of user requests. In [30], the authors studied the fairness issues of recommendation where some contents get more visible than others by recommendation. In [31], reinforcement learning is utilized for learning user behavior and optimizing caching and recommendation.

In the second category of studies, the purpose of recommendation is 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], [32], [33], [34]. 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 illustrated how “soft cache hit” is able to improve the caching performance. They also considered 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 proposed algorithms with performance guarantees. Later, in [8], the authors considered a more realistic system model in which only a limited number of contents can be recommended. Then, they proposed a polynomial-time algorithm based on first solving the caching problem, and then finding the recommendations sets. In [32], the authors modeled the relation among contents as a graph, and then studied the characteristics of this graph to predict whether it is worth finding the optimal solution or a low complexity heuristic will be sufficient. In [33], the authors attempted to find the best caching policy for a sequence of requests where recommendation is accounted for. In [34] 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 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 PROBLEM FORMULATION

3.1 System Scenario

The system scenario consists of a content server, a base station (BS), and a set of content items $I = \{1, 2, \ldots, I\}$. 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 cached contents can be updated. The size of content item $i \in I$ is denoted by $s_i$. The system scenario is shown in Fig. 1.

![System model](image)

We consider a time-slotted system with a time period of $T$ time slots, denoted by $T = \{1, 2, \ldots, T\}$. 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 $i \in I$, 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 $A_{ti} = \{0, \ldots, \min(A_i, t - 1)\}$. The cost associated with content item $i$ with AoI $a$ in time slot $t$ is characterized by a cost function $f_{ia}$ that is monotonically increasing in AoI $a_i$ and, therefore, is also monotonically increasing in $t$ unless the content is refreshed. We remark that our algorithms allow for use of any type of AoI-cost function. Namely, an AoI-cost function that increases rapidly can be used for content items that change frequently, and an AoI-cost function that increases slowly (or even is constant) for a content that is seldom updated.

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 and cached contents, hereinafter referred to as a recommendation set, is recommended to the user. If the content is not refreshed, 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], [35]. Denote by $R_i = \{1, 2, \ldots, R_i\}$ 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 $R_i$. Note that the recommendation set may change from a time slot to another.

Denote by $h_{it}$ the number of requests for content $i \in I$ in time slot $t \in T$. The value of $h_{it}$ can be estimated via recent requests of the contents, popularity of the contents, and/or machine learning algorithms [6], [36]. In our 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, $c_s \gg c_b$ to encourage downloading from the cache.

The goal of 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. COPRA is a discrete optimization problem with a combinatorial nature and knapsack-like constraints. We will prove that the problem is at least as hard as 3-SAT. The difficulty arises from the fact that caching, updating, and recommendation decisions are intertwined. Notation related to system model is summarized in Table 1.

### 3.2 Cost Model

Denote by $x_{ita}$ a binary optimization variable that equals one if and only if content $i$ with AoI $a$ is cached in time slot $t$. Hence, $x_{ita} = 1$ means that content $i$ is updated in time slot $t$ and thus the AoI becomes zero. This requires downloading the content from the server to the cache with cost

$$
\Delta_{update} = \sum_{t \in T} \sum_{i \in I} (c_a - c_b)s_i x_{ita}.
$$

Moreover, the cost of obtaining requested contents that are cached (and thus no recommendation is involved) is

$$
\Delta_{direct} = \sum_{t \in T} \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}_i} c_h s_i h_{ti} f_{ita} x_{ita}.
$$

Here, $f_{ita}$ is a function that maps the AoI of a content to a (cost) value. Note that the cost in (2) is not the downloading cost, but the cost of jointly considering downloading and AoI. Here, $f_{ita}$ acts as a scaling factor, with the effect of discouraging downloading from the cache if the cached content has large AoI (i.e., it has not been updated for a long time). The expression in (2) has a sum over all possible AoL values of a cached content. However, at most one of these may materialize, and this is accounted for by imposing the constraint that at most one of the $x$-variables may be one in the optimization model (see Section 3.3).

Next, we calculate the downloading cost related to content recommendation. Denote by $p_{j;ta}$ 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 $p_{j;ta}$ can be calculated based on historical statistics [6], item-item recommendation [37], and/or collaborative filtering techniques [38]. 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 $c_i$ the set of such recommendation sets for content $i \in \mathcal{I}$ in time slot $t$.

Denote by $v_{litc}$ a binary optimization variable that takes value one if and only if content $i$ is not stored in the cache in time slot $t$, but instead content set $c \in c_{it}$ is recommended. The probability of not accepting any of the contents in $c$ is $P_{ic} = \prod_{j \in c}(1 - p_{j;tc})$. Thus, the probability of accepting at least one of them is $1 - P_{ic}$, and hence the expected cost is

$$
\Delta_{recom} = \sum_{t \in T} \sum_{i \in \mathcal{I}} \sum_{c \in c_{it}} (c_b(1 - P_{ic}) + c_s \bar{P}_{ic}) s_i h_{ti} v_{litc}.
$$

As $c$ may contain up to $N$ items, the cardinality of $c$ is at most $N$. Moreover, as accepting one of the contents in recommendation set $c$ is probabilistic, $\Delta_{recom}$ is the expected cost of downloading from the cache for requested contents that are not in the cache but are satisfied via recommendation. From an algorithmic standpoint, there is a trade-off between problem size and the number of recommended contents. In the most restrictive case, one content may be recommended. This simplifies the problem formulation but significantly limits the potential of recommendation. In practice, the size of recommendation set can be regarded as a control parameter, and set by the network operator. Finally, the total cost of system is the sum of $\Delta_{update}, \Delta_{direct}$, and $\Delta_{recom}$.

Note that for recommendation, whether or not accepting a recommended content is a probabilistic event. Hence, the use of AoI-specific probability $p_{j;ta}$ is the mechanism for avoiding recommending old contents. Namely, the probability $p_{j;ta}$ decreases in AoI $a$. By the definitions of $\bar{P}_{ic}$ and cost for recommendation (3), lower $p_{j;ta}$ means the cost of recommending old contents grows.

From (2) and (3), one can see that AoI is a metric to aid in decision making in system modeling. Another metric is latency that, in the context of content caching, is characterized by whether content requests are served by the cache (low latency) or the central server (high latency). This is reflected in the cost model by assuming that $c_b$ is much higher.

---

1. We have considered an overall acceptance probability of $p_{j;ta}$ for all users for the sake of simplicity. However, our algorithms and formulations can be easily adapted such that they can accommodate user’s preference in the acceptance probability. This can be simply achieved by adding user index to $p_{j;ta}$.

2. In this cost, $1 - P_{ic}$ is multiplied by the size of initially requested content $i$. The reason is that we consider recommending only contents having similar size to $i$. However, one can also consider recommending contents with very different sizes and use the average of them for calculating the expected cost.
lower that \( c_i \) in (2) and (3). If the model would focus only on latency without consideration of AoI (i.e., replacing the AoI coefficient \( f_{iia} \) with a constant of one), the overall cost will then decrease as there is no penalty of holding old contents in the cache. The resulting solution will become rather different, as caching (and recommendation) will become governed only by content popularity over time, not information freshness. As a result, old contents will be used to serve requests (albeit with low latency). This observation further justifies the inclusion of AoI via \( f_{iia} \) in (2) and (3), with the effect that the system will refresh cached contents when the AoI becomes large.

