Collaborate Algorithms for the Multi-channel Program Download Problem in VOD Applications

Wenli Zhang, Lin Yang, Kepi Zhang, and Chao Peng(✉)

School of Computer Science and Software Engineering,
East China Normal University, 3663 Zhongshan North Road, Shanghai 200062, China
{51141500065,51141500051,51141500059}@ecnu.cn, cpeng@sei.ecnu.edu.cn

Abstract. Video-on-demand (VOD) is a multimedia technology that allows users to watch video programs from a server flexibly at any time. In recent years, VOD applications are very popular in many networks, especially in internet of vehicles where video programs can often be collaboratively downloaded from multiple channels simultaneously. In this paper, we first study the Multi-Channel Program Download Problem (McPDP), which is to download a set of interested programs from different channels within limited time. We prove that McPDP is NP-complete by reduction from 3-SAT(3). For another version with neatly placed programs of equal length, the aligned multi-channel program download problem (AMcPDP), we present an algorithm by transforming it into a max-flow problem. Finally, we have also analyzed the performance of these proposed algorithms by simulation using MATLAB.

Keywords: Video-on-demand · Multi-channel Program Download Problem · NP-complete · Scheduling algorithms

1 Introduction

The last decades have witnessed the prevalence of multimedia technology, which has many different content forms, such as images, text, audio, animation and video. With the rapid development of science and technology, it has penetrated into all aspects of the national economy and improved the quality of our life. In recent years, more and more video services have become our research hotspots, especially, the video-on-demand (VOD) technology [1,2]. A VOD system allows users to watch video programs from a sever flexibly at any time, it is usually in a client-server architecture supported by certain transport networks such as cable TV systems (CATV) [3], LAN and community antenna television systems [4]. In a VOD system, a server will broadcast segments of a set of programs periodically in several channels. A group of clients can be served simultaneously, and will download their interested programs from channels if their local storage can accommodate [5–7]. Programs will be downloaded into the local storage of the client, and then the client can watch programs at any time without a break.

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 333–342, 2017.
DOI: 10.1007/978-3-319-59288-6_30

The research objective of VOD, no matter the server side or the client side, is to find the fastest way collaboratively to download and watch interested programs as soon as possible. In recent years, much attention has been paid to broadcasting protocols on the server side. In [8], the authors develop an optimal bandwidth allocation algorithm for hybrid VoD streaming. This paper proposes a novel Demand Driven Max-Flow Formulation, which treats each peers bandwidth demand as the flow commodity. Authors in [5] present a permutation based pyramid scheme in which the storage requirements and disk transfer rates are greatly reduced, and yet the viewer latency is smaller as well. But they usually focus only on the tradeoff between bandwidth and delay, thus they are usually not efficient for the local storage. Authors in [4] proposed some new effective broadcasting protocols, which can intelligently adjust the solution according to available bandwidth and local storage to achieve an ideal waiting time. As for client side, authors in [9] presented a Program Download Problem (PDP) which is to download a set of desired programs from channels. But their assumption is that client can only download programs from one channel at one time. A client has to switch between different channels since his/her interested programs might be broadcasted in them.

However, this assumption may be inconsistent with the actual situation. For example, in vehicular networks [10], the Road Side Unit (RSU) plays the server role. It will keep broadcasting data, such as advertisements, local weather forecast, traffic situation etc. in multiple channels. Clients (video equipment in vehicles) within a range will be connected to the channel via wireless networks. To avoid the unnecessary waste of bandwidth, multiple channels downloading can be implemented by using MIMO technology [11]. But, many other factors such as limited buffer and conflicts in the client side should be considered, so it is necessary for us to find efficient algorithms to schedule the downloading process [12–15].

The remainder of this paper is organized as follows. Section 2 defines the Multi-Channel program download problem(McPDP) and proves McPDP is NP-Complete. In Sect. 3, the Aligned multi-channel program download problem(AMcPDP) is described, and we present an algorithm to solve it. In Sect. 4, we implement all the algorithms with Matlab, and analyze their performance. Finally, Sect. 5 concludes the paper.

