Optimal Scheduling of Age-centric Caching: Tractability and Computation

Ghafour Ahani, Di Yuan, *Senior Member, IEEE,* and Sumei Sun, *Fellow, IEEE*

Abstract—The notion of age of information (AoI) has become an important performance metric in network and control systems. Information freshness, represented by AoI, naturally arises in the context of caching. We address optimal scheduling of cache updates for a time-slotted system where the contents vary in size. There is limited capacity for the cache for making updates. Each content is associated with a utility function that depends on the AoI and the time duration of absence from the cache. For this combinatorial optimization problem, we present the following contributions. First, we provide theoretical results of problem tractability. Whereas the problem is NP-hard, we prove solution tractability in polynomial time for a special case where all contents have the same size, by a reformulation using network flows. Second, we derive an integer linear formulation for the problem, of which the optimal solution can be obtained for small-scale scenarios. Next, via a mathematical reformulation, we derive a scalable optimization algorithm using repeated column generation. In addition, the algorithm computes a bound of global optimum, that can be used to assess the performance of any scheduling solution. Performance evaluation of large-scale scenarios demonstrates the strengths of the algorithm in comparison to a greedy schedule. Finally, we extend the applicability of our work to cyclic scheduling.

Index Terms—age of information, caching, optimization, scheduling.

1 Introduction

The research interest in age of information (AoI) has been rapidly growing in the recent years. The notion of AoI has been introduced for characterizing the freshness of information [1]. AoI is defined as the amount of time elapsed with respect to the time stamp of the information received in the most recent update. The AoI grows linearly between two successive updates. For a wide range of control and network systems, e.g., status monitoring via sensors, AoI has become an important performance metric.

The AoI aspect arises naturally in the context of caching for holding content items with dynamic updates [2]. Consider a local cache for which updates of content items take place via a network such as a backhaul link. Rather than obtaining content items via the backhaul network, users can download them from the cache if the content items of interest are in the cache, thus reducing network resource consumption. Due to limited backhaul network capacity, not all content items can be updated simultaneously to the cache. Hence, at a time point, the performance of caching depends on not only the cached content items, but also how old these items are; the latter can be characterized by AoI.

We address optimal scheduling of cache updates, where the utility of the cache is based on AoI. The system under consideration is time-slotted, with a given scheduling horizon. The content items are of different sizes. Updating the cache in a time slot is subject to a capacity limit, constraining which content items that can be downloaded in the same time slot. Moreover, the cache itself has a capacity. For each content item, there is a utility function that is monotonically decreasing in the AoI, and because the content items differ in popularity, the utility function is allowed to be content-specific and time-specific. Moreover, when adding a content, the system distinguishes between if this content was cached in the previous time slot, or it has been absent and for how long. The former has lower utility as the incremental change in the content itself is smaller. For every time slot, the optimization decision consists of the selection of the content items to be updated in the cache, subject to the network and cache capacity constraints. Thus the problem falls into the domain of combinatorial optimization. The objective is to find the schedule maximizing the total utility over the scheduling horizon.

Our work consists in the following contributions toward understanding and solving the outlined AoI-driven cache optimization problem (ACOP).

- We provide theoretical results of problem tractability. Specifically, for the special case where all contents have the same size and no cost of absence, the global optimum of ACOP admits polynomial-time tractability (even if the problem remains combinatorial optimization). We establish this result via mapping the problem to a graph, and proving that the optimal schedule is a minimum-cost flow in the graph. For non-uniform content size, the problem is NP-hard due to the knapsack structure.
- We derive an integer linear programming (ILP) formulation for ACOP in its general form, enabling the use of off-the-shelf optimization solvers to approach the problem. This is particularly useful for examining the performance of sub-optimal solutions for small-scale scenarios, for which the global optimum is computed via ILP.
- We derive a mathematical reformulation, by considering sequences representing the evolution of the AoI over time for each content item. The reformulation enables a scalable solution approach. Specifically, we present a column generation algorithm that addresses the linear programming

- *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).*
- *S. Sun is with the Institute for Infocomm Research, Singapore (e-mail: sunsm@i2r.a-star.edu.sg).*

(LP) version of the reformulation, by keeping only a small subset of all possible sequences and augmenting the subset based on the optimality condition. To obtain integrality, we present a rounding concept based on disjunctions rather than on fractional variables, such that column generation is repeated for improvement after rounding. Performance evaluation demonstrates the strengths of the repeated column generation algorithm in comparison to a greedy schedule. Moreover, as the algorithm provides an optimality bound in addition to a problem solution, it is possible to gauge performance even if the global optimum is out of reach. Using the bound, our results show the algorithm consistently yields near-to-optimal solutions for large-scale problem instances.

- Finally, we discuss the extension of our results to cyclic scheduling, i.e., the schedule is repeated in a cyclic manner. In this case, the AoI values at the beginning of the schedule are determined by the caching updates made later in the schedule. We present adaptations, such that the tractability analysis, the ILP formulation, and the repeated column generation algorithm all remain applicable.

2 Related Work

The notion of AoI was introduced in [1]. Generalizations to multiple information sources have been studied in [3]. The aspect of queue management has been introduced in [4], [5], [6]. Early application scenarios of AoI include channel information [7] and energy harvesting [8]. In [9], the authors address optimal update policy for AoI, and provide conditions under which the so called zero-wait policy is optimal. In [10], the authors have considered AoI under a pull model, where information freshness is relevant (only) when the users are interested in the information. For transmission scheduling with AoI minimization, [11] considers a system model with error probability of transmission, and [12] proposes algorithms for multiple link scheduling with presence of interference. We refer to [13] for a comprehensive survey of research on AoI. Below we outline very recent developments that demonstrate a rapidly growing interest in the topic.

One line of research has consisted in sampling, scheduling, and updating policies for various system models that have queuing components with AoI as the performance objective. In [14], the scenario has multiple source-destination pairs with a common queue, along with a scheduler. In [15], the authors have investigated optimal sampling strategies addressing AoI as well as error in estimating the state of a Markov source. The system model in [16] combines stochastic status updates and unreliable links; the authors make an approximation of the Whittle index and derive a solution algorithm thereby. The study in [17] considers an energy-constrained server that is able to harvest energy when no packet from the source requires service. The notion of preemption has been addressed in [18], [19] for a single flow in a multi-hop network with preemptive intermediate nodes, and multiple flows such that different flows preempt each other, respectively. The authors of [20] have proposed the use of dynamic pricing to provide AoI-dependent incentives to sampling and transmission. For network control systems (NCPs), [21] has addressed sampling and scheduling, relating estimation error to AoI, and [22] has focused on approximately optimal scheduling strategy for multi-loop NCPs, where the centralized scheduler takes a decision based on observed AoI.

Another line of research arises from the introduction of AoI to various applications and networking context. By including energy constraints, [23] extends [12] for optimal link activation for AoI minimization in wireless networks with interference. The study in [24] is similar to [12] and [23] in terms of the scheduling aspect and the presence of interference, however the system model considers AoI for all node pairs of the network, and the emphasis is on performance bounds. The work in [25] considers a two-way data exchanging system, where the AoI is constrained by the uplink transmission capability and the downlink energy transfer capability. Application of the AoI concept in camera networks where information from multiple cameras are correlated has been presented in [26]. For remote sensor scenarios, [27], [28] have studied AoI-optimal trajectory planning for unmanned aerial vehicles (UAVs). In [29], the average AoI of data generated by an energy harvesting sensor node is minimized.

In [30] content freshness in opportunistic mobile networks with device-to-device communications is investigated. The trade-off between delay and data staleness has been studied in [31] for distributed content storage systems, and in [32], [33] for vehicular networks. In [34], a learning algorithm is proposed to minimize cache updates in Internet-of-Things (IoT) networks subject to data freshness. In [35], the authors have studied AoI with respect to orthogonal multiple access (OMA) and non-orthogonal multiple access (NOMA), and revealed that NOMA, even though allowing for better spectral efficiency, is not always better than OMA in terms of average AoI. AoI has also appeared as the performance metric in using machine learning as a tool in communication systems. In [36], online adaptive learning has been used for addressing error probability in the context of AoI minimization. In [37], learning is used for optimal data sampling to minimize AoI.

In [38], [39] content caching accounting for both popularity and AoI are studied. In [38], cache miss is minimized, and in [39] the load of backhaul link is minimized via balancing the AoI and cache updates. In [40], using tools from stochastic geometry and queuing theory, random content caching with the AoI metric is analyzed. Additional research for caching with AoI are available in [2], [41], [42], [43]. The general problem setup resembles the system model we consider. However, our work has significant differences. First, it is (implicitly) assumed in these works that the items are of uniform size. We do not have this restriction. Second, these works derive updating policy with respect to expected AoI, whereas our problem falls into the domain of combinatorial optimization and we use methods thereof. Moreover, the problems in these works are approached by either assuming given intervals of updating the cache for each item or a given total number of cache updates. In our work, these entities remain optimization variables throughout the optimization process. Finally, in these works only linear or binary AoI-cost functions can be used, whereas our system model admits any type of functions.

Optimizing caching for AoI applies to scenarios where content items themselves are continuously updated. One such scenario with this type of property is caching of user-generated content (UGC) [44]. Another scenario, used in [2] for AoI-driven caching, is news distribution where the news are updated continuously and the cache is used for holding local copies.

We remark that there is a significant amount of work on time-to-live (TTL) driven caching, in which the cached contents are evicted based on timers [45]. For this type of caching problems,

there is some underlying assumption on the content request model (typically Markov arrival processes). The task is to determine a cache replacement policy, such as first-in-first-out (FIFO) and least-recently-used (LRU) when a timer expires. Whereas FIFO favors consistency and hence more useful for UGC, LRU leads to a higher hit rate but the content gets outdated more easily. We refer to [45] and the references therein for performance analysis of TTL-based caching. As noted earlier, UGC is a scenario that can be addressed with the AoI metric. However, our system setup, similar to that in [2], [41], does not explicitly have individual content requests or request arrival model. In addition, without updates, the value of a cached content gradually degrades rather than a hard timer. Moreover, the task in our problem is not to derive a policy, but a full schedule specifying the updates (i.e., the cache should be updated for which contents in each time slot) with respect to aggregated input data of content popularity over time, obtained via historical observations and/or prediction.

Unlike request-driven caches, publisher-driven caching [46], [47] has the focus on users that generate and publish contents. The key question is then which contents to add and remove after a content is published (rather than when specific requests are made). As publisher-driven caching concerns UGC, the contents may become outdated quickly, and hence it is related to the notion of AoI. In [46], the authors analyze timeline and publisher-driving cache, and derive algorithms for workload shaping, and admission control and pricing. In [47], the authors propose an analytical model and a fairness-based news feed design, using real data from a parliament election. In these works, the modeling approach assumes specific arrival processes of content publishing and the cache updates are mainly based on TTL. In our case we do not assume specific stochastic content update processes and the optimization delivers an update schedule without the use of TTL.

For UGC, one aspect is to deal with long-tailed contents (i.e., contents that are of long-term interest though low popularity). An effective approach, proposed in [44], is to optimize the schedule of distribution of such contents among geographically diverse points of storage, to achieve significant reduction of bandwidth costs as well as delay when a content is requested. The approach utilizes information of social relationships (via social networks), difference of time zones of users interested in a content, and regularities in request patterns. The last aspect is related to cyclic scheduling. However, different from [44], in our study the focus is not on individual user requests or distribution cost, but AoI-driven cache update scheduling.

