Deadline and Buffer Constrained Knapsack Problem

Anis Elgabli and Vaneet Aggarwal, *Senior Member, IEEE*

***Abstract*—In this paper, we formulate a problem that is a variant of the knapsack problem. Even though the problem is NP-hard in general, we consider a special case of the problem where the problem is in P. For this special case, the proposed algorithm is linear time complexity in the number of bins. The proposed framework is a generalization of the framework that has been used recently in the context of finding rate adaptation algorithms for video streaming.**

***Index Terms*—Knapsack problem, video streaming, polynomial time algorithm.**

I. Introduction

Combinatorial optimization has important applications in several fields, including, resource management, scheduling, machine learning, and software engineering [1]–[4]. In combinatorial optimization, the objective is to find an optimal solution from a finite set of solutions [5]. In most combinatorial problems, exhaustive search which searches for the optimal solution in a discrete set is not feasible. Some of the famous combinatorial problems are the traveling salesman problem (TSP), the minimum spanning tree problem (MST), and the knapsack problem [5].

In the knapsack problem, given a set of items, each with a weight and a value, which items should be included in a collection so as to maximize total value given a weight constraint must be determined. The knapsack problem has been a subject of study since 1897 [6].

In this paper, we consider a problem which is an extension of the knapsack problem. Fig 1 compares our problem with knapsack problem. Each item has multiple components, and each component must be packed in the C bins, each of certain capacity. There are four types of constraints in the packing, namely Component Dependency Constraint, Deadline Constraint, Buffer Constraint, and Bin-Capacity Constraint. We formulate packing the items into bins efficiently to maximize the total value of the components packed. In the special case, when the bin capacity for bins 2 to C is zero, and each item has a single component, the problem reduces to a knapsack problem. Thus, the problem is harder than the NP-hard knapsack problem. We consider a special case of the problem with certain constraints on the weights and values of the components. With such constraints, we show that the problem can be optimally solved with an algorithm, whose complexity is linear in the number of bins. Thus, the NP-hard problem with simple limitations on weights and values can reduce to a polynomial-time solvable problem.

We note that the video streaming formulation proposed in [7] is a special case of this particular special case. Elgabli *et al.* [7] formulated the problem of adaptive video streaming with the knowledge of future bandwidth. The goal of the scheduling algorithm is to determine up to which layer we need to fetch for each chunk such that the overall quality-of-experience (QoE) is maximized and the number of stalls or skipped chunks is minimized. The proposed algorithm can be implemented both with Scalable Video Coding [7] and Advanced Video Coding [8]. Further, it was shown to achieve minimum re-buffering (stall) time and the maximum average playback rate in every single trace as compared to the original dash.js rate adaptation scheme, Festive, BBA, RB, and FastMPC algorithms [8]. The key advantage of the proposed algorithm is a linear time complexity, which makes the algorithm easy to run online. In contrast, the previous optimization-based algorithms are not able to run optimally in that low time complexity. Further, the formulation models the diminishing returns of user-perceived quality of experience (QoE) with the playback quality [9]. In this paper, we aim to generalize the formalism in [7] and [8] to a special case of Knapsack problem. The general formalism in this paper can allow for applications of such knapsack problems beyond those in video streaming.

Manuscript received September 29, 2018; revised February 26, 2019; accepted February 27, 2019. This paper was recommended by Associate Editor W. Lin. *(Corresponding author: Vaneet Aggarwal.)*

A. Elgabli was with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA. He is now with the Center of Wireless Communications, University of Oulu, 90014 Oulu, Finland (e-mail: anis.elgabli@oulu).