2 Complexity of the Multi-channel Program Download Problem

Definition 1. *In an instance of the Multi-Channel program download problem (McPDP), there is a channel set C which contains n channels, said $\{c_1, c_2, \cdots, c_n\}$. The server (such as an RSU) broadcasts a set of programs U in these channels, and each program segment might appear multiple times. Each client (such as a video equipment in a vehicle) within range may have a target set of programs $P(P \subseteq U)$ to be downloaded as soon as possible. We assume that each program segment has a unit time interval length. A client can download*

any program from its starting time, and may switch between different channels at the end of the program. (Switching between different channels may take some time, it is very short and can be ignored here.) A client can download programs from up to $k(k < n)$ channels simultaneously. For any given target program set and a corresponding deadline d, the multi-channel program download problem (McPDP) asks whether we can download all programs in this set before d, it will be denoted as $McPDP(n, k, d)$. Notice that there is no limit on the appearance times of a program in each channel, next we will analyze the complexity of McPDP.

Theorem 1. *Multi-Channel program download problem (McPDP) is NP-complete [16].*

Proof. We will reduce a simple version $McPDP(2, 1, d)$ into $McPDP(n, k, d)(k < n)$ and then prove that $McPDP(2, 1, d)$ is NP-Complete. Both $McPDP(n, k, d)$ and $McPDP(2, 1, d)$ are obviously in NP, as we can check whether a download schedule is a yes certificate or not in polynomial time.

The first reduction process is rather simple. Given an instance of $McPDP(2, 1, d)$, we will add $n - 2$ channels, $k - 1$ reading heads and some programs to keep these $k - 1$ heads busy. To do this, we simply put $(k - 1) \cdot d$ different programs into $k - 1$ channels and we join these programs into the original target set to make the final target set P. Then we put a peg program into the remain $n - k - 1$ channels.

The above transformation process can be finished in polynomial time, and it is not difficult to find out that any solution of the original version is corresponding to exact one solution of the $McPDP(n, k, d)$ version. That is because the later one has to use $k - 1$ heads to download those newly added programs at any time, and only one head can be used to download those old programs inherited from the original target set located in the first 2 channels.

Thus a solution to $McPDP(n, k, d)$ can be easily changed into one of $McPDP(2, 1, d)$ and vice versa.

Next, we will reduce 3-SAT(3), a known NP-complete problem to $McPDP(2, 1, d)$ to prove its NP-completeness. 3-SAT(3) is a special case of 3-SAT where each clause includes 3 items at most, and every literal will appear three times exactly (twice positively and once negatively or vice versa).

Given an instance of 3-SAT(3), we suppose the input formula F contains N variables x_i $(1 \leq i \leq N)$ and M clauses $C_1, C_2, \ldots, C_M$. Then we will build a corresponding new instance of $McPDP(2, 1, d)$.

For each clause C_i, we construct a unit-length program C_i and all these programs will form the target set P. Next we construct two channels c and $\bar{c}$, and evenly divide the two channels into segments with length of 4 time units.

Since each variable x_i will appear exactly three times in F, there will be three corresponding clause programs. W.o.l.g., we assume that the F contains two x_is and one $\overline{x}_i$, x_i appears in clause C^{i1} and clause C^{i2} while $\overline{x}_i$ appears in clause C^{i3}. These three clauses correspond to three unit-time programs C^{i1}, C^{i2} and C^{i3} in the new instance. We will put C^{i1} and C^{i2} in channel c, starting at

time $4 \times (i-1) + 0.5$ and $4 \times (i-1) + 1.5$, followed by a unit-time peg program P_i starting at $4 \times (i-1) + 3$. On the other hand, C^{i3} will be put in channel $\overline{c}$ starting at $4 \times (i-1) + 1$, also followed by a peg program P_i. The case of one x_i and two $\overline{x}_i$s can be similarly handled.

Notice that a client can only download one program at one moment in the new instance of $McPDP(2, 1, d)$. Thus, a client cannot download C^{i1}, C^{i2} and C^{i3} in the meanwhile.