In [48], the authors have considered a hierarchical caching architecture using RAM and hard drive, and derived a cache updating policy that is asymptotically optimal. Similar to our scenario, the contents vary in size and hence the cache capacity exhibits a knapsack constraint. The main differences to our work are that [48] focuses on the asymptotic performance and the performance metric does not involve AoI.

3 PRELIMINARIES

3.1 System Model

Consider a cache of capacity C. The content items for caching form a set $\mathcal{I} = \{1, \dots, I\}$. Item $i \in \mathcal{I}$ is of size s_i. Time is slotted, and the time horizon consists of a set of time slots $\mathcal{T} = \{1, \dots T\}$. In each time slot, the cache can be updated via a backhaul communication link whose capacity is denoted by L.

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 the cache is updated for an item, 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. The value of holding content item i cached in time slot t is characterized by a utility function f_{ita} that is monotonically decreasing[1] in AoI a. The utility function is item-specific and time-specific to reflect, for example, the difference in the popularity of the content items and the variation of popularity in time. The utility value of adding content item i in time slot t if the content has been absent for the previous τ time slots is denoted by $v_{it\tau}$.

It is worth pointing out the following underlying system assumptions before presenting the optimization problem.

- All downloads of contents (up to the backhaul link capacity) for a time slot can be carried out, i.e., there is zero download delay.
- All contents are retrieved from the same source, i.e., it is not the case that some contents are harder to download than others.
- All contents are ephemeral, i.e., without updating by new downloads they become less useful as time elapses, thus the utility monotonically decreases with AoI.
- Except for the extension to cyclic scheduling (Section 8), there is a finite scheduling horizon.
- The entities defined for the system model and hence the input parameters to our optimization problem are deterministic.
- Optimal scheduling of when and how to update the cache occurs at the aggregated level of content popularity, i.e., optimization is not for individual content requests.

Our AoI-driven cache optimization problem, or ACOP in short, is to determine which content items to store in each time slot and when to do cache updates for them, such that the total utility over the time horizon $1 \dots T$ is maximized, subject to the capacity limits of the cache and the backhaul. Later in Section 8, we will consider the case of optimizing a cyclic schedule of cache updates. We remark that as ACOP is a regular combinatorial optimization problem rather than a stochastic one, its optimum is specified by which contents are downloaded to the cache in every time slot, rather than an update policy with random components. Thus, the optimum is fully determined by the problem input including the initial condition (e.g., initial AoI of the contents).

As mentioned earlier, the utility function of AoI depends on both content and time. One example is $f_{ita} = \frac{\epsilon_i - p_{it}}{T} a + p_{it}$. Here p_{it} represents the popularity of content i in time slot t. By this function, if the AoI $a = 0$, the utility is p_{it}. The utility then linearly decreases in AoI, and if the AoI is as large as the scheduling horizon T, the value becomes ϵ_i that is a small positive value set for content i. Another example is the nonlinear utility function $f_{ita} = \frac{p_{it}}{a+1}$. As for the linear case, the utility is p_{it} if $a = 0$. However, the utility degrades more rapidly than the linear function in the AoI. A third example, originated from [49], is the inverse exponential function $f_{ita} = \frac{1}{e^{a p_{it}}}$. In our numerical study we will use these types of utility functions. However, for the

1. The utility function is not necessarily strictly monotonically decreasing. For a content that gets seldom updated itself, the utility function may be set to decrease slowly in AoI.

algorithms we will use the generic form of the function for the sake of generality.

Remark 1. *Our system model is not necessarily restricted to caching scenarios. For example, consider a system monitoring a number of remote sites, for which the information generated differ in size. The amount of information that can be sent to the monitoring center is constrained by the network bandwidth, and the task is optimal scheduling of updates to address AoI as well as information absence. This setup corresponds to ACOP without the cache capacity constraint.* □

Remark 2. *The above system model has no downloading costs for adding contents to the cache or refreshing contents in the cache. However, the system model can be easily adapted to accommodate a downloading cost that naturally grows linearly in content size. Namely, the utility of zero AoI of a content can be reduced by subtracting the downloading cost of this content, as the AoI is zero if and only if the cache is updated for the content. Thus the aspect of downloading cost is easily addressed by updating the problem input data. The reason of not explicitly including downloading cost is that our focus is on content-centric performance metric, i.e., having as many contents as possible and keeping them as fresh as possible.* □

We end the system model by a mathematical model of ACOP. We use binary variable x_{ita} that equals one if the AoI of item i in time slot t is a, otherwise the variable equals zero. Note that index a is up to $t-1$, because the AoI in slot t can never exceed $t-1$. Also, $x_{it0}=1$ means item i is added to the cache or updated in the cache in time slot t. Binary variable y_{it} is used to indicate if item i is in the cache in time slot t or not. Moreover, binary variable $z_{it\tau}$ equals one if item i is added to the cache in time slot t and the item has been absent from the cache in the previous τ consecutive time slots.

$$\max_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\in\{0,1\}} \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}\left(\sum_{a=0}^{t-1} f_{ita}x_{ita}+\sum_{\tau=1}^{t-1} v_{it\tau}z_{it\tau}\right) \quad \text{(1a)}$$

$$\text{s.t.} \quad x_{ita}=\begin{cases}1 & y_{it}x_{i(t-1)(a-1)}(1-x_{it0})=1\\ 0 & \text{Otherwise}\end{cases}, \quad i\in\mathcal{I}, t\in\mathcal{T}\setminus\{1\}, a=1,\ldots,t-1 \quad \text{(1b)}$$

$$z_{it\tau}=\begin{cases}1 & \prod_{k=t-\tau}^{t-1}(1-y_{ik})x_{it0}=1\\ 0 & \text{Otherwise}\end{cases}, \quad i\in\mathcal{I}, t\in\mathcal{T}\setminus\{1\}, \tau=1,\ldots,t-1 \quad \text{(1c)}$$

$$\sum_{i\in\mathcal{I}} s_i y_{it}\le C, t\in T \quad \text{(1d)}$$

$$\sum_{i\in\mathcal{I}} s_i x_{it0}\le L, t\in T \quad \text{(1e)}$$

The objective function (1a) is to maximize the overall utility. As stated by (1b), $x_{ita}=1$ for $a\ge 1$ if and only if three conditions hold: item i is in the cache ($y_{it}=1$), it has AoI $a-1$ in the time slot $t-1$ ($x_{i(t-1)(a-1)}=1$), and it is not downloaded again in t ($x_{it0}=0$). The definition of $z_{it\tau}$ in (1c) follows a similar rational, i.e., item i is added to the cache in t but not in the previous τ time slots. The next two constraints state cache and backhaul capacity limits, respectively.

The system model and the optimization formulation essentially deal with offline caching scheduling. To address more dynamic scenarios, scheduling that is reactive to changes in the set of contents as well as their popularities becomes relevant. One approach of reactive scheduling is to embed our optimization problem into a rolling horizon scheme. With rolling horizon, the scheduling process optimizes the schedule based on the current knowledge of contents and their popularities, but the optimized schedule is only executed partially, i.e., for some number of time slots. A new optimization problem is then solved accounting for dynamic updates in the input.

3.2 Greedy Solution

For a combinatorial problem, a simple and greedy strategy typically serves as a reference solution. For ACOP, a greedy solution is to maximize the total utility of each time slot by simply considering the utilities of the individual items if they are added to the cache or become updated in the cache. This solution for a generic time slot t is given in Algorithm 1.

Algorithm 1 Greedy solution for a generic time slot

Input: $\mathcal{I}$, $s_i, i\in\mathcal{I}$, current cache $\mathcal{H}$, current AoI $a_i, i\in\mathcal{I}$, current duration of absence $b_i, i\notin\mathcal{H}$
Output: $\mathcal{H}$

$\mathcal{I}'\leftarrow\mathcal{I}$; $L'\leftarrow L$; $r\leftarrow C-\sum_{i\in\mathcal{H}} s_i$; $\mathcal{H}'\leftarrow\emptyset$
while $\mathcal{I}'\neq\emptyset$ and $L'>0$ **do**
  $i^*\leftarrow \text{argmax}_{i\in\mathcal{I}'}(f_{it0}+v_{it\tau_i})$
  $\mathcal{I}'\leftarrow\mathcal{I}'\setminus\{i^*\}$
  if $i^*\in\mathcal{H}$ and $s_{i^*}\le L'$ **then**
    Update cache item $i^*\in\mathcal{H}$
    $\mathcal{H}'\leftarrow\mathcal{H}'\cup i^*$
    $L'\leftarrow L'-s_{i^*}$
  else
    if $r+\sum_{i\in\mathcal{H}\setminus\mathcal{H}'} s_i\ge s_{i^*}$ and $s_{i^*}\le L'$ **then**
      while $r<s_{i^*}$ **do**
        $i'\leftarrow \text{argmin}_{i\in\mathcal{H}\setminus\mathcal{H}'} f_{ita_i}$
        $\mathcal{H}\leftarrow\mathcal{H}\setminus i'$
        $r\leftarrow r+s_{i'}$
      $\mathcal{H}\leftarrow\mathcal{H}\cup\{i^*\}$
      $\mathcal{H}'\leftarrow\mathcal{H}'\cup\{i^*\}$
      $r\leftarrow r-s_{i^*}$
      $L'\leftarrow L'-s_{i^*}$

In the algorithm, $\mathcal{I}'$ is the candidate set of items for caching, r is the residual cache capacity, and $\mathcal{H}'$ is used to keep track on those items that are currently in the cache and selected for update. The item i^*, if added or updated in the cache, leading to the maximum utility, is selected and then removed from $\mathcal{I}'$. If i^* is in the cache, it gets updated and recorded in $\mathcal{H}'$. Otherwise, if the remaining download capacity admits and there will be sufficient residual capacity by removing cached items except those in $\mathcal{H}'$, the algorithm removes items in ascending order of their utility values with respect to the current AoI, followed by adding item i^*. The process ends when the candidate set $\mathcal{I}'$ becomes empty.

4 Problem Complexity Analysis

4.1 NP-Hardness

ACOP is in the domain of combinatorial optimization. As the capacity limits of the cache and the backhaul link imply constraints of knapsack type, it is not surprising that ACOP is NP-hard in general, with a proof based on the binary knapsack problem. This result is formalized below.

Theorem 1. *ACOP is NP-hard.*

Proof: Consider a knapsack problem with N items and knapsack capacity B. Denote the value and weight of item i by v_i and w_i, respectively. We construct an ACOP instance by setting $T = 1$ (i.e., single time slot), $I = N$, $L = C = B$, and $s_i = w_i, i = 1, \ldots, I$. The utility of having items is defined such that $f_{i10} = v_i, i = 1, \ldots, I$, and the utility of adding contents are not needed as there is one single slot. Downloading a content item to the cache amounts to selecting the corresponding item in the knapsack problem. Obviously, the optimal solution maximizing the total utility of the cache leads to the optimum of the knapsack problem, and the result follows. □

4.2 A Tractable Case

Next, we consider a special case of ACOP where the items are of uniform size and the utility of each item i is fully determined by the AoI along time (i.e., $v_{it\tau} = 0$ for any t and τ). We denote this case by ACOP_u. Due to uniform size, both cache and backhaul capacities can be expressed in the number of items. Even though the problem is still combinatorial along the time dimension, we prove it is tractable, i.e., the global optimum can be computed in polynomial time. The key is to transform the problem into a minimum-cost flow problem [50] in a specifically constructed graph.