V. Aggarwal is with the School of Industrial Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: vaneet@purdue.edu).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2019.2902759

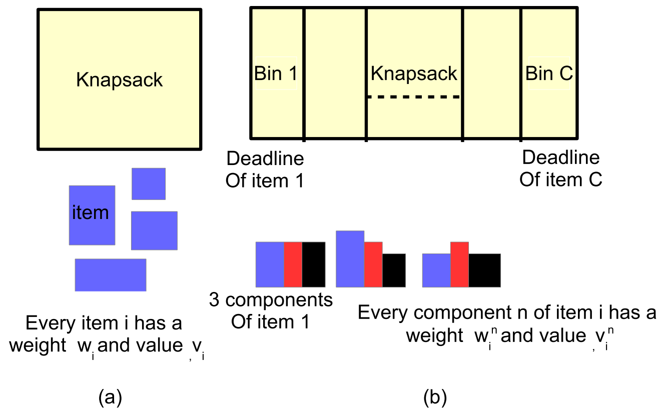
Knapsack
item
Every item i has a weight w_i and value v_i
(a)
Bin 1
Knapsack
Bin C
Deadline Of item 1
Deadline Of item C
3 components Of item 1
Every component n of item i has a weight w_i^n and value v_i^n
(b)

Fig. 1. (a) 0-1 Knapsack. (b) Deadline and Buffer constrained knapsack.

The rest of the paper is organized as follows. Section II provides the formulation of the problem, with a proof of NP-hardness. Section III describes the special case of weights and values considered, and the proposed algorithm which is shown to be optimal. Section IV describes the application of the problem in adaptive bit-rate video streaming in brief. Section V concludes the paper. Appendices A-C (see the Supplementary Material) provide the proofs for the results. Appendix D-G (see the Supplementary Material) explain the application of the problem to video streaming in [7] with the evaluation results.

II. Problem Formulation

This section formally describes the problem. We consider C items that must be packed in C bins. Each item has N components (labeled as components 0 to $N-1$ for each item i), which can be packed in the bins (See Fig. 1). Every component n of item i has a weight of w_i^n and a value of v_i^n. We consider the following constraints:

1) *Component Dependency Constraint:* If component $n-1$ of an item is not packed, its component n can't be packed.
2) *Deadline Constraint:* Any component of item i cannot be packed in a bin with index greater than i.
3) *Buffer Constraint:* For any i, the bins with index $\leq i$ cannot pack components from more than B_{max} of the items with index $>i$.
4) *Bin-Capacity Constraint:* The weight of components packed up-to bin t cannot cross $\sum_{j=1}^{t} B_j$, where B_j is the capacity of bin j.

We note that a bin can have partial amounts of component of an item. Thus, the component can be spread over multiple bins. More formally, let $x_{i,n,t}$ be the amount of n-th component of the i-th item that is packed in bin t. Then, $x_{i,n,t}$ could be in [0, 1]. However, since a component is either fully packed or not at all, we have $\sum_{t=1}^{C} x_{i,n,t} \in \{0, 1\}$.

The objective is to maximize the total value subject to the bin-Capacity, deadline, buffer, and component dependency constraints. The overall optimization problem can be written as follows.

$$\textbf{maximize:} \sum_{n=0}^{N-1}\sum_{i=1}^{C}\sum_{t=1}^{C} v_i^n x_{i,n,t} \tag{1}$$

$$\text{s.t} \sum_{n=0}^{N-1}\sum_{i=1}^{C}\sum_{j=1}^{t} x_{i,n,j} w_i^n \leq \sum_{j=1}^{t} B_j, \quad \forall t \tag{2}$$

$$\sum_{t=1}^{C} x_{i,n,t} \leq \sum_{t=1}^{C} x_{i,n-1,t}, \quad \forall i, n \tag{3}$$

$$x_{i,n,t} \geq 0, \quad \forall i, n, t \tag{4}$$

$$\sum_{j=i+1}^{C} \mathbf{I}\Big(\sum_{t=1}^{i}\sum_{n=0}^{N-1} x_{j,n,t} > 0\Big) \leq B_{max}, \quad \forall i \tag{5}$$

$$x_{i,n,t} = 0, \quad \forall i, n, t > i \tag{6}$$

$$\sum_{t=1}^{C} x_{i,n,t} \in \{0, 1\}, \quad \forall i, n \tag{7}$$

where $\mathbf{I}(.)$ is an indicator function, where $\mathbf{I}(x) = 1$ if x is true and 0 otherwise. (2) represents the bin-capacity constraints, (3) represents the component dependence constraint, (5) is the buffer constraint, and (6) is the deadline constraint. Further, (7) is the integer constraints, representing that component n of item i is either fetched completely, or not at all.