Now we will show that the formula F is satisfiable if and only if all the programs in P can be downloaded before deadline $d = 4n$.

($\rightarrow$) At first, we will show that formula F is satisfiable then all the programs in P can be downloaded by time $d = 4n$. Given a truth assignment of the formula F, for each clause C_i there will be at least one variable x_j in C_i that makes it true. So the client in $McPDP(2, 1, d)$ will correspondingly download clause program in the j th segment of channel c if x_j appears in C_i positively, or in the j th segment of channel $\overline{c}$ if x_j appears in C_i negatively. Since all clauses are true by this assignment, all the required programs will be downloaded before time $4n$.

($\leftarrow$) If all the required programs have been downloaded by time $4n$. Since we cannot download programs from channel c and channel $\overline{c}$ at the same time, we will assign $True$ to x_j if the client has downloaded clause programs C^i in the j-th segment of channel c. On the other hand, $False$ should be assigned to x_j if clause program C^i has been downloaded in the j-th segment of channel $\overline{c}$.

The above reduction takes polynomial time. Hence the theorem is proven. □

Here, we show a simple example of the instance. Given an instance of 3-SAT(3) where the input formula is as follows:

$$F = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee \overline{x}_3 \vee x_5) \wedge (\overline{x}_1 \vee \overline{x}_5 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_2 \vee \overline{x}_4 \vee x_5) \quad (1)$$

The corresponding instance of $McPDP(2, 1, d)(P = \{C_1, C_2, C_3, C_4, C_5\})$ is in Fig. 1.

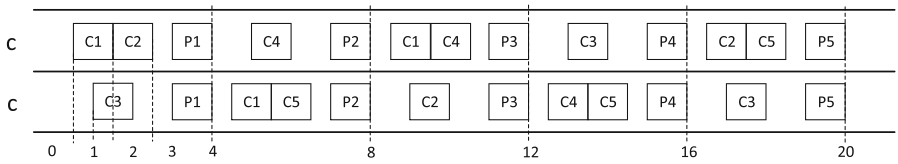


Fig. 1. The corresponding instance of formula F in Eq. 1

3 Aligned Multi-channel Program Download Problem

Definition 2. *The Aligned Multi-channel Program Download Problem (AMcPDP) is based on McPDP, it requires that all programs should be neatly aligned in the channels, thus if two programs overlap, they will start together and finish together.*

Instance. A program set U contains a series of programs which are neatly aligned in a set of channels C, where $|C| = n$. Similarly, each client within a range will have a target set of programs $P(P \subseteq U)$ to download, and we can download k programs at the same time. The Aligned multi-channel program download problem (AMcPDP) can be denoted as $AMcPDP(n, k, d)$, it aims to download all programs in the target set P with the least time by deadline d.

Theorem 2. *Aligned multi-channel program download problem (AMcPDP) is in P.*

Proof. To solve $AMcPDP(n, k, d)$, we can transform it into a maximum flow problem. Firstly, for each program in set P, we create a node $p_i(1 < i < |P|)$. We also need a set of nodes $r_t(1 \leq t \leq d)$ to represent time unit. A source node a and a sink node b are also necessary. We add a directed edge from a to all the p_i nodes. Then we connect node p_i to node r_t if program p_i can be downloaded at time unit t. Finally, we link all the nodes r_t to b. For the edge from r_t to b, we set a capacity of k, because programs can be downloaded from k channels at the same time. The remaining edges are all assigned one unit capacity.

Next we will prove that $AMcPDP(n, k, d)$ can be solved iff we can find a $|P|$-flow from a to b in the graph we have created.

($\rightarrow$) If we have downloaded all the programs in the target set P in the fastest way, then we can send $|P|$ units of flows from source node a to each node p_i, then it will flow to node r_t, and finally node b.