In the following, we assume that the backhaul capacity constraint, stating that the number of items downloaded in a time slot is upper bounded by L, is non-redundant with respect to the cache capacity C. In particular, the backhaul capacity constraint is always active, and hence we assume $L < C$. Moreover, the download capacity L is always fully used, because making cache updates always leads to better or equal utility (or lower or equal cost for the minimum-cost flow problem).

4.2.1 Graph Construction

In optimization, a network flow problem is defined in a (directed) graph of a set of nodes and a set of arcs (i.e., directed links). A node is called a source if it has some amount of flow to be sent, and a sink if it has to receive some amount of flow. The amount of flow to be sent from or received by a node is typically an integer. Each arc has a cost that is linearly growing in the amount of flow put on the arc. Moreover, an arc has an upper bound and sometimes a lower bound on the flow on this arc. The task is to find a flow pattern, to route the flows from the sources to the sinks, such that the total flow cost is minimum, subject to the upper and lower bounds of the arc flows. For a mathematical representation and solution algorithms, please see [50].

The graph we construct for proving the tractability of ACOP_u is illustrated in Fig. 1. For clarity, we illustrate the construction for two items and the first two time slots. The construction of the remaining time slots follows the same pattern. There is one source node, n_0, and one sink node n_T. The source node generates I flow units to be sent to the sink through the network.

The underlying rationale is as follows. There are three types of nodes. First, for each item i and time slot t, node α_{ita} represents that item i is in the cache in slot t with AoI a. Second, nodes $n_t, t = 1, \ldots, T-1$ act as the collection points for all items except those to be kept in the cache in time slot t. Third, nodes $m_t, t = 0, \ldots, T-1$, are collection points for items to be added or updated in the cache.

Each arc of the graph is associated with tuple (c, l, u), where c is the cost per flow unit, and l and u are the lower and upper bounds of the arc flow, respectively. These values are shown in the figure, however not for the arcs entering the nodes $n_t, t = 1, \ldots, T$ due to lack of space. For these arcs, the tuple is $(0, 0, 1)$. Moreover, we do not show explicitly the tuples for some arcs of time slot two for the sake of clarity; the tuple values of any such arc are the same as for the corresponding arc in the previous time slot. We remark that in the graph construction, the costs, representing the negative values of utility, are all associated with arcs. The network nodes do not have any cost value.

Let us consider the first time slot with any integer flow solution. The I flow units leaving source n_0 have to either enter m_0 or n_1. Note that the maximum number of units entering m_0 has upper bound L, allowing at most L items to be added or updated. Node m_0 has an arc of capacity one to node α_{i10} of item i, $i = 1, \ldots, I$. Having a flow of one unit on the arc corresponds to putting item i in the cache, generating utility f_{i10} that is equivalent to an amount of $-f_{i10}$ in cost minimization. There will be L such items as it is optimal to fully use the backhaul capacity. The flow units from n_0 to n_1 correspond to the items not cached in time slot one. Note the flow lower bound of arc (n_0, n_1) is $I - C$, meaning that at least $I - C$ flow units must enter node n_1, i.e., at least $I - C$ items are outside the cache.

For a generic time slot t and item i, if one unit of flow enters node α_{ita}, then the flow has two choices by graph construction. Either it goes to node $\alpha_{i(t+1)(a+1)}$, representing that item i remains in the cache, with AoI $a + 1$ for the next time slot and the corresponding utility, or it has to be sent to the collection node n_{t+1}. The latter case represents that the item is removed from the cache, and, from n_{t+1}, the unit flow either enters $\alpha_{i(t+1)0}$ via node m_{t+1} (i.e., item i is downloaded again to the cache with AoI zero), or it enters collection node n_{t+2} meaning that item i stays outside the cache.

Remark 3. *In Fig. 1, some of the nodes (and hence also their adjacent arcs) are redundant. For example, no flow will enter nodes α_{122} and α_{123}, because the AoI will never attain two or three in time slot two. These nodes are however kept in the figure to better reflect the general structure of graph construction.* □

4.2.2 Tractability Analysis

Lemma 2. *The optimal solution to ACOP_u corresponds to an integer flow solution for the constructed graph with equivalent objective function value.*

Proof: See Appendix A. □

Lemma 3. *The optimal integer flow solution in the constructed graph corresponds to a solution of ACOP_u with equivalent objective function value.*

Proof: See Appendix B. □

Theorem 4. *ACOP_u is tractable with polynomial-time complexity.*

Proof: The maximum possible AoI of any item in the cache is bounded by the number of time slots T. Hence the size of the constructed graph is polynomial in I and T. Moreover, the graph is clearly acyclic. For minimum-cost flow problems with integer input, there is an optimal solution in which the arc flows are integers, and there are (strong) polynomial algorithms for the problem including that with negative costs in an acyclic

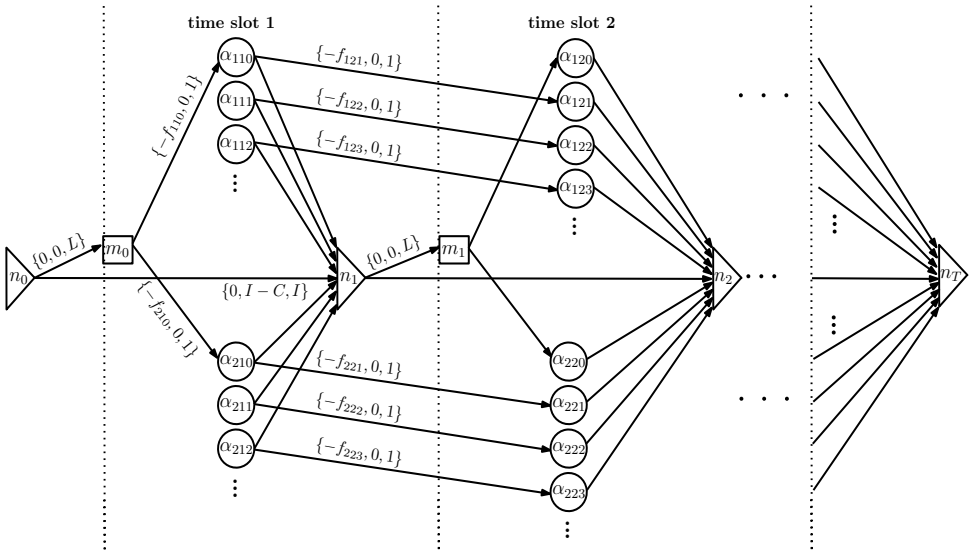

Fig. 1. An illustration of the network flow problem corresponding to ACOP_u.

graph [50]. These facts, together with Lemmas 2-3, establish the theorem. □

Remark 4. *For minimum-cost flow problems, the corresponding integer linear optimization formulation has a constraint matrix that is always totally unimodular [50, Chapter 11]. Consequently, solving the linear relaxation, which is tractable in polynomial time, will yield the integer optimum. Note that for minimum-cost flow, there are specialized polynomial-time algorithms [50, Chapter 10] that run faster than solving it as a generic linear program. It is also interesting to note that the notion of total unimodularity is generally useful for proving tractability. In [51], for example, the authors used this notion to prove the tractability of a special case of another caching optimization problem that is NP-hard in general.*

Remark 5. *For combinatorial optimization problems, tractable problem sub-classes are often identified by proving that a greedy solution leads to optimum. For $ACOP_u$, however, the greedy solution in Section 3.1 remains sub-optimal. This is due to item-specific utility function and backhaul capacity. Consider a simple example of two items. Both are of size one. The capacity values are $C = 2$ and $L = 1$. For item one, the utility function $f_{1t0} = 2k$ (with $k > 1$) for any t and $f_{1ta} = 0$ for $a \geq 1$ and any t. For item two, $f_{2ta} = k$ for $a \leq t'$, where t' is a positive integer, and $f_{2ta} = 0$ for $a \geq t' + 1$. The greedy solution would cache and update item one in all time slots. Thus for $T = t'$, the total utility is $2kt'$. The optimal solution is to cache item two in time slot one. Then, item one is added and kept updated in the next $t' - 1$ time slots. This gives a total utility of $kt' + 2k(t' - 1) = 3kt' - 2k$. As a result, the greedy solution becomes highly sub-optimal for large t'. Thus the notion of network flows is necessary for arriving at the conclusion of the tractability of $ACOP_u$.* □

5 Integer Linear Formulation

We derive an integer linear programming (ILP) formulation for ACOP. This leads to a solution approach of using an off-the-shelf optimization solver (e.g., [52]). Even though the approach does not scale, it can be used to obtain the optimum of small-size problem instances, for the purpose of performance evaluation of other algorithms. The ILP formulation is given below. In the formulation, we reuse the binary variables defined for (1).

$$\max_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\in\{0,1\}} \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}\left(\sum_{a=0}^{t-1} f_{ita}x_{ita} + \sum_{\tau=1}^{t-1} v_{it\tau}z_{it\tau}\right) \tag{2a}$$

$$\text{s.t.} \sum_{a=0}^{t-1} x_{ita} = y_{it}, i \in \mathcal{I},\ t \in \mathcal{T} \tag{2b}$$

$$x_{ita} \geq y_{it} + x_{i(t-1)(a-1)} - x_{it0} - 1, \quad i \in \mathcal{I}, t \in \mathcal{T}\setminus\{1\}, a = 1,\ldots,t-1 \tag{2c}$$

$$x_{i(t-1)(a-1)} \geq x_{ita}, \quad i \in \mathcal{I}, t \in \mathcal{T}\setminus\{1\}, a = 1,\ldots,t-1 \tag{2d}$$

$$\sum_{\tau=0}^{t-1} z_{it\tau} = x_{it0}, i \in \mathcal{I}, t \in \mathcal{T} \tag{2e}$$

$$z_{it\tau} \leq 1 - y_{ik}, \quad i \in \mathcal{I}, t \in \mathcal{T}, \tau = 1,\ldots,t-1, k = t-\tau,\ldots,t-1 \tag{2f}$$

(1d), (1e).

The objective function (2a) is the same as in (1). By (2b), if item i is cached in a time slot t, i.e., $y_{it} = 1$, then exactly one of the binary variables representing the possible AoI values is one, otherwise these binary variables have to be zeros. Inequality (2c) states that if item i is cached with AoI $a-1$ in time slot $t-1$, then for time slot t, the AoI becomes a (represented by $x_{ita} = 1$), unless it gets refreshed (represented by $x_{it0} = 1$) or the item is no longer in the cache (represented by $y_{it} = 0$). By (2d), if item i is cached with AoI $a \geq 1$, it must have AoI $a-1$ in the previous time slot. Constraints (2e) and (2f) together have the effect that $z_{it\tau}$ can be one only if the item is added or refreshed in time slot t and it has been absent in the previous τ slots where $\tau \in \{0,\ldots,t-1\}$.

6 Repeated Column Generation Algorithm

For efficient solution of ACOP, we propose an algorithm based on repeated column generation. Column generation is an efficient

method for solving large-scale linear programs with the following two structural properties [53]. First, there are exponentially many columns (i.e., variables), hence including all is not practically feasible and the method deals with only a small subset of them. Second, identifying new columns that improve the objective function value can be performed by solving an auxiliary problem, named the pricing problem or sometimes the subproblem. This enables successive addition of new and promising columns until optimality is reached.