We note that in addition to discrete constraints, the proposed problem also has a non-convex constraint due to the presence of the indicator function in (5). We now prove that problem (1)-(7) is NP-Hard. In order to prove that the problem is NP-Hard, we need first to prove that it is NP. Given a solution $x_{i,n,t}$, verifying the constraints in (1)-(7) can be done in $O(N^2C)$ time, and thus the problem is in NP.

We see that 0-1 Knapsack problem is a special case of our problem. It is the case when every item consists of one component ($N = 1$), $B_t = 0$ for $t \geq 2$, and $B_{max} = \infty$. In this case, there is no component dependence, deadline, or buffer constraint. There is a single bin where the items can be packed, and it has a size constraint. Further, $x_{i,n,t} = 0$ for $t > 1$. Thus, the problem in a special case is equivalent to a knapsack problem. Since the problem in a special case is harder than a NP-hard problem and the problem is in NP, the proposed problem is NP-hard.

Algorithm 1 Deadline and Buffer Aware Bin Packing Algorithm

```
1: **Input:** $w^n$, $B_{max}$
2: **Output:** $X(i)\forall i$: The size of all packed components of item $i$, $I_n$: set contains the indices of the components that can have up to their $n$-th component packed.
3: **Initialization:**
4: $X_n = \sum_{m=0}^{n} w^n$ cumulative size of every item up to component $n$
5: $c(j) = \sum_{j'=1}^{j} B(j')$ cumulative bin sizes up to the $j$-th bin
6: $t(i) = 0, \forall i$, first bin in which item $i$ can be packed
7: $a(i) = 0, \forall i$, size of lower level components of item $i$ packed at its lower deadline time $t(i)$
8: $e(j) = B_j, \forall j$, remaining size at bin j after all non skipped components are packed according to lower level component decisions
9: $X(i) = 0$
10: $bf(j) = 0, \forall j$, buffer size at time $j$
11: **For each component,** $n = 0, \cdots, N$
12:     $[X, I_n] = backward(B, X, X_n, C, B_{max}, bf, t, c, a, e)$
13:     $[t, a, e] = forward(B, X, C, B_{max}, bf, I_n)$
```

III. Considered Special Case and Polynomial Time Algorithm

In this section, we will describe the considered special case over the weights and values. Further, a novel algorithm is described for the problem, and its optimality will be derived.

A. Special Case

In this subsection, we will discuss a special case of the problem (1)-(7). The case when the following conditions are satisfied:

1) $w_i^n = w^n \forall i$
2) $v_i^n = v^n \forall i$
3) $C \sum_{k=a+1}^{N-1} v^k < v^a$, for all $a = 0, \cdots, N-2$

The first condition implies that the weight of component n for each item is the same. The second condition implies that value of component n is same for all items. The third condition implies that the combined value of all components $> a$ is lower than component a of any item. Thus, this condition implies diminishing returns in obtaining larger components. For the special case of video streaming, this constraint has been explained in Appendix E (see the Supplementary Material), where we demonstrate that this models user's QoE being concave in the playback rate [9].

We note that even though this is a special case, it is not clear a priori whether this case will result in an optimal polynomial-time algorithm. This is because the components placements depend on each other and packing of components n influence the packing of components $> n$ even with these conditions.

B. Proposed Algorithm

We now show when the above three conditions are satisfied, the problem can be solved optimally in polynomial time. In fact under these assumptions, this paper provides an algorithm that solves problem (1)-(7) with a complexity that is O(NC).

The algorithm is summarized in Algorithm 1. The algorithm defines priority levels, so the components 0 of each item belongs to the highest priority level (level-0). Therefore, since there are N components for every item, there will be N levels. The algorithm performs backward and forward scans (as explained later in this section) at each level given the decisions of the lower levels.