($\leftarrow$) Denote $G(r_t)$ as the subset of the graph whose path contains node $r_1, r_2, \cdots, r_t$, if we can find $|P|$ flow paths in total in $G(r_t)$, each of which is from source node a to node p_i, and r_t, finally to sink node b, correspondingly, we can make a schedule to download program p_i at the time unit of r_t. In this way, we can download all $|P|$ programs in the target set with the least time when we find the least t that contains $|P|$ flows in $G(r_t)$. □

The following pseudo-code shows an algorithm to solve $AMcPDP(n, k, d)$. This algorithm is based on Ford-Fulkerson algorithm. To get the earliest time which contains $|P|$ flows, we shall find the possible earliest time $t_0(t_0 = max\{t, \lceil \frac{|P|}{K} \rceil\}$, where t is the earliest unit time when all the programs in P have appeared at least once and $\lceil \frac{|P|}{K} \rceil$ represents that at each time, the client can download K programs from n channels). If we cannot find $|P|$ flows within time t_0, then we increase the time unit t_0 time $t_0 + 1, t_0 + 2, \cdots, t_0 + i$ until d.

In Algorithm 1, we firstly construct a graph in which the number of nodes is $|V| = 2 + |P| + d$, and the edges are no more than $|E| = |P| + d \times n + d$. Since the graph can be done in $O(d \times n)$ time. The outer loop will execute at most $d - t_0 + 1$ rounds and the major part in each loop is the inner loop. To find an augmenting path in the graph will take $O(d \times n)$ time, therefore the inner loop will take at most $O(|P| \times d \times n)$ time, which is at most $O((d \times n)^2)$. Thus the total complexity of this algorithm is $O(d^3 \times n^2)$.

For example, we give a program broadcasting schedule in Fig. 2(a) as follows, the target set $P = \{S_1, S_2, S_3, S_4\}$ and $k = 2$, then construct the corresponding

Algorithm 1. $AMcPDP(n, k, d)$ algorithm

Input:

A broadcasting schedule $I = \{(p_i, t)\}$ (program p_i broadcasts at time t) of a program set P

Output:

The fastest download schedule W of programs;

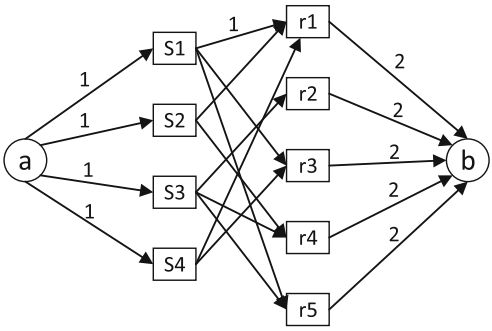
Create node set $V = \{a\} \cup \{p_i\} \cup \{r_t\} \cup \{b\}, 1 \leq i \leq |P|, 1 \leq t \leq d$;
Create edge set $E = \{(a, p_i) \cup (r_t, b)\}$;
Add edge (p_i, r_t) to E iff program p_i can be downloaded at time t;
Set $f(e) \leftarrow 0$ for all the edge $e \in E$;
Set $t \leftarrow t_0 \leftarrow max\{t, \lceil \frac{|P|}{K} \rceil\}$ while r_t means time t;
while $t \leq d$ **do**
  Set $G_f(r_t) \leftarrow G(r_t)$ while $G_f(r_t)$ is the residual graph;
  while $G_f(r_t)$ contains an a-b path **do**
    let m denote the a-b path;
    $f \leftarrow augment(f, m)$;
  end while
  if find a $|P|$ flow **then**
    break;
  end if
  $t \leftarrow t + 1$;
end while
Decompose f into $|P|$ unit-flow paths S;
build the schedule $W = \{(p_i, t)\}$ based on the path set S;
return W

graph as Fig. 2(b). At first, time $t_0 = max\{2, \lceil \frac{|4|}{2} \rceil\} = 2$. Next, we attempt to find the max flow in $G(r_2)$ but failed, then we try to continue searching for such a max flow. In this example, we can find it within time 3.