6.1 Problem Reformulation

Applying column generation to ACOP is based on a reformulation. In the reformulation, a column of any item is a vector representing the caching and updating decisions of the item over all time slots. We denote by $\mathcal{L}_i$ the index set of all possible such vectors for item i. For each column $\ell \in \mathcal{L}_i$, there is a tuple $(b_{it\ell}, u_{it\ell})$ for every time slot t. Both elements are binary. Specifically, a column is represented by $[(b_{i1\ell}, u_{i1\ell}), (b_{i2\ell}, u_{i2\ell}), \ldots, (b_{iT\ell}, u_{iT\ell})]$ in which $b_{it\ell}$ and $u_{it\ell}$ are one if and only if item i is cached and updated in the cache, respectively, in time slot t. Note that there are exactly three possible outcomes of the tuple values: $(0,0)$, $(1,0)$, and $(1,1)$. As a result, the cardinality of set $\mathcal{L}_i$ is bounded by 3^T.

Because a column ℓ fully specifies the caching and updating solution, the associated AoI values as well as the time of being absent from the cache over time are known for any given column. Hence, the overall utility of column ℓ for item i, denoted by $f_{i\ell}$, can be computed.

The problem reformulation uses a binary variable $\chi_{i\ell}$ for each item $i \in \mathcal{I}$ and column $\ell \in \mathcal{L}$. This variable is one if column ℓ is selected for item i, and zero otherwise. The reformulation is given below.

$$\max_{\boldsymbol{\chi}} \sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}} f_{i\ell}\chi_{i\ell} \tag{3a}$$

$$\text{s.t.} \sum_{\ell\in\mathcal{L}} \chi_{i\ell} = 1,\ i \in \mathcal{I} \tag{3b}$$

$$\sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}} b_{it\ell} s_i \chi_{i\ell} \le C,\ t \in \mathcal{T} \tag{3c}$$

$$\sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}} u_{it\ell} s_i \chi_{i\ell} \le L,\ t \in \mathcal{T} \tag{3d}$$

$$\chi_{i\ell} \in \{0,1\},\ i \in \mathcal{I}, \ell \in \mathcal{L}_i \tag{3e}$$

In (3), (3b) states that exactly one column (i.e., caching and updating solution) has to be selected for every item. The next two constraints formulate the cache and backhaul capacity, respectively. Note that both (3) and (2) are valid optimization formulations of ACOP. However they differ in structure.

6.2 Column Generation

The structure of (3) can be exploited by using column generation [53]. The underlying principle originates from the well-known simplex method for linear programming (LP). In every iteration of this method, the variables are partitioned into basic variables with positive values (except for the case of degeneracy), and non-basic variables that are zeros. The method updates this partition of variables from one iteration to another. The update is based on the so called reduced cost that in turn is dependent on the values of dual variables associated with the constraints.

Some linear programs, such as (3), are too large in the number of variables, so not all of them can be considered explicitly. However, at optimum, most of the variables will be non-basic with value zero. Note that a variable with value zero has the same effect as not having the variable present at all. Column generation utilizes this fact by excluding a large portion of the variables (i.e., treating them as non-basic ones). In each iteration, if any of these shall become a basic variable, it becomes included in the problem. The difference to the regular simplex method is that, since the reduced cost cannot be directly computed for the excluded variables, an auxiliary optimization problem needs to be solved in order to find one such variable that needs to be included for improvement.

Consider the linear programming counterpart of (3), where (3e) is replaced by the relaxation $0 \le \chi_{i\ell} \le 1, i \in \mathcal{I}, \ell \in \mathcal{L}$. Moreover, for each item i, a small subset $\mathcal{L}'_i \subset \mathcal{L}_i$ is used and successively augmented to approach optimality. Initially, $\mathcal{L}'_i$ can be as small as a singleton, containing only the column representing not caching the item at all. One iteration of column generation solves the following LP, referred to as the restricted master problem (RMP) because $\mathcal{L}'_i$ is a subset of $\mathcal{L}_i$.

$$\max_{\boldsymbol{\chi}} \sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}'_i} f_{i\ell}\chi_{i\ell} \tag{4a}$$

$$\text{s.t.} \sum_{\ell\in\mathcal{L}'_i} \chi_{i\ell} = 1,\ i \in \mathcal{I} \tag{4b}$$

$$\sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}'_i} b_{it\ell} s_i \chi_{i\ell} \le C,\ t \in \mathcal{T} \tag{4c}$$

$$\sum_{i\in\mathcal{I}} \sum_{\ell\in\mathcal{L}'_i} u_{it\ell} s_i \chi_{i\ell} \le L,\ t \in \mathcal{T} \tag{4d}$$

$$0 \le \chi_{i\ell} \le 1,\ i \in \mathcal{I}, \ell \in \mathcal{L}'_i \tag{4e}$$

At the optimum of (4), denote by λ_i^* $(i \in \mathcal{I})$, π_t^* $(t \in \mathcal{T})$, and μ_t^* $(t \in \mathcal{T})$ the corresponding optimal dual variable values of (4b), (4c), and (4d), respectively. For any item i, the LP reduced cost of column ℓ equals $f_{i\ell} - \lambda_i^* - \sum_{t\in\mathcal{T}} b_{it\ell} s_i \pi_t^* - \sum_{t\in\mathcal{T}} u_{it\ell} s_i \mu_t^*$. We are interested in knowing if there exists any column with a positive reduced cost (among those in $\mathcal{L}_i \setminus \mathcal{L}'_i$ for item i). If so, adding the column to (4) shall improve the objective function value. This can be accomplished by finding the column of maximum reduced cost for each item. Clearly, the resulting optimization task decomposes by item.

Recall that any column ℓ is characterized by a vector of tuples of binary values $[(b_{i1\ell}, u_{i1\ell}), (b_{i2\ell}, u_{i2\ell}), \ldots, (b_{iT\ell}, u_{iT\ell})]$. Finding the column of maximum reduced cost amounts to setting binary values of the tuples, such that the corresponding reduced cost is maximized. These values represent the update and caching decisions of one item, and hence we can use the variable definitions in Section 5. The subproblem for a generic item i is formulated below in (5).

There are clear similarities between (5) and (2), as both concern optimizing caching and updating decisions. However, they differ in several significant aspects. First, (5) deals with a single item. Second, the cache and backhaul capacities are not present in (5) because these are addressed in (3). Finally, the objective function (5a) contains the dual variables as the purpose is to maximize the reduced cost. Note that dual variable λ_i^* is a constant for item i and hence it is not explicitly included in (5a).

Subproblem (5) is an integer linear problem. As illustrated in Fig. 2, it can be transformed into a shortest path problem that is polynomial-time solvable. Note that the dual variables appear as

part of the arc costs in the graph. In the figure, notation κ_t^* is the sum of dual variables π_t^* and μ_t^*.

$$(\text{SP}_i) \quad q_i^* = \max_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \sum_{t\in\mathcal{T}} \left(\sum_{a=0}^{t-1} f_{ita} x_{ita} + \sum_{\tau=1}^{t-1} v_{it\tau} z_{it\tau} \right) - s_i \sum_{t\in\mathcal{T}} (\pi_t^* y_{it} + \mu_t^* x_{it0}) \tag{5a}$$

$$\text{s.t.} \sum_{a=0}^{t-1} x_{ita} = y_{it},\ t \in \mathcal{T} \tag{5b}$$

$$x_{ita} \geq y_{it} + x_{i(t-1)(a-1)} - x_{it0} - 1, \quad i \in \mathcal{I}, t \in \mathcal{T} \setminus \{1\}, a = 1, \ldots, t-1 \tag{5c}$$

$$x_{i(t-1)(a-1)} \geq x_{ita}, \quad t \in \mathcal{T} \setminus \{1\}, a = 1, \ldots, t-1 \tag{5d}$$

$$\sum_{\tau=0}^{t-1} z_{it\tau} = x_{it0}, t \in \mathcal{T} \tag{5e}$$

$$z_{it\tau} \leq 1 - y_{ik}, \quad t \in \mathcal{T}, \tau = 1, \ldots, t-1, k = t-\tau, \ldots, t-1 \tag{5f}$$

$$x_{ita} \in \{0,1\},\ t \in \mathcal{T}, a = 0, \ldots, t-1 \tag{5g}$$

$$y_{it} \in \{0,1\},\ t \in \mathcal{T} \tag{5h}$$

$$z_{it\tau} \in \{0,1\},\ t \in \mathcal{T}, \tau = 0, \ldots, t-1 \tag{5i}$$

Theorem 5. *The shortest path from node n_0 to n_T gives the optimal solution of* (5)*.*

Proof: To establish the theorem, we show that there is a one-to-one mapping between a path from node n_0 to node n_T in Fig. 2 and a sequence of caching decisions of a generic item i. In the figure, the α-nodes of a time slot represent caching the item with different AoI values. For example, a path entering node α_{i31} corresponds to caching the content item in time slot 3 with AoI 1. The β-nodes are introduced to represent that the item is outside the cache and to keep track on how long time it has been absent from the cache. For example, a path entering β_{i32} means the item is not cached in time slot 3 with a duration of two time slots (i.e., time slots 2 and 3).

To see the one-to-one mapping, we first observe that the graph in Fig. 2 is acyclic, and by its construction, a path from n_0 to n_T will have to pass exactly one arc between the two sets of nodes connecting two consecutive time slots t and $t+1$. There are four types of such arcs.

1) An arc from an α-node of time slot t to some α-node of time slot $t+1$ corresponds to having the content in both time slots. For a genetic α-node α_{ita}, there are two possible arcs by graph construction, connected to $\alpha_{i(t+1)0}$ and $\alpha_{i(t+1)(a+1)}$, respectively The former represents to update the cache by refreshing the content, and the cost is $-f_{i(t+1)0} + s_i\kappa_{t+1}^* = -f_{i(t+1)0} + s_i(\pi_{t+1}^* + \mu_{t+1}^*)$. This value is the negation of the objective coefficients of having $x_{i(t+1)0} = 1$ in the subproblem. The latter arc means to continue caching the content without update, and the arc cost equals that in the objective function (5a) for $x_{i(t+1)(a+1)} = 1$.
2) An arc from a β-node to an α-node corresponds to adding the item (that has been absent from the cache for at least one time slot) to the cache. By graph construction, for node $\beta_{it\tau}$ there is only one such arc $(\beta_{it\tau}, \alpha_{i(t+1)0})$, i.e., the AoI of the item in the next time slot is zero. The cost value is $-f_{i(t+1)0} - v_{i(t+1)\tau} + s_i\kappa_{t+1}^* = -f_{i(t+1)0} - v_{i(t+1)\tau} + s_i(\pi_{t+1}^* + \mu_{t+1}^*)$. This is the negation of the corresponding utility in the subproblem and the only difference to the above case is the utility term $v_{i(t+1)\tau}$ of adding item to the cache after being absent for τ time slots.
3) An arc from an α-node to a β-node corresponds to removing a currently cached content item. For node α_{ita}, there is only one such arc $(\alpha_{ita}, \beta_{i(t+1)1})$, i.e., in the next time slot the item is absent from the cache with a duration of one time slot. The arc cost is zero because there is no utility of having the item outside the cache in the next time slot.
4) Finally, an arc from a β-node to another β-node corresponds to keeping the content outside the cache. Note that for any β-node, say $\beta_{it\tau}$, there is only such arc, leading to $\beta_{i(t+1)(\tau+1)}$, that is, if the content item has been outside the cache in time slot t for τ time slots and it remains outside the cache in the next time slot $t+1$, the duration of absence is $\tau+1$. The arc cost is again zero because this action gives zero utility in the subproblem.