### 3.3 Problem Formulation

COPRA can be formulated using integer-linear programming (ILP), as shown in (4). In (4), we use \( y_i \) as an auxiliary binary variable that equals one if and only if content item \( i \) is cached in time slot \( t \). Constraints (4b) state that if content \( i \) is cached in time slot \( t \), then \( y_i \) should exactly take one of the possible AoIs values in \( A_i \). Constraints (4c) and (4d) together ensure that content \( i \) is cached in time slot \( t \) then it has AoI \( a \) (i.e., \( x_{tia} = 1 \)) if and only if three conditions hold: Item \( i \) is in the cache \( (y_i = 1) \), it has AoI \( a - 1 \) in time slot \( t - 1 \) \( (x_{(t-1)(a-1)} = 1) \), and it is not refreshed again in slot \( t \) \( (x_{tia} = 0) \). By (4e), either \( x_{tia} \) is in the cache or a set of contents is recommended. This set can be empty which corresponds to not recommending any content and therefore the requested content will be downloaded from the server. Constraints (4f) ensure that the contents in recommendation set \( \epsilon \) are indeed cached. Constraints (4h) and (4g) formulate the cache and backhaul capacities.

\[
\text{ILP} : \quad \min \sum_{y \in \{0,1\}} \Delta_{\text{update}} + \Delta_{\text{direct}} + \Delta_{\text{recom}} \quad (4a) \\
\text{s.t.} \sum_{a \in A_i} x_{tia} = y_i, t \in T, i \in I \\
x_{tia} \geq y_i + x_{(t-1)(a-1)} - x_{t00} - 1, \quad t \in T \setminus \{1\}, i \in I, a \in A_i \setminus \{0\} \\
x_{tia} \leq x_{(t-1)(a-1)}, \quad t \in T \setminus \{1\}, i \in I, a \in A_i \setminus \{0\} \\
\sum_{c \in C_i} y_i + y_i = 1, t \in T, i \in I \\
\sum_{c \in C_{i}(j) \in c} v_{tie} \leq x_{tju}, t \in T, i \in I, j \in R_i, a \in A_i \\
\sum_{t \in T} s_{jti} \leq S, t \in T \\
\sum_{t \in T} s_{jti} \leq L, t \in T \quad (4h) \\
\]

As the number of recommendation sets is exponential, the ILP is exponential in size. However, the ILP is of interest for solving small-scale problem instances for gauging the performance of other suboptimal solutions. Notation related to ILP (4) is summarized in Table 2.

### 4 Complexity Analysis

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

For the NP-hardness proof, we provide a reduction from the 3-SAT problem. This problem is the first problem that was proven to be NP-complete [39]. Thus no algorithm with polynomial-time complexity is expected for 3-SAT. We will show that the recognition version of COPRA is NP-complete, and by complexity theory this means immediately COPRA itself is NP-hard. The recognition version of an optimization problem does not ask for the optimum, but the answer to the question: Does there exist a solution such that the corresponding objective function value reaches some given value?

The term reduction refers to problem transformation. Namely, for a generic instance of 3-SAT, we define a special case of COPRA, such that solving the recognition version of this special case gives the correct answer to the generic 3-SAT instance. If there would exist some polynomial-time algorithm for solving COPRA, then obviously the algorithm can solve the special case of COPRA, also in polynomial time. Because solving the special case leads to the correct answer of a generic instance of 3-SAT, this would mean we can solve 3-SAT in polynomial time, contradicting the fact that 3-SAT is NP-complete. The contradiction, in turn, proves the hardness of COPRA.

Thus, the core of our reduction proof is to define a special case of COPRA and prove that solving the recognition version of this special case will solve 3-SAT. To this end, for such a proof we have the freedom of choosing the problem size as well as setting the problem parameters of COPRA, including the number of contents, how they are related, the content sizes, the cost values, etc.

**Theorem 1.** COPRA is NP-hard.

**Proof.** A 3-SAT problem consists of \( k \) clauses and \( n \) Boolean variables \( u_1, u_2, \ldots, u_n \). A variable or its negation is called a literal. Denote by \( \bar{u}_i \), the negation of \( u_i \), \( i = 1, 2, \ldots, n \). Each clause consists of a disjunction of exactly three different literals, for example, \( u_1 \lor u_2 \lor u_7 \). A literal typically appears in more than one clause. A clause is said to be satisfied if at least one of its literals has value true. The task is to determine if there is an assignment of true/false values to the Boolean variables, such that all clauses are true.

We construct a special case of COPRA based on 3-SAT as follows. First, we define one content for each 3-SAT literal, and one content for each 3-SAT clause. To distinguish between them and relate them to the 3-SAT instance, we call...
them literal and clause contents, respectively. Hence there are $2n$ literal contents, and $k$ clause contents. The former is because a literal can be a Boolean variable or its negation. Moreover, we define $n$ additional contents, one for each pair of variable and its negation in the 3-SAT instance. These contents are defined to help establish the conclusion that solving the resulting COPRA special case would also solve 3-SAT, and we call them auxiliary contents. Hence, there are in total of $I = 3n + k$ contents in the COPRA special case that we are constructing.

All contents have unit size, i.e., $s_i = 1$ for $i \in I$. Recall that for any pair of Boolean variable and its negation (in 3-SAT), we have defined two contents (in category literal contents), and one additional content (in category auxiliary contents). These three contents are related with acceptance probability of one. Thus, requests made for any of these three can be fully satisfied by recommending any of the other two contents (as long as the recommended contents are cached). This construction applies to each pair of Boolean variable and its negation.

The number of time slots is one, i.e., $T = \{1\}$, 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 (defined for the three literals in the clause in 3-SAT) 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. Moreover, in the special case of COPRA that we construct, 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 $c_0$ and $c_s$ are set to $1$ and $2$, respectively. We now show there is an optimal solution of the defined COPRA special case, such that, for the two literal contents defined for each pair of Boolean variable and its negation, the cache stores exactly one of them. This also implies that no auxiliary content or clause content is cached. Suppose the above solution structure is not the case at optimum, then some of the following scenarios must be true. Note that these scenarios are not mutually exclusive.

- For some pair of Boolean variable and its negation, the two corresponding literal contents are both cached. This is clearly not optimal, because they are related with acceptance probability of one in recommendation. Hence it is better to keep only one of them, and use the spare capacity to cache some other content.
- At least one auxiliary content $a$ is cached. Recall that each auxiliary content is defined for some pair of variable and its negation in 3-SAT, and for the same pair there are two literal contents, and these three contents are related with probability one. Suppose the two corresponding literal contents of $a$ are $\ell$ and $\bar{\ell}$. By the argument above, $\ell$ and $\bar{\ell}$ are not both cached. Hence, either one or none of them is cached. The former case is clearly non-optimal, because any request for $a$ can be satisfied by recommending $\ell$ or $\bar{\ell}$ whichever is cached, thus it is better to remove $a$ and cache some other content. For the latter case, suppose we replace $a$ with $\ell$ or $\bar{\ell}$. Doing so will not increase the total cost, again because any request for $a$ can be satisfied by $\ell$ or $\bar{\ell}$. The cost may decrease, because unlike $a$, $\ell$ or $\bar{\ell}$ is related also to some clause contents with probability one, by the construction of the COPRA special case.
- Finally, suppose a clause content is cached. Then, at least one request for an auxiliary content must be served using the server with cost $2(n + k + 1)$. For the requests made 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 content 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 easily verified that $\Delta_1 > \Delta_2$.

By the above, there is an optimal solution where the cache stores exactly one of the two literal contents defined for each pair of Boolean variable and its negation, and no auxiliary contents or clause contents. Thus the optimal total cost due to requesting the literal and auxiliary contents is known.

Clearly the construction above is polynomial in size. If the correct answer to the 3-SAT instance is no, then for the requests made for the clause contents, at least one has to be satisfied by the server with cost $c_s = 2$, and each of the others has at least the cost of $c_0 = 1$. Thus, the total cost is at least $\delta_1 = k + 1$. If the 3-SAT instance is satisfiable, then the corresponding cost 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 $\delta_2$ gives the correct answer to 3-SAT. Therefore, the recognition versions of COPRA is NP-complete and its optimization version is thus NP-hard.

In practice, the content items may naturally fall into different categories based on the type of the content, e.g., video contents can be categorized based on if it is science fiction, drama, or comedy, etc., [27], [28]. Suppose all items in a category are related with the same acceptance probability, and items in different categories are unrelated (i.e., zero acceptance probability). We show that 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 still differ by category. We refer to this special case as COPRA-CAT.