Based on Algorithm 1, we can try to apply binary search to find a relative optimal solution. Firstly, we find the possible earliest time t_0. Then, we can get the earliest time in the following way: we try to find a maximum flow within time t_0. If such a maximum flow exists, $AMcPDP(n, k, d)$ can be solved with the shortest time t_0. However, if there doesn't exist such a maximum flow, we should find another time t_i $(i = 1, 2, \cdots, d)$ which is the earliest time with a $|P|$ flow and $t_i = 2^i \cdot t_0$. We use binary search to find a $|P|$ flow within the shortest time, and the searching range is from time $t_{i-1} + 1$ to t_i.

Next, we will show the pseudo-code of Algorithm 2, we call it $IAMcPDP$ (n,k,d) algorithm. In this algorithm, we use binary search to approach the earliest time. Denote $G(r_t)$ as the subset of the graph whose path contains node $r_1, \cdots, r_t$, and $G_f(r_t)$ as the residual graph of it, and this algorithm can solve $AMcPDP$ in polynomial time obviously.

S1	S5	S1	S2	S1
S2	S6	S4	S6	S3
S4	S3	S5	S3	S5



(a) Broadcasting schedule of programs (b) the corresponding maximum flow graph

Fig. 2. An AMcPDP example

Algorithm 2. $IAMcPDP(n, k, d)$ algorithm

Input:

A broadcasting schedule $I = \{(p_i, t)\}$ (program p_i broadcasts at time t) of a program set P

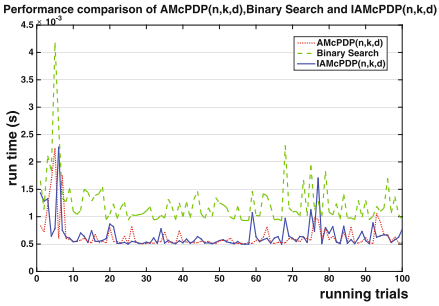
Output:

The fastest download schedule of programs;

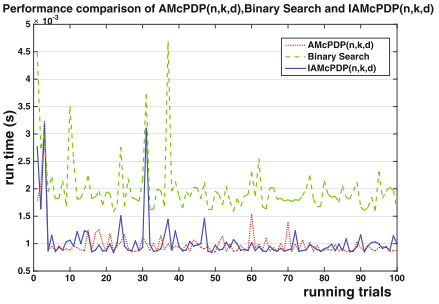
Create node set $V = \{a\} \cup \{p_i\} \cup \{r_t\} \cup \{b\}, 1 \leq i \leq |P|, 1 \leq t \leq d$;
Create edge set $E = \{(a, p_i) \cup (r_t, b)\}$;
Add edge (p_i, r_t) to E iff program p_i can be downloaded at time t;
Set $f(e) \leftarrow 0$ for all the edge $e \in E$;
Set $t \leftarrow t_0 \leftarrow max\{t, \lceil \frac{|P|}{K} \rceil\}$ while r_t means time t;
while $t \leq d$ **do**
  Set $G_f(r_t) \leftarrow G(r_t)$ while $G_f(r_t)$ is the residual graph;
  if find $|P|$ flows in $G_f(r_t)$ **then**
    if $t! = t_0$ **then**
      use binary search to find the earliest time t which contains $|P|$ flows between the time $t_0 + 1$ and t
    end if
    Decompose f into $|P|$ unit-flow paths S;
    build the schedule $W = \{(p_i, t)\}$ based on the path set S;
    return W
  else
    $t = 2t$
  end if
end while

4 Performance Evaluation

In this section, we use Matlab to evaluate the performance of these proposed algorithms above.