In addition to the above, any path has to use one of the arcs connecting the nodes defined for time slot T to node n_T; these arcs all have cost λ_i as the reduced cost has a constant utility term of $-\lambda_i$.

By the analysis, there is a one-to-one mapping between the paths in the graph and the subproblem solutions, and the shortest path problem (or the longest path problem if we do not take the negation of the values) from n_0 to n_T gives the optimal subproblem solution. The theorem then follows from that the graph is acyclic and its size is polynomial in instance size. □

Remark 6. *Suppose a partial decision is taken such that an item shall not be cached in some particular time slot. This amounts to simply deleting all arcs entering the α-nodes for that time slot. Conversely, requiring that the item is cached in a time slot corresponds to removing all arcs entering the β-nodes of the time slot. Moreover, accounting for adding or refreshing the content in a particular time slot corresponds to removing the arcs except those entering the α-node of AoI zero. Thus solving the subproblem can easily accommodate partial decisions. This observation is useful for attaining an integer solution (Section 6.3).* □

Column generation for ACOP is summarized in Algorithm 2. Applying Algorithm 2, the optimal objective function value is the LP optimum of (3) and is therefore an upper bound of the global optimum of ACOP. This upper bound is very useful to gauge the performance of any suboptimal solution, because the deviation from the global optimum is bounded by that to the upper bound.

6.3 Attaining Solution Integrality

The solution by Algorithm 2 may be fractional. A naive rounding algorithm would pick some fractional χ-variable of some item i and round it either up to one or down to zero, depending on the value. This way of rounding is rather aggressive, however, because the caching and updating decisions over all time slots become fixed if a variable is set to be one, and there would be no opportunity to make any further decision for item i.

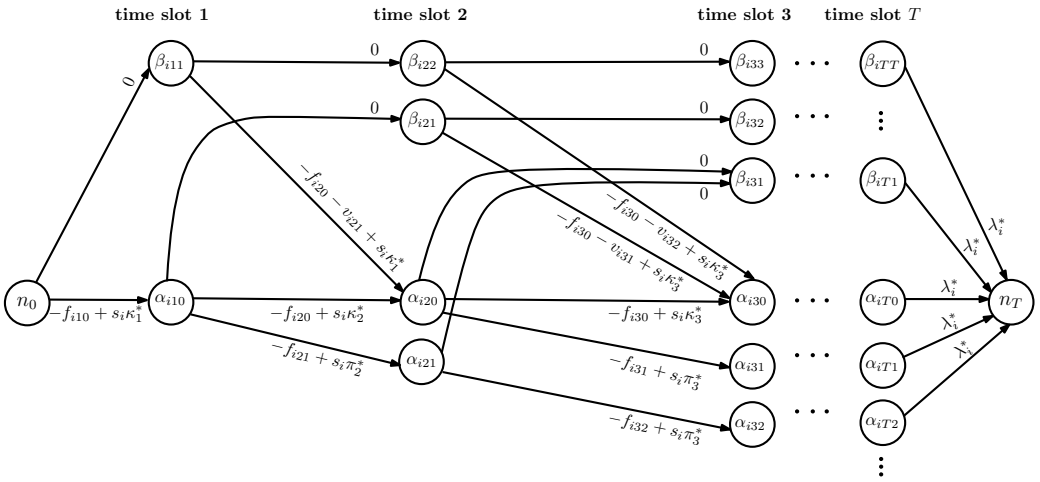

Fig. 2. The graph for which finding the shortest path gives the optimum of (5).

Algorithm 2 Column generation

Input: $\mathcal{I}$, $\mathcal{T}$, $s_i, i \in \mathcal{I}$, C, L, f_{ita}, $i \in \mathcal{I}, t \in \mathcal{T}, a \in \mathcal{T}$, $v_{it\tau}, i \in \mathcal{I}, t \in \mathcal{T}, \tau \in \mathcal{T}$
Output: $\boldsymbol{\chi}^*$

```
Initialize L_i, i ∈ I
Stop ← False
while Stop = False do
    Solve (4) to obtain optimum χ* and dual optimum (λ*, π*, μ*)
    Stop ← True
    for i = 1, ..., I do
        Solve (5)
        if q_i^* − λ_i^* > 0 then
            Stop ← False
            Add the column corresponding to q_i^* to L'_i
```

We consider performing rounding more gracefully. The idea is to make a rounding decision for one item and one time slot at a time. Given the optimum LP solution $\boldsymbol{\chi}^*$ via column generation, we define (cf. variables used in formulation (2) in Section 5) $w_{it} = \sum_{\ell \in \mathcal{L}_i'} b_{it\ell}\chi_{i\ell}^*$ and $r_{it} = \sum_{\ell \in \mathcal{L}_i'} u_{it\ell}\chi_{i\ell}^*$. These entities can be interpreted as the likelihood of caching and updating item i in time slot t, respectively. Note that for item i, there may be multiple (and factional) χ^*-variables contributing to the values of w_{it} and r_{it}. The following theorem states that, to attain an integer solution for (4), it is sufficient that w_{it} and r_{it}, $i \in \mathcal{I}, t \in \mathcal{T}$, become integer.

Theorem 6. *$\boldsymbol{\chi}^*$ is integer if and only if $\boldsymbol{r}$ and $\boldsymbol{w}$ are integer.*

Proof: The necessity is obvious by the definition of $\boldsymbol{r}$ and $\boldsymbol{w}$. For sufficiency, assume $\boldsymbol{r}$ and $\boldsymbol{w}$ are integer. Suppose $\boldsymbol{\chi}^*$ is fractional, and assume more specifically $0 < \chi_{i\ell}^* < 1$. By (4b), there must exist at least another χ-variable for item i that has fractional value. Let $\hat{\mathcal{L}}_i$ denote the column index set of the fractional χ-variables of item i. Because there are no identical columns in $\mathcal{L}_i'$, there must exist some time slot t, for which $\hat{\mathcal{L}}_i$ can be partitioned into two sets, $\hat{\mathcal{L}}_i^0$ and $\hat{\mathcal{L}}_i^1$, such that $b_{t\ell} = 0, \ell \in \hat{\mathcal{L}}_i^0$ and $b_{t\ell} = 1, \ell \in \hat{\mathcal{L}}_i^1$, and none of the two sets is empty. Thus $w_{it} = \sum_{\ell \in \mathcal{L}_i'} b_{t\ell}\chi_{i\ell}^* = \sum_{\ell \in \hat{\mathcal{L}}_i^1} b_{t\ell}\chi_{i\ell}^* < 1$, contradicting that w_{it} is integer. The same argument applies to $\boldsymbol{r}$, and the theorem follows. □

By the above result, rounding can be performed on $\boldsymbol{r}$ and $\boldsymbol{w}$, instead of $\boldsymbol{\chi}^*$. This amounts to considering disjunctions, i.e., a partition of the χ-variables into two subsets rather than considering a single χ-variable (cf. the proof of Theorem 6), and resembles the use of disjunctions in branch-and-bound in solving integer programs [54]. Hence we use the term disjunction-based rounding (DR) to refer to the rounding concept.

After performing DR, column generation is applied again. This is because additional columns may be needed to reach the LP optimum given the constraint imposed by rounding. For example, setting $w_{it} = 0$ means to exclude columns $\{\ell \in \mathcal{L}_i : b_{it\ell} = 1\}$ and since the current $\mathcal{L}_i'$ is a small subset of $\mathcal{L}_i$, some new column $\ell \notin \mathcal{L}_i'$ with $b_{it\ell} = 0$ may improve the objective function. This yields the repeated column generation algorithm, or RCGA in short.

Note that, after a DR operation, both (4) and the subproblem of column generation have to comply with the rounding decision. Namely, if $w_{it} = 0$ by DR, then $\{\ell \in \mathcal{L}_i' : b_{it\ell} = 1\}$ are removed from (4). If $w_{it} = 1$, then $\{\ell \in \mathcal{L}_i' : b_{it\ell} = 0\}$ are removed from (4). Similar updates apply for DR on $\boldsymbol{r}$. The subproblem graph is updated accordingly as well (see Remark 6).

6.4 Algorithm Flow

The algorithm flow has the following key components.

- In each major iteration, column generation is used to solve the linear programming version of (3) in which some of the caching and updating decisions (represented by $\boldsymbol{w}$ and $\boldsymbol{r}$) are made. To this end, column generation iterates between the RMP (4) and the subproblem (5) until optimality is reached.
- After a major iteration, based on the values of $\boldsymbol{w}$ and $\boldsymbol{r}$, the items and time slots that have integer values will be fixed. Next, one content item i and time slot t with fractional value is selected, for which the value of either w_{it} or r_{it} is set to be one or zero.
- Next, the algorithm removes the columns in RMP that are not compatible with the rounding decision, and makes the corresponding restriction to the subproblem. The process then repeats until an integer solution is obtained.

The details of DR is presented in Algorithm 3. In the algorithm, symbol $\leftarrow$ and symbol $\leftleftarrows$ both represent the assignment of

a value to an entity. The difference is that with the latter symbol $\leftleftarrows$, the assignment value will be kept in all subsequent algorithm steps, with the purpose of making the algorithm easier to follow. Algorithm 3 uses two subroutines presented in Algorithms 4 and 5.

Algorithm 4 calculates $\boldsymbol{r}$ and $\boldsymbol{w}$ (Lines 1-2), and makes fixing decisions of $\boldsymbol{x}$ and $\boldsymbol{y}$ in the subproblem (Lines 3 and 5), and discards non-complying columns from the RMP (Lines 4 and 6). Lines 7-10 calculate the remaining spare backhaul and cache capacities, denoted by L' and C', respectively. Finally, Lines 11 and 12 compute the number of fractional elements of $\boldsymbol{r}$ and $\boldsymbol{w}$, denoted by ξ and η, respectively. The values of ξ and η are returned to Algorithm 3.

If both ξ and η equal zero, the current solution is integer by Theorem 6. Otherwise, one DR is applied for $\boldsymbol{r}$ if $\xi > 0$ (Lines 3-13). If $\boldsymbol{r}$ is integer but $\eta > 0$, DR is applied to a fractional element of $\boldsymbol{w}$ (Lines 15-27).

More specifically, Line 3 finds the fractional element of $\boldsymbol{r}$ being closest to zero, and the corresponding item and time slot. These entities are denoted by $\underline{r}$, $\underline{i}$, and $\underline{t}$, respectively. The corresponding entities in examining the fractional element of $\boldsymbol{r}$ that is closest to one (Line 4) are denoted by $\bar{r}$, $\bar{i}$, and $\bar{t}$, respectively. If $\underline{r} < \bar{r}$, DR fixes $x_{\underline{i}\underline{t}0}$ to be zero by Line 6. Furthermore, the non-complying columns are discarded from $\mathcal{L}_{\underline{i}}$ by Line 7. If $\underline{r} \geq \bar{r}$, the algorithm checks whether or not there is enough remaining backhaul capacity to download item $\bar{i}$. If the answer is positive, $x_{\bar{i}\bar{t}0}$ is fixed to be one in $\text{SP}_{\bar{i}}$ by Line 9, and the non-complying columns are deleted from $\mathcal{L}_{\bar{i}}$ by Line 10. If the remaining capacity does not permit to download $\bar{i}$, $x_{\bar{i}\bar{t}0}$ is fixed to be zero by Line 12 and the non-complying columns will be discarded from $\mathcal{L}'_{\bar{i}}$ by Line 13. DR based on $\boldsymbol{w}$ is similar to that for $\boldsymbol{r}$, and the details are presented in Lines 15-27.