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

**Proof.** Let $K$ be the number of categories We compute a matrix, called cost matrix and denote it by $g$, where entry $g(k, i)$ represents the cost of 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 $w$, where $w(k, s')$ represents the cost of the optimal solution by considering the first $k$
categories using a cache size of $s', s' = 0, 1, \ldots, S$. The value of $w(k, s')$ is computed by the following recursion:

$$w(k, s') = \min_{r=0,1,\ldots,s'} \{ g(k, r) + w(k-1, s' - r) \}.$$  \hfill (5)

Using Equation (5), 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 $w(1, s') = \min_{r=0,1,\ldots,s'} g(1, r)$ for all $r = 0, 1, \ldots, s'$. 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

$$w(l+1, s') = \min_{r} \{ g(l+1, r) + w(l, s' - r) \}.$$  \hfill (6)

The possible values for $r = 0, 1, \ldots, s'$, 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.

The complexity of computing $g$ is of $O(KI)$. By the above, the computational complexity of $w$ is of $O(KS^2)$ where $S$ is up to the number of contents.

5 Greedy Algorithm

A commonly considered strategy for fast but suboptimal solution is a greedy approach (GA) that builds up a solution incrementally. GA goes through time slots one by one. For each time slot, GA examines all content items and finds the item that gives the minimum cost. This process is repeated for the remaining items until no more content can be added to the cache due to cache or backhaul link capacity. The algorithm is shown in Algorithm 1.

Line 1 initializes the variables. Lines 2-25 go through all time slots and content items. Lines 8-13 examine the non-cached item that gives the minimum cost. This process is repeated for the remaining items until no more content can be added to the cache. The resulting ILP, with equality constraints requiring that the duplicate variables are equal to the original ones. The resulting ILP, mathematically, is equivalent to the original ILP. Next, these constraints are relaxed using Lagrangian relaxation and some method (often a subgradient method [40], [41]) is applied to solve the resulting Lagrangian dual. Note that the Lagrangian multipliers act as penalties when the original variables and the duplicate ones take different values. By iteratively updating the multipliers, LD strives to make the difference between the variables as small as possible.

|Algorithm 1. Greedy Algorithm |
|---|
|**Input:** COPRA as defined in (4) |
|**Output:** $y^*, x^*, v^*$ |
|1: $y \leftarrow 0, x \leftarrow 0, v \leftarrow 0$ |
|2: for $t \in T$ do |
|3: $I = \mathcal{I}, L' = L, S' = S$ |
|4: STOP $\leftarrow$ False |
|5: while not STOP do |
|6: STOP $\leftarrow$ True |
|7: $\Delta^* \leftarrow \infty, r^* \leftarrow -1$ |
|8: for $i \in I'$ do |
|9: $i' \leftarrow -1$ |
|10: if $(s_i \leq L')$ then |
|11: STOP $\leftarrow$ False, $(i', a') \leftarrow (i, 0)$ |
|12: else if $(y(t_{i-1}) = 1$ and $s_i \leq S'$ and $x_{ti}A_t \neq 1$) then |
|13: STOP $\leftarrow$ False, $(i', a') \leftarrow (i, a + 1)$ |
|14: if $(i' > -1)$ then |
|15: $c \leftarrow$ The $N$ content items in $\mathcal{I} \setminus I'$ with highest acceptance probability with respect to $i'$ |
|16: $\Delta < \Delta^*$ then $\Delta^* \leftarrow \Delta, (i^*, a^*) \leftarrow (i', a')$ |
|17: else if $(i' > 0)$ then |
|18: $y_{ti} \leftarrow 1, x_{ti}a^* \leftarrow 1$ |
|19: $\mathcal{I}' \leftarrow \mathcal{I}' \setminus \{i\}$ |
|20: if $(a^* = 0)$ then $L' \leftarrow L' - s_i$ |
|21: $S' \leftarrow S' - s_i$ |
|22: $v_{ti} \leftarrow 1$ |
|23: for $i \in I'$ do |
|24: $c \leftarrow$ The $N$ content items in $\mathcal{I} \setminus I'$ with highest acceptance probability with respect to $i$ |
|25: $v_{ti} \leftarrow 1$ |

6.1 Lagrangian Decomposition

In our LD-based algorithm (LDA), we duplicate the $x$ variables. Specifically, we replace $x$ variables in AoI constraints (4b), (4c), and (4d) by $x'$ and add a set of constraints requiring $x = x'$. Next, we relax constraints $x = x'$ with multipliers $\lambda$ and the resulting Lagrangian relaxation is given in (7). Note that $\Delta^*$ update and $\Delta^*$ direct are the same as $\Delta^*$ update and $\Delta^*$ direct but the $x$ variables are replaced by $x'$.

$$\min_{y \in \{0,1\}} \Delta^* \text{ update } + \Delta^* \text{ direct } + \Delta^* \text{ recon } + \sum_{i \in \mathcal{I}} \sum_{a \in A_t} \lambda_{tia} (x'_{tia} - x_{tia})$$  \hfill (7a)

s.t. $\sum_{a \in A_t} x'_{tia} = y_{ti}, t \in T, i \in \mathcal{I}$  \hfill (7b)

$$x'_{tia} \geq y_{ti} + x_{(t-1)(a-1)} - x'_{tia} - 1,$$  \hfill (7c)

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

We propose an algorithm by applying Lagrangian decomposition to ILP (4). In LD, some variables are duplicated, with equality constraints requiring that the duplicate variables are equal to the original ones. The resulting ILP, mathematically, is equivalent to the original ILP. Next, these constraints are relaxed using Lagrangian relaxation and
As can be seen ILP (7) is decomposed into to subproblems, one consists of all terms having \( \mathbf{x}' \), and the other all terms with \( \mathbf{x} \). Below, we formally state each of them.

### 6.2 Subproblem One

Subproblem 1, hereinafter referred to as \( \text{SP}_1 \), is shown in (8). \( \text{SP}_1 \) consists of all terms having \( \mathbf{x}' \), namely the downloading costs and Lagrangian multiplier terms in the objective function, and the constraints related to AoI.

\[
\begin{align*}
\text{SP}_1 : & \quad \min_{\mathbf{y} \in \{0, 1\}} \Delta_{\text{update}}' + \Delta_{\text{direct}}' + \sum_{t \in T} \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}_{ti}} \lambda_{ti} x_{ti} \quad (8a) \\
\text{s.t.} & \quad \sum_{a \in \mathcal{A}_{ti}} x_{ti} = y_{ti}, t \in T, i \in \mathcal{I} \quad (8b) \\
& \quad x_{ti}' \geq y_{ti} + x_{(t-1)(a-1)}' - x_{t(a)}' - 1, \\
& \quad t \in T \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \\
& \quad x_{ti}' \leq x_{(t-1)(a-1)}', \\
& \quad t \in T \setminus \{1\}, i \in \mathcal{I}, a \in \mathcal{A}_{ti} \setminus \{0\} \quad (8d)
\end{align*}
\]

We exploit the structure of \( \text{SP}_1 \) as follows. First, as there is no constraint bundling the content items together, \( \text{SP}_1 \) decomposes by content, leading to \( T \) 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 \( \text{SP}_1 \) 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 the destination is equivalent to solving \( \text{SP}_1(i) \). The graph is shown in Fig. 2. The objective function of \( \text{SP}_1(i) \) is

\[
\begin{align*}
\sum_{t \in T} \left( (c_s - c_h) s_i x_{t0} + \sum_{a \in \mathcal{A}_{ti}} c_h s_i h_{ti} f_{ti} x_{ti}' \right) \\
\sum_{t \in T} \sum_{a \in \mathcal{A}_{ti}} \lambda_{ti} x_{ti}' \\
= \sum_{t \in T} \sum_{t \in T} \sum_{a \in \mathcal{A}_{ti}} (c_s - c_h) s_i x_{t0} + \sum_{t \in T} \sum_{a \in \mathcal{A}_{ti}} (c_h s_i h_{ti} f_{ti} + \lambda_{ti}) x_{ti}' 
\end{align*}
\]

For time slot \( t \), there are 1 \( + \min \{ A_i, t - 1 \} \) vertically aligned nodes. A path passing through node \( V^0_{t0} \) and \( V^1_{t0} \) corresponds to the following two scenarios, respectively:

1) The content is in the cache.
2) The content is in the cache and has AoI \( a, a \in A_i \).