(a) Performance comparison of $AMcPDP(5, 2, 20)$, $IAMcPDP(5, 2, 20)$ and $Binary\ Search(5, 2, 20)$



(b) Performance comparison of $AMcPDP(5, 2, 40)$, $IAMcPDP(5, 2, 40)$ and $Binary\ Search(5, 2, 40)$

Fig. 3. Performance comparison of three algorithms where the program set U is $\{S_1, S_2, S_3, \cdots, S_{10}\}$ and the target set P is $\{S_1, S_2, S_3, \cdots, S_9\}$

4.1 AMcPDP(n,k,d), IAMcPDP(n,k,d) and Binary Search

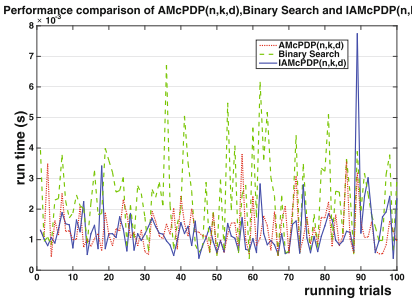
In our experiment, for better comparison, besides $AMcPDP\ (n, k, d)$ and $IAMcPDP\ (n, k, d)$, we have also implemented the binary search algorithm, which finds maximum flow from the last time unit, and then use binary search to find the final target.

Firstly, we generate a $d \times n$ matrix to simulate our experiment environment: n channels and d time units. Each element in the matrix is a representation of a program. Then we run these three algorithms respectively, and compare the performance of them. In order to get a trustable conclusion, we have tried multiple parameters. The following figures show the run time of these three algorithms with different parameters over 100 trials.

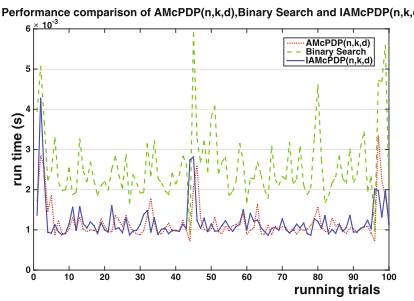
In figures above, the green line represents the running time of Binary Search algorithm, while the blue line and the red line represent the running time of $IAMcPDP(n, k, d)$ and $AMcPDP(n, k, d)$ respectively. From these figures, we find that the binary search algorithm is inferior to the other two algorithms in all examples, and the running time of $AMcPDP(n, k, d)$ and $IAMcPDP(n, k, d)$ are nearly close to each other since our target programs are very easy to be found near time unit $t_0 (t_0 = max\{t, \lceil \frac{|P|}{K} \rceil\})$, which is the reason why $IAMcPDP(n, k, d)$ doesn't have obvious advantages over $AMcPDP(n, k, d)$.

Obviously, there are some sharp fluctuations in the figures, which are caused by the complexity of the examples. The matrixes are generated randomly, so it inevitably will produce different examples. A hard example may lead to a peak value of a curve, while an easy example may cause a downward trend, and the fluctuating trend of two curves in the same figure almost keeps the same.

From Fig. 3(a) and (b), we see that curves in Fig. 3(b) is higher than that in Fig. 3(a). We conclude that the setting of deadline d will make a difference to the performance of the algorithms. The reasons are as follows: The first step of these algorithms is to check whether there exists a $|P|$ flow before deadline d, and a



(a) Performance comparison of $AMcPDP(5,2,20)$, $IAMcPDP(5,2,20)$ and $Binary\ Search(5,2,20)$



(b) Performance comparison of $AMcPDP(5,2,40)$, $IAMcPDP(5,2,40)$ and $Binary\ Search(5,2,40)$

Fig. 4. Performance comparison of three algorithms where the program set U is $\{S_1, S_2, S_3, \cdots, S_{30}\}$ and the target set P is $\{S_1, S_2, S_3, \cdots, S_9\}$

later deadline will make the maximum flow algorithms more complex. Another possibility is that there doesn't exist such a $|P|$ flow before the early deadline, so the algorithms will stop executing under this situation. On the other hand, the latter deadline is later enough to find a $|P|$ flow, the following algorithms will continue running.

The only different parameters used in Fig. 3(a) and Fig. 4(a) is the program set U, we can see that the overall running time in Fig. 4(a) is slower than that in Fig. 3(a). This is an obvious difference as under the same conditions, the hitting ratio of selecting 9 specific programs from 10 ones is undoubtedly higher than that from 30 ones. Two maximum flow algorithms in Fig. 4(a) and (b) don't have a clear superiority over that in Fig. 3(a) and (b) as binary search algorithm will show its advantage when the hitting ratio of target programs isn't very high.