As the next step, the algorithm calls Algorithm 5 to update the remaining spare backhaul and cache capacity limits (Lines 2-5). Thereafter, Algorithm 5 examines if any content item has a larger size than what can be admitted by the capacity limit in any time slot. For any such item and time slot, the corresponding variable is fixed to zero, and the non-complying columns are discarded (Lines 6-13).

6.5 Algorithm Summary

The framework of RCGA is shown in Algorithm 6 that iterates between CGA and DR. As there are I items and T time slots, and at least one element of $\boldsymbol{x}$ and $\boldsymbol{y}$ becomes fixed in value in each iteration, Algorithm 6 terminates in at most $I \times T$ iterations.

We remark that Algorithm 6 is deterministic. That is, the algorithm will return a solution specifying fully the cached contents and updates in every time slot.

7 Performance Results

In this section, we present performance evaluation results of RCGA and the greedy algorithm (GA). We consider ACOP instances of both small and large sizes. For the former, we compare the utility achieved by RCGA and GA to the global optimum obtained from solving ILP (2). For ILP, we use Gurobi optimizer [52]. By using global optimum as the reference, we obtain accurate evaluation in terms of the (relative) deviation from the optimum, referred to as the optimality gap. For large-size ACOP instances, it is computationally difficult to obtain global optimum. Instead, we use the upper bound derived from the first iteration of RCGA as the reference value. This is a valid comparison because

Algorithm 3 DR Algorithm

1: Run Algorithm 4 and obtain ξ and η
2: **if** $\xi > 0$ **then**
3: $(\underline{r}, \underline{t}, \underline{i}) \leftarrow \min_{i\in\mathcal{I}, t\in\mathcal{T}}\{r_{it} | r_{it} > 0 \text{ and } r_{it} < 1\}$
4: $(\bar{r}, \bar{t}, \bar{i}) \leftarrow \min_{i\in\mathcal{I}, t\in\mathcal{T}}\{1 - r_{it} | r_{it} > 0 \text{ and } r_{it} < 1\}$
5: **if** $(\underline{r} < \bar{r})$ **then**
6: $x_{\underline{i}\underline{t}0} \leftleftarrows 0$ in $\text{SP}_{\underline{i}}$
7: $\chi_{\underline{i}\ell} \leftleftarrows 0$ if $u_{\underline{i}\underline{t}\ell} = 1$, $\ell \in \mathcal{L}'_{\underline{i}}$
8: **else if** $(s_{\bar{i}} \leq L')$ **then**
9: $x_{\bar{i}\bar{t}0} \leftleftarrows 1$ in $\text{SP}_{\bar{i}}$
10: $\chi_{\bar{i}\ell} \leftleftarrows 0$ if $u_{\bar{i}\bar{t}\ell} = 0$, $\ell \in \mathcal{L}'_{\bar{i}}$
11: **else**
12: $x_{\bar{i}\bar{t}0} \leftleftarrows 0$ in $\text{SP}_{\bar{i}}$
13: $\chi_{\bar{i}\ell} \leftleftarrows 0$ if $u_{\bar{i}\bar{t}\ell} = 1$, $\ell \in \mathcal{L}'_{\bar{i}}$
14: **else if** $\eta > 0$ **then**
15: $y_{it} \leftleftarrows 1$ in SP_i if $w_{it} = 1$, $i \in \mathcal{I}, t \in \mathcal{T}$
16: $\chi_{i\ell} \leftleftarrows 0$ in RMP if $b_{it\ell} = 0$, $i \in \mathcal{I}, t \in \mathcal{T}, \ell \in \mathcal{L}'_i$
17: $(\underline{w}, \underline{t}, \underline{i}) \leftarrow \min_{i\in\mathcal{I}, t\in\mathcal{T}}\{w_{it} | w_{it} > 0 \text{ and } w_{it} < 1\}$
18: $(\bar{w}, \bar{i}, \bar{t}) \leftarrow \min_{i\in\mathcal{I}, t\in\mathcal{T}}\{1 - w_{it} | w_{it} > 0 \text{ and } w_{it} < 1\}$
19: **if** $(\underline{w} < \bar{w})$ **then**
20: $y_{\underline{i}\underline{t}} \leftleftarrows 0$ in $\text{SP}_{\underline{i}}$
21: $\chi_{\underline{i}\ell} \leftleftarrows 0$ if $b_{\underline{i}\underline{t}\ell} = 1$, $\ell \in \mathcal{L}'_{\underline{i}}$
22: **else if** $(s_{\bar{i}} \leq C')$ **then**
23: $y_{\bar{i}\bar{t}} \leftleftarrows 1$ in $\text{SP}_{\bar{i}}$
24: $\chi_{\bar{i}\ell} \leftleftarrows 0$ if $b_{\bar{i}\bar{t}\ell} = 0$, $\ell \in \mathcal{L}'_{\bar{i}}$
25: **else**
26: $y_{\bar{i}\bar{t}} \leftleftarrows 0$ in $\text{SP}_{\bar{i}}$
27: $y_{\bar{i}\ell} \leftleftarrows 0$ if $b_{\bar{i}\bar{t}\ell} = 1$, $\ell \in \mathcal{L}'_{\bar{i}}$
28: Run Algorithm 5

Algorithm 4 Retriving fractional values

Compute $\boldsymbol{r} = \{r_{it}, i \in \mathcal{I}, t \in \mathcal{T}\}$, where $r_{it} = \sum_{\ell\in\mathcal{L}_i} u_{it\ell}\chi^*_{il}$
Compute $\boldsymbol{w} = \{w_{it}, i \in \mathcal{I}, t \in \mathcal{T}\}$, where $w_{it} = \sum_{\ell\in\mathcal{L}_i} b_{it\ell}\chi^*_{i\ell}$
$x_{it0} \leftleftarrows 1$ and $y_{it} \leftleftarrows 1$ in SP_i if $r_{it} = 1$, $i \in \mathcal{I}, t \in \mathcal{T}$
$\chi_{i\ell} \leftleftarrows 0$ in RMP if $u_{it\ell} = 0$, $i \in \mathcal{I}, t \in \mathcal{T}, \ell \in \mathcal{L}'_i$
$x_{it0} \leftleftarrows 0$ and $y_{it} \leftleftarrows 0$ in SP_i if $w_{it} = 0$, $i \in \mathcal{I}, t \in \mathcal{T}$
$\chi_{i\ell} \leftleftarrows 0$ in RMP if $b_{it\ell} = 1$, $i \in \mathcal{I}, t \in \mathcal{T}, \ell \in \mathcal{L}'_i$
$\mathcal{I}' \leftarrow \{i \in \mathcal{I} | x_{it0} \text{ is fixed to one}\}$
$\mathcal{I}'' \leftarrow \{i \in \mathcal{I} | y_{it} \text{ is fixed to one}\}$
$C' \leftarrow C - \sum_{i\in\mathcal{I}'} s_i$
$L' \leftarrow L - \sum_{i\in\mathcal{I}''} s_i$
$\xi \leftarrow \text{cardinality}_{i\in\mathcal{I}, t\in\mathcal{T}}\{r_{it} | r_{it} > 0 \text{ and } r_{it} < 1\}$
$\eta \leftarrow \text{cardinality}_{i\in\mathcal{I}, t\in\mathcal{T}}\{w_{it} | w_{it} > 0 \text{ and } w_{it} < 1\}$

Algorithm 5 Removal of non-compatible columns

for $t = 1$ to T **do**
$\mathcal{I}' \leftarrow \{i \in \mathcal{I} | x_{it0} \text{ is fixed to one}\}$
$\mathcal{I}'' \leftarrow \{i \in \mathcal{I} | y_{it} \text{ is fixed to one}\}$
$C' \leftarrow C - \sum_{i\in\mathcal{I}'} s_i$
$L' \leftarrow L - \sum_{i\in\mathcal{I}''} s_i$
for $i \in \mathcal{I} \backslash \mathcal{I}'$ **do**
if $s_i > L'$ **then**
$x_{it0} \leftleftarrows 0$ in SP_i
$\chi_{i\ell} \leftleftarrows 0$ in RMP if $u_{it\ell} = 1$, $\ell \in \mathcal{L}'_i$
for $i \in \mathcal{I} \backslash \mathcal{I}''$ **do**
if $s_i > C'$ **then**
$y_{it} \leftleftarrows 0$ in SP_i
$\chi_{i\ell} \leftleftarrows 0$ in RMP if $b_{it\ell} = 1$, $\ell \in \mathcal{L}'_i$

Algorithm 6 Framework of RCGA

```
STOP ← 0
while (STOP= 0) do
    Apply Algorithm 2 to obtain χ* subject to DR decisions made
    if (χ* is an integer solution) then
        STOP ← 1
    else
        Apply Algorithm 3
```

the deviation with respect to the global optimum does not exceed the deviation from the upper bound. We will see that, numerically, using the upper bound remains accurate in gauging the optimality gap.

We use content-specific and time-specific utility functions based on the literature [9], [49], [55]. Specifically, for each content one of the following functions is randomly selected (cf. the example functions given in Section 3.1): $f_{ita} = \frac{\epsilon_i - p_{it}}{T} a + p_{it}$, $f_{ita} = \frac{p_{it}}{a+1}$, and $f_{ita} = \frac{1}{e^{a p_{it}}}$. By the first function, the utility decreases linearly in the AoI variable a, in the latter two cases (inverse function and inverse exponential function) the utility decreases non-linearly in the AoI. The functions are made content-specific by varying parameter ϵ_i and popularity p_{it}. We remark that the performance in terms of optimality and solution time 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.

A target application of ACOP is caching at network edge. The cached contents could be local copies of a database, or popular video clips stored at a base station, etc. Such applications are studied in [56], [57], and in [2] from an AoI perspective. We set the instance parameters similar to those in the references. The number of content items is up to 100 (see [2]) with sizes generated in interval $[1, 10]$. The number of time slots is set to 24, each with length of one hour [56]. The cache and backhaul capacities are set to $C = 0.5 \sum_{i \in \mathcal{I}} s_i$ and $L = \rho \sum_{i \in \mathcal{I}} s_i$ respectively, where parameter ρ steers the backhaul capacity in relation to the total size of content items. Parameter $v_{it\tau}$ is set to be f_{it0} for $\tau = 1$ and grows linearly in τ. We will vary parameters I, T, and ρ to study their impact on the overall utility. The instances as well as the code used for the experiments are made available on the web [58].

7.1 Algorithm Performance in Utility

Figs. 3-5 and Figs. 6-8 show the performance results for the small-size and large-size problem instances, respectively. In Figs. 3-5, the magenta line represents the global optimum computed using ILP (2). In Figs. 6-8 the black line represents the upper bound. In all figures, the green and blue lines represent the utility achieved by RCGA and GA, respectively. Overall, RCGA delivers close-to-optimal solutions (with a few percent of deviation from optimum). For GA, the deviation from optimality is significantly larger. Moreover, it can be seen that, the results for small-size problem instances are consistent with those for larger problem size. Thus we will mainly discuss the results for small-size problem instances, even though we will also comment on the difference when the instance size grows.

Fig. 3 shows the impact of the number of content items on utility. Apparently, the overall utility increases with the number of items. This is because when there are more content items, there are more opportunities to exploit the item-specific utility values in optimizing the cache. However GA is less capable of doing so in comparison to RCGA, as the optimality gap of RCGA is consistently around 3% only, whereas the optimality gap of GA is about 13%.