For each node \( V^0_{t0} \), there are two outgoing arcs, one to \( V^0_{(t+1)i} \), 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 \( V^1_{(t+1)i} \) which has weight \( d_{(t+1)i} = (c_s - c_h) s_i + c_h s_i h_{(t+1)i} f_{(t+1)i} + \lambda_{(t+1)i} \) corresponding to the case that the content is downloaded to the cache in the next time slot and has AoI zero. For each node \( V^1_{t0} \), there are three outgoing arcs to \( V^0_{(t+1)i}, V^1_{(t+1)i}, \) and \( V^0_{(t+1)i} \), respectively. A path passing through the first, second, and the third arcs corresponds to the following three scenarios, respectively:

1) The content is deleted for the next time slot with arc weight zero.
2) The content is re-downloaded from the cache and has AoI zero with weight \( d_{(t+1)i} \).
3) The content is kept and its AoI increases with one time unit and has weight \( d_{(t+1)i} = c_h s_i h_{(t+1)i} f_{(t+1)i} + \lambda_{(t+1)i} \).

Finally, there are \( T \) arcs from \( V^0_{T0} \) and \( V^1_{T0} \) to \( D \), each with weight zero.

Given any solution of \( \text{SP}_1(i) \), by construction of the graph, the solution directly maps to a path from the origin to the destination with the same objective function value. Conversely, given a path we construct an ILP solution. For time slot \( t \), if the path contains node \( V^0_{t0} \), we set \( y_{ti} = 0 \). If the path passes through node \( V^1_{t0} \), we set \( y_{ti} = 1 \) and \( x_{t0} = 1 \) (as \( \mathbf{y} \) and \( \mathbf{x}' \) are the variables present in \( \text{SP}_1(i) \)). The resulting ILP solution has the same objective function value as the path length in terms of the total arc weight. Hence the conclusion.

### 6.3 Subproblem Two

Subproblem 2, hereinafter referred to as \( \text{SP}_2 \), consists of all those terms of (7) containing \( \mathbf{z} \). \( \text{SP}_2 \) decomposes by time slot, leading to \( T \) smaller problems. Denote by \( \text{SP}_2(t) \) the problem corresponding to time slot \( t \), shown in (10).

The number of \( v \)-variables in \( \text{SP}_2(t) \) is exponential, as there is an exponential number of recommendation sets. Hence, including all \( v \) variables is impractical. To deal with this issue, we leverage the concept of column generation.

Column generation is a powerful method to obtain the global optimum of some structured linear programs with
exponential number of variables. Note that a variable in any ILP or linear program is characterized by its cost coefficient in the objective function, and its coefficients in the constraint matrix. Here, the term column refers to these coefficients, because they form a column vector in the constraint matrix.

A column generation algorithm starts with a small subset of columns (i.e., decision variables). Hence most of the variables are excluded. The optimization problem with this subset of variables is called the restricted master problem (RMP). Once the RMP is solved (using any linear programming solver), we face the following question: Does there exist a variable outside the RMP, such that adding it to the RMP will improve the total cost? Note that excluding a variable (or equivalently its column) from the RMP has the same effect as including this variable in the RMP but setting its value to be zero. Thus, we can reformulate the question: Is there an excluded variable such that increasing its value from zero will yield improvement? By linear programming theory, this question is answered by computing the reduced cost term of such a variable, and the answer is yes if the reduced cost is negative. To this end, we compute the minimum reduced cost of all excluded variables. This is seemingly impossible, as these variables “do not exist” yet. However, column generation is able to achieve this, by constructing and solving an auxiliary problem, called the pricing problem (PP). PP is constructed such that its optimum solution is the RMP that is solved again. Thus, column generation is an iterative process that alternates between the RMP and the PP. The benefit of column generation is to exploit the fact that at optimum only a few variables are positive. For full details of column generation, we refer to [42].

In our case, we apply column generation to the \( \mathbf{v} \) variables in the linear programming relaxation of \( SP_2^{(l)} \). Hence the RMP is the continuous version of formulation (10), but with a small subset \( C_i \subseteq C \), for any content \( i \in I \). Denote by \( C_i^c \) the cardinality of \( C_i \). Thus a column corresponds to a \( \mathbf{v} \)-variable for recommendation set. PP, presented below, is constructed such that solving the PP to optimum will tell us whether or not there is any \( \mathbf{v} \)-variable that is excluded from the RMP but has negative reduced cost. If so, it is added to the RMP, and column generation moves to the next iteration.

### 6.3.1 Pricing Problem

To ease the presentation, we consider a generic time slot and drop time index \( t \). The PP uses the dual information to generate new variables/columns. Denote by \( \mathbf{v}^* \) the optimal solution of RMP. After obtaining \( \mathbf{v}^* \), we need to check whether \( \mathbf{v}^* \) is the global optimum of RMP. This can be determined by finding the column with the minimum reduced cost for each content \( i \in I \). This means the PP decomposes to \( I \) smaller problems, one corresponding to each content. If all these minimum reduced cost values are nonnegative, the current solution is optimal. Otherwise, we add the columns with negative reduced costs to the RMP.

Consider content \( i \in I \). Denote by \( \pi_i^* \) and \( \beta_i^* = \{ \beta_{ja} | j \in R_i, a \in A_i \} \) the optimal dual values of the counterpart constraints of (10d) and (10e) in the RMP, respectively. Note that the dual variable values are in fact computed in every iteration of the standard simplex method for solving linear programs, and hence the optimal values of them are literally available at optimum. In practice, any optimization software (e.g., [43]) will return these values when the RMP is solved.

Applying linear programming theory, the reduced cost of the \( \mathbf{v} \)-variable of content \( i \) and recommendation set \( c \) is

\[
\begin{align*}
(c_i(1 - \bar{P}_{ic}) + c_s \bar{P}_{ic}) s_i h_i - \pi_i^* + \sum_{j \in R_i} \sum_{a \in A_i} \beta_{ja} = \\
(s_i h_i (c_s - c_i) \bar{P}_{ic}) + c_b s_i h_i - \pi_i^* + \sum_{j \in R_i} \sum_{a \in A_i} \beta_{ja},
\end{align*}
\]

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

\[
\begin{align*}
p' &= \log \left( h_i s_i (c_s - c_i) \prod_{(j,a) \in c} (1 - p_{ja}) \right) \\
&= \log (h_i s_i (c_s - c_i)) + \sum_{(j,a) \in c} \log (1 - p_{ja}).
\end{align*}
\]

Now, the reduced cost can be expressed as

\[
10^{p'} + c_b s_i h_i - \pi_i^* + \sum_{j \in R_i} \sum_{a \in A_i} \beta_{ja},
\]

where \( p' = \log (h_i s_i (c_s - c_i)) + \sum_{(j,a) \in c} \log (1 - p_{ja}) \). As \( p_{ja} \in (0, 1) \), \( \sum_{(j,a) \in c} \log (1 - p_{ja}) \) is negative. Thus, the minimum and maximum values that \( p' \) can take are \( p'_{\text{min}} = \log (h_i s_i (c_s - c_i)) + \sum_{(j,a) \in c} \log (1 - p_{ja}) \), and \( p'_{\text{max}} = \log (h_i (c_s - c_i)) \), respectively. Hence \( p' \in [p'_{\text{min}}, p'_{\text{max}}] \). The above expression is for a given \( \mathbf{v} \)-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 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 AoA \( a \) is in the set to be generated. Then PP\((i)\) can be expressed as (14). Note that the terms \( c_i, s_i, h_i \) and \( \pi_i^* \) are constants here, and hence can be dropped in the optimization process. For this reason we do not include them in (14a). However, they still need to be
accounted for once (14) is solved, as a post-processing step, to obtain the correct reduced cost value. Constraints (14c) ensure that for each content in the recommendation set, exactly one AoI value is selected. Constraint (14d) states that the total number of contents in the recommendation set can not exceed the given upper bound. In the following, we show that PP\(^{(i)}\) can be solved via DP.