Running backward scan at n-th level (Algorithm 2) finds the maximum number of items that can have their n-th level component packed given the decisions of the previous levels. Then, forward

Algorithm 2 Backward Algorithm

```
1: Input: $B, X, X_n, C, B_{max}, bf, t, c, a, e$
2: Output: $X(i)$, $I_n$.
3: Initialization:
4: $i = j = C$
5: initialize $bf(j)$ to zeros $\forall j$.
6: while ($j > 0$ and $i > 0$) do
7:    if $j <= i$ then
8:       if ($bf(i) = B_{max}$) then  $i = i - 1$
9:       if $j$ is the first time to pack item $i$ from back then
10:         if ($t(i) = 0$) then
11:            $rem1 = c(j) - c(1) + e(1)$, $rem2 = rem1$
12:         else
13:            $rem2 = c(j) - c(t(i))$, $rem1 = rem2 + e(t(i)) + a(i)$
14:         end if
15:         if ($rem1 < X_n(i)$) then
16:            if ($X(i) > 0$) then $X_n(i) = X(i)$ else  $i = i - 1$
17:         else
18:            if ($rem2 < X_n(i)$) and $rem1 \geq X_n(i)$) then
19:               $e(t(i)) = e(t(i)) + rem1 - X_n$
20:            end if
21:            $X(i) = X_n(i)$, $I_n \leftarrow I_n \cup i$
22:         end if
23:      end if
24:      $packed = min(B_j, X_n(i))$, $B_j = B_j - fetched$
25:      $X_n(i) = X_n(i) - packed$
26:      if ($X_n(i) > 0$) then  $bf(j) = bf(j) + L$
27:      if  ($X_n(i) = 0$) then  $i = i - 1$
28:      if  ($B_j = 0$) then  $j = j - 1$
29:   else
30:      $j = j - 1$
31:   end if
32: end while
```

Algorithm 3 Forward Algorithm

```
1: Input: $B, X, C, deadline, Bm, bf, I_n$
2: Output: $t(i)$, $a(i)$, $e(j)$.
3: $j = 1$, $k = 1$
4: while $j \leq C$ and $k \leq max(I_0)$ (last item to pack) do
5:    $i = I(k)$
6:    if $i = 0$ then  $k = k + 1$
7:    if $j \leq i$ then
8:       if  ($bf(j) = B_m$) then  $j = j + 1$
9:       $packed = min(B_j, X(i))$
10:      if $j$ is the first time item $i$ is packed then
11:         $t(i) = j$,
12:         $a(i) = packed$
13:      end if
14:      $B_j = B_j - packed$
15:      $e(j) = B_j$, $X(i) = X(i) - packed$
16:      if  $X(i) > 0$ then  $bf(j) = bf(j) + L$
17:      if  $X(i) = 0$ then  $k = k + 1$
18:      if  $B_j = 0$ then  $j = j + 1$
19:   else
20:      $k = k + 1$
21:   end if
22: end while
```

scan (Algorithm 3) is run to simulate packing items in sequence as early as possible, so the start time for packing every item i (the lower deadline $t(i)$) is found. Lower ($t(i)$) and Upper ($deadline(i)$) deadlines will be used to check if the next component of item i can be packed or not.

1) Backward Algorithm for Level 0: Given the bin sizes, deadlines, and the buffer size, the algorithm simulates packing the level-0 components of all items starting from the last towards the first item. The deadline of the last item is the starting time slot of the backward algorithm scan. The goal is to have items packed closer to their deadlines. For every item i, the backward algorithm checks the bin sizes and the buffer; if the component can be packed and the buffer constraint is not violated, then component 0 of item i is selected to be packed (line 18-22). The algorithm keeps checking this feasibility to select components to be fetched. If a component 0 of an item i' is not selected to be packed, one of the two scenarios could have happened. The first is the violation of **buffer** constraint, where selecting the item to be packed would violate the buffer constraint. The second is the **bin-capacity** constraint where the size of the remaining bins is not enough for packing the component 0 of the item. This scenario also means that the component could not be packed by the deadline, so it can also be called deadline violation.

For buffer constraint violation, we first note that, there could be an item $i'' > i'$ in which if its component 0 is skipped will possibly lead to packing the component 0 of item i'. However, the backward algorithm decides to skip packing the component 0 of item i' (line 8). We note that since there is a buffer capacity violation, one of the items cannot be fetched and must be skipped. Note that since skipping the component 0 of any item will lead to skip all remaining components of the same items, we use skip item and component 0 of the item interchangeably. The reason of choosing to skip item i' rather than a one with higher index is that i' is the closest to its deadline. Therefore, i' is not better candidate to have its next component packed than any of the later ones.