5 Conclusion

In this paper, we proved that the Multi-Channel program download problem (McPDP) is NP-complete by reducing 3-SAT(3) to it. We then find aligned multi-channel program download problem (AMcPDP) can be solved in polynomial time by reducing it to a max-flow problem, and we present two algorithms to solve it. Finally, we implement these algorithms in Matlab and the simulation results corroborate their efficiency.

References

1. Hanczewski, S., Stasiak, M.: Modeling of video on demand systems. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2014. CCIS, vol. 431, pp. 233–242. Springer, Cham (2014). doi:10.1007/978-3-319-07941-7_24

2. Miesler, L., Gehring, B., Hannich, F., Wüthrich, A.: User experience of video-on-demand applications for smart TVs: a case study. In: Marcus, A. (ed.) DUXU 2014. LNCS, vol. 8520, pp. 412–422. Springer, Cham (2014). doi:10.1007/978-3-319-07638-6_40
3. Atzori, L., De Natale, F.G.B., Di Gregorio, M., Giusto, D.D.: Multimedia information broadcasting using digital TV channels. IEEE Trans. Broadcast. **43**(3), 242–251 (1997)
4. Peng, C., Tan, Y., Xiong, N., Yang, L.T., Park, J.H., Kim, S.-S.: Adaptive video-on-demand broadcasting in ubiquitous computing environment. Pers. Ubiquit. Comput. **13**(7), 479–488 (2009)
5. Aggarwal, C.C., Wolf, J.L., Yu, P.S.: A permutation-based pyramid broadcasting scheme for video-on-demand systems. In: Proceedings of the International Conference on Multimedia Computing and Systems, pp. 118–126 (1996)
6. Almeroth, K.C., Ammar, M.H.: The use of multicast delivery to provide a scalable and interactive video-on-demand service. IEEE J. Sel. Areas Commun. **14**(5), 1110–1122 (1996)
7. Juhn, L., Tseng, L.: Harmonic broadcasting for video-on-demand service. IEEE Trans. Broadcast. **43**(3), 268–271 (1997)
8. Tian, C., Sun, J., Wu, W., Luo, Y.: Optimal bandwidth allocation for hybrid video-on-demand streaming with a distributed max flow algorithm. Comput. Netw. **91**, 483–494 (2015)
9. Peng, C., Zhou, J., Zhu, B., Zhu, H.: Complexity analysis and algorithms for the program download problem. J. Comb. Optim. **29**(1), 216–227 (2015)
10. Xie, H., Boukerche, A., Loureiro, A.A.F.: MERVS: a novel multichannel error recovery video streaming protocol for vehicle ad hoc networks. IEEE Transaction on Vehicular Technology **65**(2), 923–935 (2016)
11. Yang, T., Liang, H., Cheng, N., Deng, R., Shen, X.: Efficient scheduling for video transmissions in maritime wireless communication networks. IEEE Transaction on Vehicular Technology **64**(9), 4215–4229 (2015)
12. Zaixin, L., Shi, Y., Weili, W., Bin, F.: Efficient data retrieval scheduling for multi-channel wireless data broadcast. In: Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM), pp. 891–899 (2012)
13. Lu, Z., Wu, W., Fu, B.: Optimal data retrieval scheduling in the multi-channel data broadcast environments. IEEE Trans. Comput. **62**(12), 2427–2439 (2013)
14. Paris, J.-F., Carter, S., Long, D.D.E.: Efficient broadcasting protocols for video on demand. In: MASCOTS, pp. 127–132 (1998)
15. Aggarwal, V., Robert Calderbank, A., Gopalakrishnan, V., Jana, R., Ramakrishnan, K.K., Yu, F.: The effectiveness of intelligent scheduling for multicast video-on-demand. In: ACM Multimedia, pp. 421–430 (2009)
16. Darmann, A., Dcker, J.: Monotone 3-Sat-4 is NP-complete. CoRR abs/1603.07881 (2016)