PP\(^{(i)}\) : \[
\min_{z \in \{0,1\}^p} \ 10p' + \sum_{j \in R_i} \sum_{a \in A_j} \beta_{ija} z_{ja}
\]  
\(\text{s.t.} \quad p' = \log (h_is_i(c_s - c_b)) + \sum_{j \in R_i} \sum_{a \in A_j} \log (1 - p_{ija}) z_{ja}
\]

(14a)

(14b)

\[\sum_{a \in A_j} z_{ja} \leq 1, j \in R_i \]

(14c)

\[\sum_{j \in R_i} \sum_{a \in A_j} z_{ja} \leq N
\]

(14d)

\[p' \in [p'_{min}, p'_{max}]
\]

(14e)

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 (14b) can be stated equivalently as the following inequality constraint, where \(p'' = \log (h_is_i(c_s - c_b)) - p'.\)

\[\sum_{j \in R_i} \sum_{a \in A_j} -\log (1 - p_{ija}) z_{ja} \geq p''
\]

(15)

Since \(p' \in [p'_{min}, p'_{max}],\) the minimum and maximum values that \(p''\) can take are zero and \(\sum_{j \in R_i} \sum_{a \in A_j} -\log (1 - p_{ija})\), respectively. 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 (large) positive integer \(M\) and rounding. Let \(W = \lceil M \sum_{j \in R_i} \sum_{a \in A_j} -\log (1 - p_{ija}) \rceil\) be the maximum possible value of \(p''\) after multiplying it by \(M\) and rounding. Similarly let \(q_{ja} = \lceil -M \log (1 - p_{ija}) \rceil\) for \(j \in R_i, \ a \in A_j\). Thus, \(p''\) is now restricted to be an integer in interval \([0, W]\).

After these two steps, (14) can be re-expressed as (16). Formulation (16) resembles an inversed multiple-choice knapsack problem with an upper bound (16d) on the number of items. The difference is that we have \(p'\) and \(p''\) as two additional variables and they differ by a constant in their values. There is a term in the objective function involving \(p'\), whereas \(p''\) affects the right-hand side of the knapsack constraint, 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 for some given capacity, but also those for all intermediate capacity values starting from zero. This implies that by setting the right-hand side of (16b) to be \(W\), one computation using DP is enough to examine the effect of all possible \(p''\) values.

Then the optimum can be obtained by post-processing considering the function term with \(p'.\)

\[
\min_{z \in \{0,1\}^p, p'} 10p' + \sum_{j \in R_i} \sum_{a \in A_j} \beta_{ija} z_{ja}
\]

\(\text{s.t.} \quad \sum_{a \in A_j} z_{ja} \geq p''
\]

(16a)

(16b)

(16c)

(16d)

\(p'' = \log (h_is_i(c_s - c_b)) - p'
\]

(16e)

(16f)

The DP algorithm for solving PP\(^{(i)}\) is shown in Algorithm 3. Lines 1-5 are the initialization steps. Lines 6-13 solve (16) with maximum capacity \(W\), where matrix \(B'\) 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 an integer knapsack capacity \(w \in [0, W]\). Matrix \(A'\) is an auxiliary matrix that stores the AoI corresponding to the optimum for each tuple \((w, j, n)\) where \(w \in [0, W], \ j = 1, \ldots, R_i, \ \text{and} \ n = 1, 2, \ldots, \min\{j, N\}.\) Lines 14-35 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 far, in order to find the global minimum of the problem. The complexity of this algorithm is of \(O(WR, NA_1)\). The column generation algorithm for solving SP\(^{(i)}\) is shown in Algorithm 2 in which Algorithm 3 is used for solving PP\(^{(i)}\), \(i \in I\).

Algorithm 2. Column Generation for SP\(^{(i)}\)

\textbf{Input:} SP\(^{(i)}\) as defined in (10)

\textbf{Output:} Optimal values of \(x, y, \) and \(z\) for the continuous relaxation of (10) defined for \(C_i, \forall i \in I\)

1: Initialize \(C_i\) for \(i \in I\)

2: Stop \(\leftarrow\) False

3: \textbf{while} Stop \(\leftarrow\) False \textbf{do}

4: Solve RMP (10) and obtain dual optimum values \(\pi\) and \(\mu\)

5: Stop \(\leftarrow\) True

6: for \(i \in I\) do

7: Solve PP\(^{(i)}\) by Algorithm 3 and obtain OPT and \(c'\)

8: if OPT \(\leq 0\) \textbf{then}

9: Stop \(\leftarrow\) False

10: Add \(c'\) to \(C_i\)

6.4 Attaining Integer Feasible Solutions

The solutions of the two subproblems will likely violate some original constraints. The solution of SP\(_1\) may violate the cache and backhaul capacity limits, as constraints (4g) and (4h) are not present in SP\(_1\). SP\(_2\) decomposes by time slot, and because the constraints formulating AoI evolution over time, namely (4b), (4c), and (4d), are not in SP\(_2\) but in SP\(_1\), the AoI values across time slots in the solution of SP\(_2\) may be inconsistent (i.e., the AoI of a cached content may grow more than one in two consecutive time slots).
The idea of obtaining feasible solutions to the original problem is to “repair” a solution that violates some of the problem constraints. The question is then if one shall repair the solution of SP₁ or that of SP₂. We present an approach for repairing based on SP₂. The reason for using SP₂ is that its solution contains decisions of recommendation sets. In contrast, in SP₁ there is no decision related to recommendation and hence its caching solution has no associated recommendation solution.

Algorithm 3. Dynamic Programming for PP⁽ⁱ⁾  

**Input:** \( bᵢ, πᵢ, N, Rᵢ, cᵢ, sᵢ, hᵢ, q, W, M \)  
**Output:** Optimal objective function value \( OPT \) and optimal recommendation set \( c^* \)  

1: Create matrix \( B' \) of size \( (1 + W) \times (1 + Rᵢ) \times (1 + N) \)  
2: Create matrix \( A' \) of size \( (1 + W) \times (1 + Rᵢ) \times (1 + N) \)  
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: \( w \leftarrow 1, \text{Stop} \leftarrow \text{False} \)  
6: while \( \text{Stop} = \text{False} \) do  
   7: for \( j = 1, \ldots, Rᵢ \) do  
      8: for \( n = 1, \ldots, \min\{j, N\} \) do  
         9: \( B'(w, j, n) \leftarrow \min\{B'_{ja} + B'(w', j, n - 1), B'(w, j - 1, n')\} \)  
         10: \( A'(w, j, n) \leftarrow \arg\min\{B'_{ja} + B'(w', j, 1, n - 1), B'(w, j - 1, n')\} \)  
         11: where \( w' = \max(0, w - q_{ja}) \) and \( n' = \min\{j - 1, n\} \)  
      12: if \( B'(w, Rᵢ, N) = \infty \) or \( w = W \) then  
         13: \( \text{Stop} \leftarrow \text{True} \)  
      14: OPT \leftarrow \infty, c^* \leftarrow \emptyset, w \leftarrow 1, \text{Stop} \leftarrow \text{False} \)  
   15: while \( \text{Stop} = \text{False} \) do  
      16: \( q^* \leftarrow 0, w' \leftarrow w, j' \leftarrow Rᵢ, n' \leftarrow N \)  
      17: if \( B'(w', j', n') = \infty \) or \( w > W \) then  
         18: \( \text{Stop} \leftarrow \text{True} \)  
      19: else  
         20: \( c \leftarrow \emptyset \)  
         21: while \( j' \geq 1 \) and \( n' \geq 1 \) do  
            22: if \( B'(w', j', n') < B'(w', j' - 1, n') \) then  
               23: \( a^* \leftarrow A'(w', j', n') \)  
               24: \( w' \leftarrow \min(0, w' - q_{ja}) \)  
               25: \( q^* \leftarrow q^* + q_{ja} \)  
               26: \( c \leftarrow c \cup \{j', a^*\} \)  
               27: \( n' \leftarrow n' - 1 \)  
            28: else  
               29: \( n' \leftarrow \min\{j' - 1, N\} \)  
               30: \( j' \leftarrow j' - 1 \)  
               31: \( p \leftarrow \log\left(\frac{h_i s_i}{c_i - c_j}\right) - q^*/M \)  
               32: if \( 10^p + B'(w, Rᵢ, N) < \text{OPT} \) then  
                  33: OPT \leftarrow \text{OPT} + c^* s_i h_i - \pi_i^* + B'(w, Rᵢ, N) \)  
                  34: \( c^* \leftarrow c \)  
               35: \( w \leftarrow w + 1 \)  

For SP₂, the solution does not respect the AoI evolution of contents across the time slots. The repairing algorithm (RA) is shown in Algorithm 4 that consists of three main steps. In the algorithm, symbol \( \leftarrow \) is used to indicate the assignment of a value. Symbol \( \equiv \) is used to indicate that an assigned value of an optimization variable is kept fixed subsequently.