In the second case of deadline/size violation, backward algorithm decides to skip items up 1 to i' since there is not enough bins. As before, since equal number of items need to be skipped any way, skipping earlier ones is better because it helps in increasing the potential of getting next components of later items.

2) Forward Algorithm for Level-0: Forward algorithm takes the decisions from the Backward step (the items that can have their component 0 packed), and simulates packing items in sequence starting from the first one. Items are packed as early as possible with the deadline, and buffer constraints being considered. The items that were not decided to be packed by the Backward Algorithm are skipped (any chunk $i \notin I_0$, line 6). This provides the earliest time slot when item i can be fetched ($t(i)$, line 10). This time is used as a **lower deadline** on the time allowed to pack chunk i when backward algorithm is run for the next component, so decisions of backward algorithm for component 0 of all items do not get violated when backward algorithm is run again for the next component (component 1, Level-1). Moreover, it provides the portion that can be packed of item i at its lower deadline $t(i)$ ($a(i)$, line 11), and, remaining size of the bin in which the item is packed in after all non skipped items are packed ($e(j)$, line 12).

Modifications for Next Levels: The same backward and forward steps are used for each level given the backward-forward decisions of the previous one. The key difference when the algorithm is run for next levels as compared to that for Level-0 is that the component n of an item is skipped if the component $n - 1$ was not packed. When running the backward algorithm for component 1 decisions, for every item i, we consider the remaining size starting from the bin at the lower deadline of that chunk $t(i)$, so previous decisions (component 0 decisions) of early items do not get violated. The same procedure is used to give all remaining component decisions when all the lower level decisions have already been made.

C. Optimality of the Proposed Algorithm

In this subsection, we will describe the optimality of the proposed algorithm, in the sense that the proposed algorithm achieves the optimal solution of the problem (1)-(7). We define in-order packing algorithm as the algorithm that fetches items in order based on the deadlines, such that all components of item i before that of item j for any $i < j$. We note that for any other feasible packing algorithm, we can convert it to an in-order packing algorithm with the same bin-capacity utilizations for each item. Thus, it is enough to prove that the algorithm is the best among any in-order packing algorithm. Getting the different items in-order helps the buffer and other constraints and would only improve the satisfiability of constraints. Thus, we can obtain the same objective while satisfying all the constraints. The proofs in the following thus only compare the proposed algorithm with the other in-order packing algorithms.

We say that n-th component of item i is skipped if $\sum_{t=1}^{C} x_{i,n,t} = 0$. Thus, this component is not packed in the bins. The following Lemma states that given the lower and upper deadlines $((t(i))$ and $i)$ of every item i, the $(n-1)$-th component decision, running backward algorithm for the n-th component maximizes the nth component objective, i.e., maximizes the number of items that have their n-th component packed.

Lemma 1: Given size decisions up to $(n-1)$-th level (X), and lower and upper deadlines ($t(i)$, and i) for every item i, backward algorithm achieves the minimum number of n-th component skips (or obtains the maximum number of n-th components) as compared to any feasible algorithm which packs the same components up to component $n-1$.

Proof: A detailed proof is provided in Appendix A (see the Supplementary Material). □

The above lemma shows that backward algorithm minimizes n-th component skips given the lower and upper deadline of every item. However, we still need to consider the optimality of the lower deadline. The following lemma shows that for any size decisions, the forward algorithm finds the optimal lower deadline on the packing time of any item.

Lemma 2: Among all algorithms with the same number of the n-th component skips, the proposed algorithm leaves the largest possible space for all candidate items to the next component. In other words, the proposed algorithm maximizes the resources to the next components among all algorithms that have same n-th component skips.

Proof: Proof is in the Appendix B (see the Supplementary Material). □

The above two lemmas show that the backward algorithm maximizes the number of items packed up to the n-th component and maximizes the number of candidate items to the next component given the lower and upper deadline on the packing time of every item. However, they do not tell us if that lower deadline is optimal or not. The following proposition shows that for given packing decisions, the forward algorithm finds the optimal lower deadline on the packing time of every item.