The overall utility will obviously increase for a longer caching time horizon. This is seen in Fig. 4. RCGA and GA offer solutions that are approximately 2% and 12% from global optimality for values of T from 8 to 12. These optimality gap values are quite constant over T, and the reason is that extending the time has little structural effect on ACOP.

Fig. 5 shows the impact of ρ on utility. Larger ρ means higher backhaul capacity and thus higher utility as well. There is a saturation effect, however, as when ρ approaches 0.5 (which corresponds to the cache capacity), the backhaul capacity hardly constrains the performance. For the lowest ρ-value of 0.1, both algorithms have an optimality gap of 20%, and in fact GA gives slightly better result. The increase in optimality gaps of both algorithms shows that ACOP becomes noticeably harder if very few content items can be updated per time slot.

The results in Figs. 6-8 follow a similar trend as the first three performance figures. Interestingly, for large-size ACOP instances, RCGA delivers better performance, as the achieved utility by RCGA virtually overlaps with the upper bound. We believe this is an effect of the knapsack structure of ACOP. When the number of content items is small, even few sub-optimal caching and updating decisions made by RCGA have noticeable impact on the overall sub-optimality. This is less an issue with many content items. GA, however, has a worse performance for larger-size problem instances. The observation demonstrates the strength of RCGA over the simple greedy schedule. Finally, our performance evaluation using the upper bound is accurate, because the global optimum is between the utility by RCGA and the upper bound that are almost overlapping.

7.2 Additional Results

In Table 1, we report the running times of solving the ILP (2) and RCGA, as well as the optimality gap values, for three groups of instances that differ in the number of contents I and the number of time slots T. For ILP (2), we set a time limit of 50, 100, and 300 seconds, for the easy, medium, and hard instances, respectively. The optimality gap of ILP is the one reported by the Gurobi optimizer; this value is positive if the running time hits the time limit. For RCGA, the optimality gap is that to the upper bound (hence the true deviation to global optimum may be smaller). For each combination of I and T, the table reports the average results of five instances.

From the table, we observe that for easy instances, computing the global optimum via ILP is feasible in terms of time. For these instances, RCGA has significantly lower computing time though there is a small optimality gap. For medium and hard instances, global optimum is beyond reach, as the solver hits the time limit with large optimality gap except for one instance. From the gap values, using the solver as a heuristic for finding good solutions does not appear to be a viable approach. RCGA, on the other hand, is able to deliver close-to-optimal solutions with 1% or smaller optimality gap in much less time. Another observation from these results is that the difficulty of problem solving with respect to problem size originates mainly from the number of time slots T rather than the number of content items I.

Fig. 9 shows the progress of solving ILP (2) using Gurobi optimizer for a representative instance with $I = 100$ and $T =$

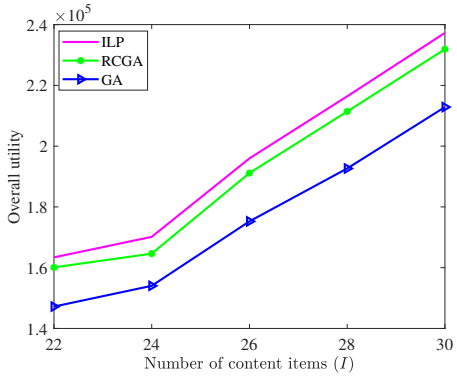

Fig. 3. Impact of I on utility when $T = 10$, $L = 0.3\sum_{i\in\mathcal{I}} s_i$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

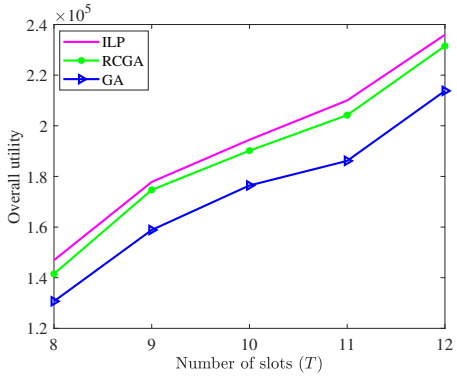

Fig. 4. Impact of T on utility when $I = 26$, $L = 0.3\sum_{i\in\mathcal{I}} s_i$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

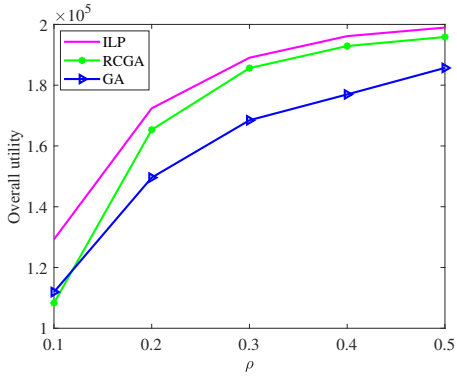

Fig. 5. Impact of ρ on utility when $I = 26$, $T = 10$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

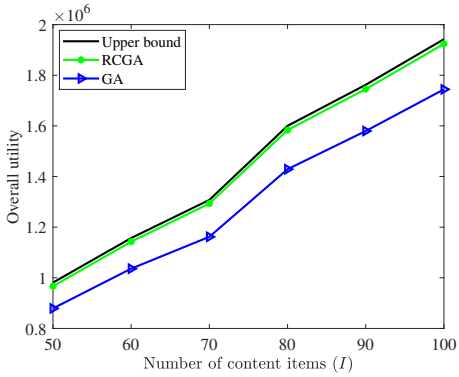

Fig. 6. Impact of I on utility when $T = 24$, $L = 0.3\sum_{i\in\mathcal{I}} s_i$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

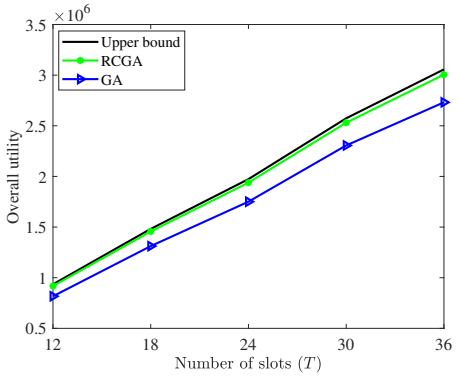

Fig. 7. Impact of T on utility when $I = 100, L = 0.3\sum_{i\in\mathcal{I}} s_i$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

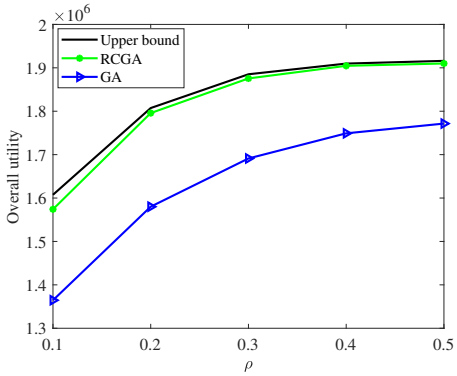

Fig. 8. Impact of ρ on utility when $I = 100$, $T = 24$, and $C = 0.5\sum_{i\in\mathcal{I}} s_i$.

Table 1: Average solution time and optimality gap.

Class	I	T	ILP gap (%)	ILP time (s)	RCGA gap (%)	RCGA time (s)
Easy	50	6	0.00	2.88	1.24	0.82
	75	6	0.00	4.23	0.92	0.91
	100	6	0.00	6.49	1.55	0.95
	50	9	0.00	23.94	1.31	2.79
	75	9	0.00	31.93	0.84	3.65
	100	9	0.00	49.51	0.73	5.05
Medium	50	12	0.00	85.00	0.24	6.96
	75	12	0.20	100.00	0.44	8.06
	100	12	7.13	100.00	0.55	11.43
	50	18	33.30	100.00	1.00	22.40
	75	18	33.92	100.00	0.58	28.01
	100	18	34.09	100.00	0.56	37.41
Hard	50	24	63.26	300.00	0.72	63.25
	75	24	74.24	300.00	0.64	82.32
	100	24	79.70	300.00	0.40	99.36
	50	30	103.82	300.00	0.75	154.16
	75	30	109.74	300.00	0.63	186.29
	100	30	115.12	300.00	0.70	201.36

18. The two curves show the best bound and incumbent (best-known integer solution) over time. The gap between the two is over 100% at the beginning, and drops to approximately 30% within one minute. The solver makes then small progress in more than 10 minutes. By the end of 15 minutes, the gap is reduced to 6%. It is observed, however, that the solver finds a good integer solution in about 100 seconds, and the difficulty is to improve the bound. For the same instance, RCGA has a running time of about 30 seconds and an optimality gap of 0.46% that is better than the final gap in the figure.

For column generation, a classic approach to obtain an integer solution is to solve the integer version of the problem with the restriction to those variables generated in the column generation process. We have carried out additional tests and comparisons based on this approach. It turns out that solving the resulting integer problem remains very challenging in solution time. However, if we allow pre-mature termination by an optimality

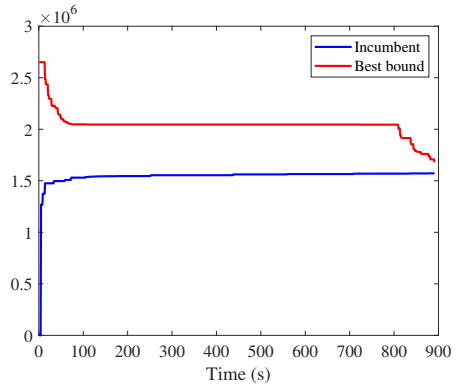

Fig. 9. An illustration of the progress of solving ILP.

tolerance of a couple of percent, the Gurobi optimizer is able to complete the solution process much faster, and in many cases it leads to comparable solutions but less time in comparison to RCGA. There are however a couple of disadvantages of such an approach, namely theoretically it has exponential solution time, and a suitable tolerance parameter is not known a priori hence it requires parameter tuning.

Another variation of the algorithm is randomized rounding. We have done some computational tests in which the fractional variable for rounding is selected randomly, and then rounded to zero or one depending on the fractional value. The tests show that such a scheme leads to similar computing time, but slightly worse performance in terms of utility.

8 Extension to Cyclic Schedule

Suppose the caching and updating schedule is cyclic. Namely, the schedule repeats itself for every T time slots. Hence the AoI of an item in time slot one depends on the scheduling decisions made for the item in later time slots. In this section, we discuss applying our results and algorithmic notions to cyclic schedule.

The NP-hardness of cyclic scheduling clearly remains, as the proof of Theorem 1 applies directly. The tractability stated in Theorem 4 can be generalized to cyclic schedule based on the notion of flow circulation, which refers to a flow pattern satisfying flow balance at every node of a graph without source or destination nodes. We modify the network graph (in Fig. 1) such that the last node n_T coincides with the first node n_0. Moreover, each node $n_i, i = 1, \ldots, T-1$ is split into two nodes n_i and n_i' connected by a single arc (n_i, n_i') of tuple $(0, I, I)$, i.e., there will be exactly I flow units on this arc for every time slot. Finding the optimal cyclic schedule corresponds then to solving the minimum-cost circulation flow problem for which the optimum can be computed in polynomial time (e.g., [59]).