Algorithm 4. RA for Constructing Integer Solutions  

**Input:** Output of Algorithm 2 for \( SP₂⁽ⁱ⁾, t \in T \)  
**Output:** An integer solution for COPRA  

1: for \( t \in T \) do  
   2: while (exists \( yᵢ \) with fractional value) do  
      3: \( yᵢ \leftarrow 1 \) if \( yᵢ \equiv 1 \)  
      4: \( g \leftarrow \max\{yᵢ : 0 < yᵢ < 1\} \)  
      5: \( j \leftarrow \arg\max\{yᵢ : 0 < yᵢ < 1\} \)  
      6: \( \Phi \leftarrow \{t \in \text{if } (t = 1) \text{ else } \Phi \} \) \( \) else \( yᵢ \equiv 0 \)  
      7: \( yᵢ \leftarrow 1 \) if \( s_j + \sum_{i=1}^{w} s_i \leq \Phi \) else \( yᵢ \equiv 0 \)  
   8: Solve \( SP₂⁽ⁱ⁾ \) \( \)  
   9: \( y \leftarrow \{yᵢ : t \in T, i \in T\} \)  
  10: Solve formulation (17) and obtain the values of \( y \)  
  11: for \( t \in T \) do  
   12: for \( i \in T : yᵢ = 0 \) do  
      13: \( c \leftarrow \{a \in Aᵢ \} \) with the highest acceptance probability with respect to \( c \)
Algorithm 5. The Main Steps of LDA

Input: COPRA as defined in (4)
Output: $\hat{w}$ (total cost of the best solution found)
1: Initialize $Q, \epsilon_1$, and $\epsilon_2$
2: $\lambda \leftarrow 0, k \leftarrow 1$
3: LBD $\leftarrow 0, \hat{w} \leftarrow \infty$
4: repeat
5: Solve SP$_{i}^{(k)}$ for $i \in I$ and obtain $x^{(k)}$
6: Solve SP$_{t}^{(k)}$ for $t \in T$ by Algorithm 2 and obtain $x^{(k)}$
7: Calculate $L(\lambda^{(k)})$ which is (7a)
8: if $L(\lambda^{(k)}) >$ LBD then LBD $\leftarrow L(\lambda^{(k)})$
9: Apply Algorithm 4 to obtain an integer solution and its objective function value $U$
10: if $U < \hat{w}$ then $\hat{w} \leftarrow U$
11: Calculate $\lambda^{(k+1)} = \lambda^{(k)} + \delta^{(k)}d^{(k)}$ where $\delta^{(k)} = \eta \frac{\bar{w} - L(\lambda^{(k)})}{\|d^{(k)}\|}$
12: $k \leftarrow k + 1$
13: until $\|d^{(k)}\| \leq \epsilon_1$ or $\|\lambda^{(k+1)} - \lambda^{(k)}\| \leq \epsilon_2$ or $k > Q$

6.5 Algorithm Summary
The main steps of LDA are shown in Algorithm 5. Line 1 initializes the total number of iterations $Q$ to perform, and tolerance parameters $\epsilon_1$ and $\epsilon_2$. Lines 2 and 3 initialize the vector of Lagrangian multipliers $\lambda$, the iteration counter $k$, the lower bound LBD, and the best found solution $\hat{w}$. Lines 5 and 6 solve the SP$_{i}^{(k)}$ for $i \in I$ and SP$_{t}^{(k)}$ for $t \in T$, respectively. Lines 7 and 8 calculate the Lagrangian dual function value, and update the LBD if a higher better bound is found. Line 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. Notation related to LDA is summarized in Table 3.

In Algorithm 5, Line 11 uses a subgradient formula to update the Lagrangian multipliers. This formula involves the use of $\hat{w}$ that is the cost of the best known feasible solution to our problem COPRA, found via the repairing process. As the Lagrangian multipliers appear in the objective function of SP$_{2}$, the repairing process does affect the solution of SP$_{2}$ in the next iteration, even though the solution corresponding to $\hat{w}$ is not used in the subproblem.

7 PERFORMANCE RESULTS
In this section, we evaluate the performance of LDA and GA. The problem setup is new and there is no existing algorithm for this problem. However, we compare the results of LDA and GA to the global optimum (for small-size problem instances) and the LBD obtained from the LDA (for large-size problem instances). More specifically, 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 (4). 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 that the $i$th content is requested is $\frac{i^{-\gamma}}{\sum_{i \in I} i^{-\gamma}}$ [44], [45]. Here $\gamma$ is the shape parameter and it is set to $\gamma = 0.56$ [44]. The sizes of content items are generated within interval $[1,10]$. We have set the cache capacity to 50% of the total size of content items, i.e., $S = 0.5 \sum_{i \in I} s_i$. The capacity of backhaul link is set to $L = \rho \sum_{i \in I} s_i$ where parameter $\rho$ 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 Aol (i.e., parameter $A$) that a content can take is set to two. Intuitively, with higher value of $a_i$ the probability of accepting content $j$ instead of content $i$ decreases. Therefore, $p_{ja}$ is a non-increasing function in $a$. In the simulation we have used linear functions $p_{ja} = 1 - \eta_{ij}a - \epsilon$ in which $\eta_{ij}$ is a coefficient indicating how the probability decreases when $a$ increases and $\epsilon$ is a small positive number.

We use content-specific and time-specific functions including linear and nonlinear ones from the literature [46], [47] to model the Aol cost of content items. Specifically, for each content, one of the following functions is randomly selected: $f_{i} = 1 + a_{i}a_{a}$, $f_{i} = \frac{1}{1-a_{a}}$, and $f_{i} = e^{a_{a}a_{a}}$. The functions are made content-specific and time-specific by varying parameter $a_{a}$. 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. Parameters $c_{0}$ and $c_{s}$ model downloading costs of unit data from the base station and server, respectively. In the simulation $c_{0}$ and $c_{s}$ are set to 1 and 10, respectively. We

| Notation | Definition |
|----------|------------|
| $x_{i}^{j}$ | Counterpart of $x_{i}^{j}$ in subproblem I |
| $\lambda_{i}$ | Lagrangian multiplier |
| $O$ | A node representing the origin of shortest path |
| $D$ | A node representing the destination of shortest path |
| $v_{t}^{i}$ | A node representing content $i$ that is not in the cache in time slot $t$ |
| $v_{t}^{i}$ | A node representing content $i$ that is in the cache with AoI $a$ in time slot $t$ |
| $d_{i0}$ | Weight of the arc coming from node $O$ to node $V_{i}^{0}$ |
| $d_{a0}$ | Weight of the arc coming from node $V_{i(t-1)}^{a}$ for any possible $a$ or $V_{i(t-1)}^{a}$ to node $V_{i}^{a}$ for $t > 0$ |
| $d_{tia}$ | Weight of the arc coming from node $V_{i(t-1)}^{a}$ to node $V_{i}^{a}$ for $a > 0$ and $t > 0$ |
| $x_{i}^{*}$ | The optimal dual value of the counterpart of constraints (10a) |
| $\beta_{ja}$ | The optimal dual value of counterpart of constraints (10e) |
| $p_{min}$ | The minimum value that $p'$ can take |
| $p_{max}$ | The maximum value that $p'$ can take |
| $z_{ja}$ | A binary variable that takes value one if and only if content $j$ with AoI $a$ is in the set to be generated |
| $M$ | A large number |
| $W$ | Knapsack capacity |
| $q_{ja}$ | Size of item $j$ with AoI $a$ in PP |
will vary parameters $I$, $T$, $\rho$, and $A$, and study their impact on the overall cost and algorithm performance. Notation related to simulation setup is summarized in Table 4.