Proposition 1: If $t_f(i)$ is the earliest time to start packing item i using forward algorithm (lower deadline), and $t_x(i)$ is the earliest time to pack it using any other in sequence packing algorithm, then the following holds true.

$$t_f(i) \le t_x(i).$$

The above proposition states that $t_f(i)$ is the lower deadline of item i, so item i can't be packed earlier without violating decisions of lower components of earlier items. Therefore, for n-th component, we are allowed to increase the item size of item i as far as we can pack it within the period between its lower and upper deadlines. If increasing its size requires us to start packing it before its lower deadline, then we should not consider that component because this will affect the lower component decisions and will cause dropping lower components of some earlier items. Since, our objective prioritizes lower components over higher components, lower deadline must not be violated to in order to pack the earlier items at at-least at the pre-decided size (number of components).

Using Lemmas 1, 2 and Proposition 1, we are ready to show the optimality of the proposed algorithm in solving problem (1)-(7), and this is stated in the following theorem.

Theorem 1: Up to a given component $M, M \ge 0$, if ${x_{i,n,t}}^*$ is the decision of every component $m \le M$ of item i that is found by running backward-forward algorithm, and ${x_{i,n,t}}'$ is a decision that is found by any other feasible algorithm, then the following holds.

$$\sum_{n=0}^{M}\sum_{i=1}^{C}\sum_{t=1}^{C} v^n {x_{i,n,t}}' \le \sum_{n=0}^{M}\sum_{i=1}^{C}\sum_{t=1}^{C} v^n {x_{i,n,t}}^* \tag{8}$$

when:

1) $w_i^n = w^n \forall i$
2) $v_i^n = v^n \forall i$
3) $C\sum_{k=a+1}^{N} v^k < v^a$, *for* $a = 0, \cdots, N-2$

In other words, the proposed algorithm finds the optimal solution to the optimization problem (1)-(7), *when the above 3 conditions are satisfied.*

Proof: A detailed proof is provided in Appendix C (see the Supplementary Material). □

IV. Specialization of the Proposed Formulation Video Streaming

In modern video systems, the video content is divided into chunks (segments), and each is encoded at different quality levels. In conventional video encoding (e.g H.264/MPEG4-AVC, Advanced Video Coding), each video chunk is stored into L *independent* encoding versions. When fetching a chunk, the player's adaptation mechanism, Adaptive Bit Rate (ABR) streaming, must select one out of the L versions based on its estimation of the network bandwidth and the buffer capacity [10]. Another encoding technique is Scalable Video Coding (SVC [11]) in which each chunk is encoded into ordered *layers*: one *base layer* (BL) with the lowest playable quality, and multiple *enhancement layers* $(E_1, \cdots, E_N)$ that further improve the chunk quality. For decoding a chunk up to enhancement layer i, a player must download all layers from 0 to i.

There are two forms of streaming - realtime streaming and non-realtime streaming. In realtime streaming, there is a playback deadline for each of the chunks, and chunks not received by their respective deadlines are skipped. In no-skip based streaming (non-realtime streaming), if a chunk cannot be downloaded by its deadline, it will not be skipped; instead, a stall (re-buffering) will incur, i.e., the video will pause until the chunk is fully downloaded. In both variants, the goal of the scheduling algorithm is to determine up to which layer we need to fetch for each chunk (except for those skipped in realtime streaming), such that the overall quality-of-experience (QoE) is maximized and the number of stalls or skipped chunks is minimized.

In Appendix D (see the Supplementary Material), we show that the constraints match the problem of real-time SVC video streaming well, and the special case matches the intuitions for an efficient quality metric for the end users. Thus, the proposed algorithm can be used to provide efficient algorithms for video streaming, as was done in [7]. The proposed algorithm can also be adapted to give results for no-skip based streaming.

Even though we consider SVC encoding, the same algorithm has also been used for ABR streaming in [8]. The key idea is that the difference between different quality levels in ABR streaming can be used as the quality of different layers. The results in [7] and [8] demonstrate significant performance improvement as compared to the considered state-of-art algorithms.