Adapting the ILP formulation in Section 5 to cyclic scheduling is easy; this amounts to adding an additional constraint of type (2d) to connect together time slots 1 and T. The reformulation (3) and our algorithm RCGA remain applicable. The difference from acyclic schedule lies in the subproblem. Namely, instead of finding a shortest path with graph construction shown in Fig. 2, the subproblem consists in finding minimum cost circulation in a modified graph as discussed above.

Finally, we remark that the underlying rationale of the greedy solution in Section 3.2 does not appear very logical for a cyclic schedule. The greedy algorithm works slot by slot, and, for each slot, the algorithm uses the AoI values of the previous slot as input. With cyclic scheduling, this input is not available as it depends on the whole scheduling solution. Nevertheless, the greedy acyclic schedule is still a valid solution to be used as a cyclic schedule, though the true utility values need to be evaluated afterward.

9 Conclusions

We have considered the optimization problem of time-dynamic caching where the performance metric is age-centric. Our work has led to findings in problem complexity. In addition, column generation offers an effective approach for problem solving in terms of obtaining close-to-optimal solutions. As another concluding remark, we believe our results and algorithm admit extensions to a couple of other related performance functions. An example is the minimization of average AoI. Moreover, we remark that a key problem structure is the knapsack constraint due to cache capacity, and for the regular knapsack problem there exist good approximations including a fully polynomial-time approximation scheme [60]. Hence an interesting direction of research is to investigate if good approximation algorithms of the problem can be derived. Finally, we remark that column generation can be embedded into a branch-and-bound algorithm (or more commonly known as branch-and-price in the context of column generation). An effective implementation of the algorithm constitutes an extension of the current work.

References

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

[2] 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.

[3] R. D. Yates and S. Kaul, "Real-time status updating: multiple sources," in *IEEE International Symposium on Information Theory (ISIT)*, 2012, pp. 2666–2670.

[4] C. Kam, S. Kompella, and A. Ephremides, "Age of information under random updates," in *IEEE International Symposium on Information Theory (ISIT)*, 2013, pp. 66–70.

[5] M. Costa, M. Codreanu, and A. Ephremidesy, "Age of information with packet management," in *IEEE International Symposium on Information Theory (ISIT)*, 2014, pp. 1583–1587.

[6] N. Pappas, J. Gunnarsson, L. Kratz, M. Kountouris, and V. Angelakis, "Age of information of multiple sources with queue management," in *IEEE International Conference on Communications (ICC)*, 2015, pp. 5935–5940.

[7] M. Costa, S. Valentin, and A. Ephremides, "First steps to understand the age of channel information," in *Proc. of 1st KuVS Workshop on Anticipatory Networks*, 2014, pp. 4–6.

[8] R. D. Yates, "Lazy is timely: Status updates by an energy harvesting source," in *IEEE International Symposium on Information Theory (ISIT)*, 2015, pp. 3008–3012.

[9] 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, pp. 7492–7508, 2017.

[10] Y. Sang, B. Li, and B. Ji, "The power of waiting for more than one response in minimizing the age-of-information," in *IEEE Global Communications Conference (GLOBECOM)*, 2017, pp. 1–6.

[11] I. Kadota, E. Uysal-Biyikoglu, R. Singh, and E. Modiano, "Minimizing the age of information in broadcast wireless networks," in *IEEE 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016, pp. 844–851.

[12] Q. He, D. Yuan, and A. Ephremides, "Optimal link scheduling for age minimization in wireless systems," *IEEE Transactions on Information Theory*, vol. 64, pp. 5381–5394, 2018.

[13] A. Kosta, N. Pappas, and V. Angelakis, "Age of information: A new concept, metric, and tool," *Foundations and Trends in Networking*, vol. 12, pp. 162–259, 2017.

[14] Y. Sun, E. Uysal-Biyikoglu, and S. Kompella, "Age-optimal updates of multiple information flows," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 136–141.

[15] C. Kam, S. Kompella, G. Nguyen, J. Wieselthier, and A. Ephremides, "Towards an effective age of information: remote estimation of a Markov source," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 367–372.

[16] J. Sun, Z. Jiang, S. Zhou, and Z. Niu, "Optimizing information freshness in broadcast network with unreliable links and random arrivals: an approximate index policy," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*, 2019, pp. 115–120.

[17] S. Farazi, A. G. Klein, and D. R. Brown III, "Average age of information for status update systems with an energy harvesting server," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 112–117.

[18] R. D. Yates, "Age of information in a network of preemptive servers," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 118–123.

[19] E. Najm and E. Telatar, "Status updates in a multi-stream M/G/1/1 preemptive queue," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 124–129.

[20] X. Wang and L. Duan, "Dynamic pricing for controlling age of information," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 962–966.

[21] J. P. Champsti, M. H. Mamduhi, K. H. Joansson, and J. Gross, "Performance characterization using AoI in a single-loop networked control system," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 197–203.

[22] M. Klugel, M. H. Mamduhi, S. Hiche, and W. Kellerer, "AoI-penalty minimization for networked control systems with packet loss," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 189–196.

[23] Q. He, G. Dán, and V. Fodor, "On emptying a wireless network with minimum-energy under age constraints," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 668–673.

[24] S. Farazi, A. G. Klein, and D. R. Brown III, "Fundamental bounds on the age of information in general multi-hop interference networks," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*, 2019, pp. 96–101.

[25] Y. Dong, Z. Chen, and P. Fan, "Uplink age of information of unilaterally powered two-way data exchanging systems," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 559–564.

[26] Q. He, G. Dán, and V. Fodor, "Minimizing age of correlated information for wireless camera networks," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 547–552.

[27] J. Liu, X. Wang, B. Bai, and H. Dai, "Age-optimal trajectory planning for UAV-assisted data collection," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 553–558.

[28] G. Ahani, D. Yuan, and Y. Zhao, "Age-optimal UAV scheduling for data collection with battery recharging," 2020. [Online]. Available: https://arxiv.org/abs/2005.00252

[29] N. Pappas, Z. Chen, and M. Hatami, "Average AoI of cached status updates for a process monitored by an energy harvesting sensor," in *54th Annual Conference on Information Sciences and Systems (CISS)*, 2020, pp. 1–5.

[30] W. Gao, G. Cao, M. Srivatsa, and A. Iyengar, "Distributed maintenance of cache freshness in opportunistic mobile networks," in *IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, 2012, pp. 132–141.

[31] J. Zhong, R. Yates, and E. Soljanin, "Minimizing content staleness in dynamo-style replicated storage systems," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2018, pp. 361–366.

[32] 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.

[33] S. Zhang, L. Wang, H. Luo, X. Ma, and S. Zhou, "AoI-delay tradeoff in mobile edge caching with freshness-aware content refreshing," 2020. [Online]. Available: https://arxiv.org/abs/2002.05868

[34] X. Wu, X. Li, J. Li, P. C. Ching, V. C. M. Leung, and H. V. Poor, "Caching transient content for iot sensing: Multi-agent soft actor-critic," 2020. [Online]. Available: https://arxiv.org/abs/2008.13191

[35] A. Maatouk, M. Assaad, and A. Ephremides, "Minimizing the age of information: NOMA or OMA?" in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)*, 2019, pp. 102–108.

[36] E. T. Ceran, D. Gündüz, and A. György, "A reinforcement learning approach to age of information in multi-user networks," in *IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2018, pp. 1967–1971.

[37] C. Kam, S. Kompella, and A. Ephremides, "Learning to sample a signal through an unknown system for minimum AoI," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 177–182.

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

[39] 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.

[40] L. Yang, Y. Zhong, F.-C. Zheng, and S. Jin, "Edge caching with real-time guarantees," 2019. [Online]. Available: https://arxiv.org/abs/1912.11847

[41] H. Tang, P. Ciblat, J. Wang, M. Wigger, and R. Yates, "Age of information aware cache updating with file- and age-dependent update durations," 2019. [Online]. Available: https://arxiv.org/pdf/1909.05930.pdf

[42] M. Bastopcu and S. Ulukus, "Maximizing information freshness in caching systems with limited cache storage capacity," 2020. [Online]. Available: https://arxiv.org/abs/2005.10683

[43] ——, "Information freshness in cache updating systems," 2020. [Online]. Available: https://arxiv.org/abs/2004.09475

[44] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki, "Tailgate: handling long-tail content with a little help from friends," in *the 21st international conference on World Wide Web*, 2012, pp. 151–160.

[45] D. S. Berger, P. Gland, S. Singla, and F. Ciucu, "Exact analysis of TTL cache networks," *Performance Evaluation*, vol. 79, pp. 2–23, 2014.

[46] A. Reiffers-Masson, E. Hargreaves, E. Altman, W. Caarls, and D. S. Menasché, "Timelines are publisher-driven caches: Analyzing and shaping timeline networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 44, pp. 26–29, 2016.

[47] E. Hargreaves, C. Agosti, D. Menasché, G. Neglia, A. Reiffers-Masson, and E. Altman, "Fairness in online social network timelines: Measurements, models and mechanism design," *Performance Evaluation*, vol. 129, pp. 15–39, 2019.

[48] G. Neglia, D. Carra, M. Feng, V. Janardhan, and P. Michiardi, "Access-time-aware cache algorithms," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 2, pp. 1–29, 2017.

[49] 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.

[50] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Pearson Press, 2014.

[51] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, pp. 1635–1648, 2017.

[52] GUROBI, "GUROBI Optimizer 9.0," http://www.gurobi.com/, 2020.

[53] M. Lubecke and J. Desrosiers, "Selected topics in column generation," *Operations Research*, vol. 53, pp. 1007–1023, 2004.

[54] M. Karamano and G. Cornuéjols, "Branching on general disjunctions," *Mathematical Programming*, vol. 128, pp. 403–436, 2009.

[55] S. Ioannidis, A. Chaintreau, and L. Massoulie, "Optimal and scalable distribution of content updates over a mobile social network," in *IEEE Conference on Computer Communications (INFOCOM)*, 2009, pp. 1422–1430.

[56] S. Shukla and A. A. Abouzeid, "Proactive retention aware caching," in *IEEE Conference on Computer Communications (INFOCOM)*, 2017, pp. 1–9.

[57] M. K. Kiskani and H. R. Sadjadpour, "Capacity of cellular networks with femtocache," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 9–14.

[58] https://github.com/astra-uu-se/TMC, 2020.

[59] É. Tardos, "A storngly polynomial minimum cost circulation algorithm," *Combinatorica*, vol. 5, pp. 247–255, 1985.

[60] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, 2014.

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.

Sumei Sun (F'16) is currently the Head of the Communications and Networks ~~Cluster,~~ Institute for Infocomm Research, ~~Singapore,~~ and a Lead Principal Investigator of the Industrial Internet of Things Research Program with the Agency for Science, Technology, and Research (A*STAR), Singapore. She is also an ~~A~~djunct ~~P~~rofessor with the National University of Singapore, ~~Singapore~~. Her current research interests include cognitive communications and networks, next-generation wireless communications, and industrial Internet of Things. ~~Ms.~~ Sun is a Distinguished Speaker of the IEEE Vehicular Technology Society from 2018 to 2021, ~~the Vice~~ Director of the IEEE Communications Society Asia–Pacific Board, and the Chapter Coordinator of the Asia–Pacific Region in the IEEE Vehicular Technologies Society. She is ~~also serving as a member of the IEEE Communications Society Globecom/ICC Management and Strategy Standing Committee, an Area Editor of the IEEE Transactions on Vehicular Technology,~~ and a member of the IEEE Transactions on Wireless Communications Steering Committee.