Fig. 3 shows the total cost returned by LDA when recommendation is utilized, and LDA with no recommendation (denoted by LDA-NC). The blue and pink lines show the total cost with and without recommendation, respectively. The light and dark green lines show their respective optimum costs. Note that in this case we have set $p_{i,j} = 0$ for any $i$, $j$, and $a$. This indicates that the probability that content $j$ is accepted instead of content $i$ is zero, and hence any requested content $i$ is downloaded from the server if it is not available in the cache. Hence, the cost corresponds to not deploying recommendation at all. Fig. 3 shows that, interestingly, the total cost decreases by about 50% with recommendation. Another interesting point is that the reduction is even more apparent when the number of content items increases. From this result, the consideration of recommendation optimization is relevant.

Figs. 4, 5, and 6 and Figs. 7, 8, and 9 show the performance results for the small-size and large-size problem instances, respectively. In Figs. 4, 5, and 6, the green line represents the global optimum computed using ILP (4). In Figs. 7, 8, and 9, the black line represents the LBD obtained from LDA. In all these 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 with each other.

Fig. 4 shows the impact of content items on the total cost for small-size problem instances. The overall cost decreases with the number of contents. This is due to the fact that the capacity of cache is set relatively to the total size of contents. Namely, with larger number of contents, more capacity is available, and hence more opportunity to serve content requests from the cache. This effect can also be seen for large problem instances, see Fig. 7. For small-size problems, the optimality gap of GA is about 35%, while for LDA it is about 9% from global optimum. For large-size problems, the performance of both LDA and GA improves with the number of contents. The reason is due to fact that the number of requests is the same for all values of $I$, and the capacities of cache and backhaul link increase with $I$ (as they are set relatively to the total size of contents). Thus, most of the requests can be satisfied from the cache.

Fig. 5 shows the impact of time slots for small-size problem instances. As can be seen, the cost increases with the 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 35%, while for LDA the gap is only 10%. The results for large-size problems are shown in Fig. 8. LDA consistently shows good performance, whereas the results of GA are much more sub-optimal. It is worth noting that the optimality gaps of both LDA and GA decrease for large problem instances. The reason is that cache and backhaul capacities are larger for the large problem instances, and hence problems become easier to solve.

Fig. 6 shows the impact of $\rho$ on the total cost. Larger $\rho$ means higher backhaul capacity. The costs of both LDA and GA decrease sharply when $\rho$ increases from 10% to 20%, then the decrease slows down due to a saturation effect. The optimality gap of LDA is 3.5% when $\rho = 10\%$. When $\rho$ increases to 20%, the cost significantly decreases and the optimality gap decreases as well to 2.4%. For higher value of $\rho$, the gap stays around 4%. For GA the deviation from optimality is high no matter $\rho$ is small or not. Similar trends can be seen for large-size problems, see Fig. 9.

Fig. 10 shows the trade-off between the total cost and maximum allowed AoI (i.e., $A$) in the system. By observing the lower bound, we can see that the total cost tends to decrease by allowing higher $A$ for the cached contents. However, this comes at the price of larger AoI cost, as high

### Table 4

| Notation | Value |
|----------|-------|
| $T$ | Generated within interval [4, 16] |
| $I$ | Generated within interval [16, 60] |
| $c_b$ | 1 |
| $c_a$ | 10 |
| $N$ | 5 |
| $s_i$ | Generated randomly in interval [1, 10] |
| $h_{ti}$ | Generated based on ZipF distribution |
| $\gamma$ | Shape parameter of ZipF distribution, set to 0.56 |
| $S$ | Set to 0.5 $\sum_{i \in I} s_i$ |
| $\rho$ | Backhaul capacity in relation to the total size of contents, with value in interval [0.1, 0.5] |
| $L$ | $\rho \sum_{i \in I} s_i$ |
| $f_{i,a}$ | Set to one of functions $1 + \alpha_i, a \cdot e^{-\alpha_i a},$ and $e^{\alpha_i a}$ where $\alpha_i$ is generated randomly |
| $p_{i,0}$ | Generated randomly in interval [0.6, 1] |

---

**Fig. 3.** Comparison to caching without recommendation for $T = 6$, $S = 0.5 \sum_{i \in I} s_i$, $L = 0.3 \sum_{i \in I} s_i$, and $\gamma = 0.56$.

**Fig. 4.** Impact of $I$ on total cost when $T = 6$, $S = 0.5 \sum_{i \in I} s_i$, $L = 0.3 \sum_{i \in I} s_i$, and $\gamma = 0.56$. 

---

AHANI AND YUAN: OPTIMAL CONTENT CACHING AND RECOMMENDATION WITH AGE OF INFORMATION 701
AoI lowers user experience. That the overall cost decreases only slightly for $A/C_21^2$ manifests this trade-off. The performance of LDA degrades somewhat with higher value of $A$. The reason is that the problem becomes much more difficult due to its combinatorial nature. For GA, allowing a higher AoI limit mitigates its sub-optimality. This is because cached contents by suboptimal caching decisions cannot be all updated due to backhaul capacity, and thus some of them become useless with small AoI limit. Allowing a higher AoI limit makes such contents remain useful. Note that LDA still outperforms GA noticeably.

Fig. 11 shows the performance of LDA and GA with respect to the time horizon. As can be seen, LDA performs consistently even for large $T$ while GA’s performance degrades significantly for $T \geq 60$. Note that for LBD and LDA, the curves tend to be linear, hence the average deviation from optimality of LDA is consistent with respect to $T$.

In addition to the total cost shown in the figures, we have examined the caching and recommendation solutions returned by the algorithms. Both algorithms tend to select small-size contents with many requests for caching. Except from this observation, there is no clear pattern in the specific caching decisions made over the time slots. Some additional findings are presented and discussed below.

- About 40% and 30% of the requests are served by the cache in the solutions by LDA and GA, respectively. Thus LDA is able to utilize the cache more efficiently.
- About 50% and 33% of the requests are satisfied via recommendation in the solutions by LDA and GA, respectively. This is probably due to that LDA considers caching and recommendation jointly, such that the recommendations are more likely to be accepted.
The average Aol of cached contents for LDA and GA are 0.65 and 0.38, respectively. The reason is that GA, by its construction, makes the caching decision from scratch for each time slot, no matter the previously cached contents still have low Aol or not. In fact, 0.65 is a low value of Aol as it means that, in average, a content is stored in less than one time slot before it is refreshed.

8 CONCLUSION

We have studied optimal scheduling of cache updates where Aol of contents and recommendation are jointly taken into account. With both Aol and recommendation, the problem is hard even for one single time slot. We formulated the problem as an integer liner program (ILP). Solving the ILP via an optimization solver provides optimal solutions, but it is not practical for 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. 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.

A natural extension of the current work is caching optimization in multiple cells. To this end, we believe the consideration of multiple time scales for caching and user-cell association, studied in [48], is of clear relevance. Here, caching and association are two intertwined problems of which the system dynamics take place at different time scales. Incorporating the key aspects of our current work, namely recommendation and Aol, in such dynamic scenarios forms an interesting line of future work.