The application of the proposed formulation to video streaming is further described in Appendix D (see the Supplementary Material). The complete formulation is provided in Appendix F (see the Supplementary Material). Appendix G (see the Supplementary Material) further provides the evaluations of the proposed algorithm for SVC video streaming. For more details on the application to video streaming, the reader is referred to [7] and [8].

V. Conclusions

A deadline and buffer constrained knapsack problem is proposed. The problem is shown to be NP hard in general. However, a polynomial complexity algorithm (linear time in the number of bins and components of each item) is proposed and shown to be optimal in a special case where the values and weights follow certain conditions. This problem is a generalization of the problem formulation considered for video streaming, where the algorithm was shown to provide a low-complexity streaming algorithm that outperforms all considered baselines for both SVC and AVC video streaming. The optimization problem has been further extended to multi-path and cooperative video streaming [15-16]. The proposed generalization of the video streaming problem has the potential to have applications in other areas (e.g., Appendix H (see the Supplementary Material)), and finding such applications is an important future direction. Generalizing the special case to still have polynomial-time solutions is another important research direction.

References

[1] A. Abraham, R. Buyya, and B. Nath, "Nature's heuristics for scheduling jobs on computational grids," in *Proc. 8th IEEE Int. Conf. Adv. Comput. Commun. (ADCOM)*, Dec. 2000, pp. 45–52.

[2] B. Jarboui, N. Damak, P. Siarry, and A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems," *Appl. Math. Comput.*, vol. 195, no. 1, pp. 299–308, 2008.

[3] K. P. Bennett and E. Parrado-Hernández, "The interplay of optimization and machine learning research," *J. Mach. Learn. Res.*, vol. 7, pp. 1265–1281, Jul. 2006.

[4] W. Kuo and V. R. Prasad, "An annotated overview of system-reliability optimization," *IEEE Trans. Rel.*, vol. 49, no. 2, pp. 176–187, Jun. 2000.

[5] L. A. Wolsey and G. L. Nemhauser, *Integer and Combinatorial Optimization*. Hoboken, NJ, USA: Wiley, 2014.

[6] H. Kellerer, U. Pferschy, and D. Pisinger, "Multidimensional knapsack problems," in *Knapsack Problems*. Berlin, Germany: Springer, 2004, pp. 235–283.

[7] A. Elgabli, V. Aggarwal, S. Hao, F. Qian, and S. Sen, "LBP: Robust rate adaptation algorithm for SVC video streaming," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1633–1645, Aug. 2018.

[8] A. Elgabli and V. Aggarwal. (2018). "FastScan: Robust low-complexity rate adaptation algorithm for video streaming over HTTP." [Online]. Available: https://arxiv.org/abs/1806.02803

[9] K. Miller, A.-K. Al-Tamimi, and A. Wolisz, "QoE-based low-delay live streaming using throughput predictions," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 1, p. 4, 2017.

[10] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proc. ACM SIGCOMM*, 2015, pp. 325–338.

[11] R. Jillani and H. Kalva, "Scalable video coding standard," in *Encyclopedia of Multimedia*, B. Furht, Ed. Boston, MA, USA: Springer, 2008, pp. 775–781. doi: 10.1007/978-0-387-78414-4_63.

[12] W. Zhang, Y. Wen, Z. Chen, and A. Khisti, "QoE-driven cache management for HTTP adaptive bit rate streaming over wireless networks," *IEEE Trans. Multimedia*, vol. 15, no. 6, pp. 1431–1445, Oct. 2013.

[13] C. Kreuzberger, D. Posch, and H. Hellwagner, "A scalable video coding dataset and toolchain for dynamic adaptive streaming over HTTP," in *Proc. ACM MMSys*, 2015, pp. 213–218.

[14] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: analysis and applications," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 114–118.

[15] A. Elgabli, M. Felemban, and V. Aggarwal, "GiantClient: Video hotspot for multi-user streaming," *IEEE Trans. Circuits Syst. Video Technol.*, 2018.

[16] A. Elgabli, K. Liu, and W. Aggarwal, "Optimized preference-aware multi-path video streaming with scalable video coding," *IEEE Trans. Mobile Comput.*, 2018.