REFERENCES

[1] D. Liu, B. Chen, C. Yang, and A. F. Molisch, “Caching at the wireless edge: Design aspects, challenges, and future directions,” IEEE Commun. Mag., vol. 54, no. 9, pp. 22–28, Sep. 2016.
[2] C. A. Gomez-Uribe and N. Hau, “The Netflix recommender system: Algorithms, business value, and innovation,” ACM Trans. Manage. Inf. Syst., 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 Proc. 10th ACM SIGCOMM Conf. Internet Meas., 2010, pp. 404–410.
[4] L. E. Chatziileftheriou, M. Karaliopoulos, and I. Koutsopoulos, “Jointly optimizing content caching and recommendations in small cell networks,” IEEE Trans. Mobile Comput., vol. 18, no. 1, pp. 125–138, Jan. 2019.
[5] L. E. Chatziileftheriou, M. Karaliopoulos, and I. Koutsopoulos, “Caching-aware recommendations: Nudging user preferences towards better caching performance,” in Proc. IEEE Conf. Comput. Commun., 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 J. Sel. Areas Commun., vol. 36, no. 6, pp. 1300–1313, Jun. 2018.
[7] L. Song and C. Fragouli, “Making recommendations bandwidth aware,” IEEE Trans. Inf. Theory, vol. 64, no. 11, pp. 7031–7050, Nov. 2018.
[8] M. Costantini, T. Spyropoulos, T. Giannakas, and P. Sermpezis, “Approximation guarantees for the joint optimization of caching and recommendation,” in Proc. IEEE Int. Conf. Commun., 2020, pp. 1–7.
[9] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, “Age-optimal constrained cache updating,” in Proc. IEEE Int. Symp. Inf. Theory, 2017, pp. 141–145.
[10] S. Kaul, R. Yates, and M. Gruteser, “Real-time status: How often should one update?,” in Proc. IEEE Conf. Comput. Commun., 2012, pp. 2737–2735.
[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 Proc. 10th Int. Conf. Wireless Commun. Signal Process., 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 Proc. IEEE Int. Symp. Inf. Theory, 2017, pp. 136–140.
[13] G. Ahani and D. Yuan, “Accounting for information freshness in scheduling of content caching,” in Proc. IEEE Int. Conf. Commun., 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 Proc. 18th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw., 2020, pp. 1–6.
[15] G. Ahani, D. Yuan, and S. Sun, “Optimal scheduling of age-centric caching: Tractability and computation,” IEEE Trans. Mobile Comput., vol. 21, no. 8, pp. 2939–2954, Aug. 2022.
[16] M. Bastopcu and S. Ulukus, “Maximizing information freshness in caching systems with limited cache storage capacity,” in Proc. 54th Asilomar Conf. Signals, Syst. Comput., 2020, pp. 423–427.
[17] M. Bastopcu and S. Ulukus, “Information freshness in cache updating systems,” IEEE Trans. Wireless Commun., vol. 20, no. 3, pp. 1861–1874, Mar. 2021.
[18] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger, “Age-optimal constrained cache updating,” in Proc. IEEE Int. Symp. Inf. Theory, 2017, pp. 141–145.
[19] S. Traverso et al., “TailGate: Handling long-tail content with a little help from friends,” in Proc. 21st Int. Conf. World Wide Web, 2012, pp. 151–160.
[20] R. D. Yates, “The age of gossip in networks,” in Proc. IEEE Int. Symp. Inf. Theory, 2021, pp. 2984–2989.
[21] B. Abolhassani, J. Tadrous, A. Eryilmaz, and E. Yeh, “Fresh caching for dynamic content,” in Proc. IEEE Conf. Comput. Commun., 2021, pp. 1–10.
[22] B. Buyukates, M. Bastopcu, and S. Ulukus, “Age of gossip in networks with community structure,” in Proc. IEEE Int. Workshop Signal Process. Adv. Wireless Commun., 2021, pp. 326–330.
[23] C. Kam, S. Kompella, and A. Ephremides, “Age of incorrect information for remote estimation of a binary Markov source,” in Proc. IEEE Conf. Comput. Commun. Workshops, 2020, pp. 1–6.
[24] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, “Age of information: An introduction and survey,” IEEE J. Sel. Areas Commun., vol. 39, no. 5, pp. 1183–1210, May 2021.
[25] S. Kastanakis, P. Sermpezis, V. Kotrotsis, D. S. Menasche, and T. Spyropoulos, “Network-aware recommendations in the wild: Methodology, realistic evaluations, experiments,” IEEE Trans. Mobile Comput., vol. 21, no. 7, pp. 2466–2479, Jul. 2022.
[27] Y. Fu, K. N. Doan, and T. Q. Quek, “On recommendation-aware content caching for 6G: An artificial intelligence and optimization empowered paradigm,” Digit. Commun. Netw., vol. 6, no. 3, pp. 304–311, 2020.

[28] 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 Trans. Wireless Commun., vol. 20, no. 10, pp. 6758–6771, Oct. 2021.

[29] K. Guo and C. Yang, “Temporal-spatial recommendation for caching at base stations via deep reinforcement learning,” IEEE Access, vol. 7, pp. 58519–58532, 2019.

[30] T. Giannakas, P. Sermepeis, A. Giovanidis, T. Spyropoulos, and G. Arvanitakis, “Fairness in network-friendly recommendations,” in Proc. IEEE 22nd Int. Symp. a World Wireless, Mobile Multimedia Netw., 2021, pp. 71–80.

[31] D. Liu and C. Yang, “A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users,” IEEE Access, vol. 7, pp. 83120–83136, 2019.

[32] M. Costantini and T. Spyropoulos, “Impact of popular content relational structure on joint caching and recommendation policies,” in Proc. Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Networks, 2020, pp. 1–8.

[33] M. Garetto, E. Leonardi, and G. Neglia, “Similarity caching: Theory and algorithms,” in Proc. IEEE Conf. Comput. Commun., 2020, pp. 526–535.

[34] J. Zhou, O. Simeone, X. Zhang, and W. Wang, “Adaptive offline and online similarity-based caching,” IEEE Netw. Lett., vol. 2, no. 4, pp. 175–179, Dec. 2020.

[35] D. Liu and C. Yang, “A learning-based approach to joint content caching and recommendation at base stations,” in Proc. IEEE Glob. Commun. Conf., 2018, pp. 1–7.

[36] N. Zhang, K. Zheng, and M. Tao, “Using grouped linear prediction and accelerated reinforcement learning for online content caching,” in Proc. IEEE Int. Conf. Commun. Workshops, 2018, pp. 1–6.

[37] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” IEEE Internet Comput., vol. 7, no. 1, pp. 76–80, Jan./Feb. 2003.

[38] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” Adv. Artif. Intell., vol. 2009, 2009, Art. no. 4.

[39] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co., 1990.

[40] J. L. Goffin, “On convergence rates of subgradient optimization methods,” Math. Program., vol. 13, pp. 329–347, 1977.

[41] M. S. Bazaraa and H. D. Sherali, “On the choice of step size in subgradient optimization,” Eur. J. Oper. Res., vol. 7, no. 4, pp. 380–388, 1981.

[42] M. E. Lubbecke and J. Desrosiers, “Selected topics in column generation,” Operations Res., vol. 53, no. 6, pp. 1007–1023, 2005.

[43] Gurobi Optimizer, version 9.5, 2021. [Online]. Available: https://www.gurobi.com/products/gurobi-optimizer/

[44] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, “FemtoCaching: Wireless content delivery through distributed caching helpers,” IEEE Trans. Inf. Theory, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[45] G. Ahani and D. Yuan, “Optimal scheduling of content caching subject to deadline,” IEEE Open J. Commun. Soc., vol. 1, pp. 293–307, Mar. 2020.

[46] 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 Trans. Inf. Theory, vol. 63, no. 11, pp. 7492–7508, Nov. 2017.

[47] Y. Sun and B. Cyr, “Sampling for data freshness optimization: Non-linear age functions,” J. Commun. Netw., vol. 21, no. 3, pp. 204–219, 2019.

[48] T. Zhang, Y. Wang, W. Yi, Y. Liu, C. Feng, and A. Nallanathan, “Two time-scale caching placement and user association in dynamic cellular networks,” IEEE Trans. Commun., vol. 70, no. 4, pp. 2561–2574, Apr. 2022.

Ghafour Ahani received the BSc and MSc degrees in applied mathematics from the University of Kurdistan, Sanandaj, Iran, in 2008 and 2010 respectively. He is currently working toward the PhD 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 (Senior Member, IEEE) received the MSc degree in computer science and engineering, and PhD degree in optimization with Linköping Institute of Technology, in 1996 and 2001, respectively. After his PhD, he has been associate professor and then full professor with 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. He has been guest professor with the Technical University of Milan (Politecnico di Milano), Italy, in 2008, and senior visiting scientist with Ranplan Wireless Network Design Ltd, U.K., 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 with the University of Maryland, College Park, Maryland. 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.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdI.