Time Dependent Multi Scheduling of Multicast

Rami Cohen† Dror Rawitz‡ Danny Raz†

August 23, 2008

Abstract

Many network applications that need to distribute content and data to a large number of clients use a hybrid scheme in which one (or more) multicast channel is used in parallel to a unicast dissemination. This way the application can distribute data using one of its available multicast channels or by sending one or more unicast transmissions. In such a model the utilization of the multicast channels is critical for the overall performance of the system.

We study the scheduling algorithm of the sender in such a model. We describe this scheduling problem as an optimization problem where the objective is to maximize the utilization of the multicast channel. Our model captures the fact that it may be beneficial to multicast an object more than once (e.g., page update). Thus, the benefit depends, among other things, on the last time the object was sent, which makes the problem much more complex than previous related scheduling problems. We show that our problem is NP-hard. Then, using the local ratio technique we obtain a 4-approximation algorithm for the case where the objects are of fixed size and a 10-approximation algorithm for the general case. We also consider a special case, which may be of practical interest, and prove that a simple greedy algorithm is a 3-approximation algorithm in this case.

Keywords: Approximation Algorithms, Multicast, Local Ratio, Scheduling.
1 Introduction

Web caching is a common way to overcome problems such as congestion and delay that often arise in the current Internet. Using this mechanism web caches are deployed between web clients and web servers and store a subset of the server content. Upon receiving a request from a client (or from another cache) a web cache tries to satisfy this request using a local (valid) copy of the required object. If the cache does not hold such a copy, it tries to retrieve the object from other caches (e.g., using Internet Cache Protocol [16]), or from the original server (e.g., using HTTP [8]).

Web caching can be useful in several aspects. From the client’s point of view, the presence of caches reduces the average response time. Internet Service Providers (ISP’s) deploy web caches in order to reduce the load over their links to the rest of the Internet. Web servers benefit from web cache deployment as well since it reduces the amount of requests that should be handled by them. These advantages led to the creation of the Content Distribution Network (CDN). CDN’s distribute content on behalf of origins web servers in order to offload work from these servers. Although CDN’s differ in their implementation model (e.g., distribution and redirection techniques) [12], they all take advantage of web caching in order to bring the content close to the clients.

A common and widespread technique that is used by CDN’s to distribute their content is to push objects from web servers into web caches using a multicast-based channel. Using a broadcast channel such as satellite link, a CDN can push an object into many caches in one transmission instead of many unicast (terrestrial) transmissions. In this kind of a model web caches receive requests from clients (or other caches). If a request cannot be satisfied locally (i.e., there is no valid copy of the required object) the cache sends the request to the origin server. Then, the server either sends the object to the cache using unicast transmission or it can distribute the object to all caches using its multicast channel (see Figure 1).

![Figure 1: Content Distribution Network](image)

The above mentioned CDN model is an example of a network application that uses both a multicast channel and unicast dissemination. This hybrid scheme is used by many other network applications that distribute content and data to a large number of clients. The multicast channel is a limited resource and the utilization of the channel is an important issue. Determining which data should be sent by multicast and which data should be sent by unicast is a big challenge.
Other examples of network applications that may use this kind of hybrid scheme are reliable multicast protocols, real-time streaming protocols, and information notification services. In a reliable multicast protocol (see, e.g., [13, 9]) the data must be received by all clients. When a client does not receive a packet, it sends a feedback packet to the source that must retransmit this packet. The retransmissions of lost packets are critical since they may prevent clients from receiving other packets (according to the client receive window) and may cause timeout events. A real-time streaming protocol (see, e.g., [14]) is used to send multimedia streams to a set of clients. In this application, retransmission of lost packets is not mandatory and clients can use fractional data. However, retransmissions can be done to improve quality but these retransmissions are subject to time constraints since delayed data is useless. In both applications the multicast channel is used by the source to distribute the original transmissions while lost packets may be retransmitted using the multicast channel or the unicast dissemination.

In this paper we study the problem of utilizing a multicast channel in this hybrid scheme. In particular we concentrate on the scheduling algorithm that determines which data is sent via multicast channel at any given time. The problem of multicast scheduling has been discussed before in many papers. In [2, 10] the authors present the problem of scheduling objects over a broadcast based channel. In both [10] and [2] the broadcast channel is the only available channel and the objective function is to minimize the average response time (i.e., the period between the arrival of a request to the time in which this request is satisfied). In this kind of a model the scope of the cost function is very limited since it is proportional to the number of the requesters and the waiting time. Both papers focus on the on-line problem and present on-line scheduling algorithms for their problem. Acharya and Muthukrishnan [1] extend these results by presenting a preemption mechanism that allows to interrupt a transmission and to serve other requests before resuming an interrupted transmission. Moreover, additional objective functions are analyzed. Nevertheless, these cost functions still depend on the service time and they are proportional to the number of requesters and the waiting time. Su and Tassiulas [15] analyze the off-line problem of broadcast scheduling and formulate it as an optimization problem using a similar objective function (i.e., minimize the average response time).

The off-line problem of multicast scheduling in a hybrid scheme is defined in [6], where the authors describe this scheduling problem as an optimization problem in which the goal is to maximize the utilization of the multicast channel. In order to formulate the problem they have defined a benefit function that determines the amount of benefit one can get when a specific object is sent over the multicast channel at a specific time. This benefit function differs from one application to another and it depends on the objective function (e.g., minimizing the delay, minimizing the total transmissions, reducing the utilization of the unicast dissemination) and on other parameters such as time constraints attached to the specific model. They have shown an algorithm that solves the problem in polynomial time for the case where the objects is of fixed size. In addition, a 2-approximation algorithm was presented for the general case.

Nevertheless, in the model used in [6], an object can be sent by multicast only once. Such a restricted model cannot be used to accurately describe the scheme we presented. In web page distribution pages may have an expiration date, and a page may be updated after this time. In reliable multicast and streaming a packet can be lost again, and thus there may be a positive benefit from broadcasting it again. In this paper we present an extended model for the same hybrid scheme, in which the benefit function is more general. In particular, the benefit function depends on previous transmissions of the same object, as it can be sent multiple times. This scheduling problem is much
more complex since the benefit at a specific time depends on the previous transmission of an object.

Overview. The remainder of the paper is organized as follows. The problem is defined and shown to be NP-hard in Section 2. We consider the case where objects have unit size in Section 3. We present a non-trivial transformation that maps this problem to a special case of maximum independent set. Using this mapping we obtain a 4-approximation algorithm. The algorithm is based on the local ratio technique [4, 3]. In Section 4 we examine the more general case where the objects are of variable size. We show that the above mentioned algorithm can be modified to work on objects with variable sizes. The approximation ratio of this algorithm is 10. In Section 5 we discuss the case where \( k \) multicast channels are available and show that our algorithms can be extended to this setting. In Section 6 we study a simple greedy algorithm. It is shown that the approximation ratio of this algorithm is unbounded. However, we show that if the benefit function follows certain restrictions, which are of practical interest, the approximation ratio of the greedy algorithm is bounded by 3. In Section 7 we consider an online version of the problem and show that any deterministic online algorithm for this problem is not competitive.

2 The Problem

In the offline problem the sequence of requests is known in advance, namely the source determines its scheduling according to full information. This kind of a problem takes place in applications such as Information Notification Services where the clients specify their requirement in advance. In other applications such as CDN or reliable multicast protocols the sequence of requests is obtained during the execution of the scheduling. In order to employ an off-line scheduling algorithm in such models one should predict the sequence of requests. For instance, the sequence can be predicted using statistical information gathered during previous periods.

First, we restrict the discussion to the case where all objects have the same size. We divide the time axis into discrete time slots, such that it takes one time slot to broadcast one object. In this case a scheduling algorithm determines which object should be sent via multicast in each time slot. Later on in Section 4 we discuss the general case, where different objects may have different sizes.

2.1 The Benefit Function

The benefit function describes the benefit of sending a specific object via the multicast channel at a specific time. This benefit is the difference between the benefit one can get with and without the multicast channel and it depends on the objective function of each application. The benefit of sending an object at a specific time depends on the requisiteness of the object before and after that time. Therefore, the exact computation of the benefit function requires knowledge regarding the full sequence of requests, and it can be computed and used only in the off-line problem. In addition, the benefit that may be gained by transmitting an object via multicast depends on the last time at which the object was sent, since this previous transmission updates all clients and affects their need for a new copy. According to these observations we define the following:

**Definition 2.1 (Benefit Function)** The benefit function \( B : \mathbb{N}^3 \rightarrow \mathbb{R} \) is defined as follows. \( B[t', t, j] \) is the benefit of broadcasting object \( p_j \) at time slot \( t \), where \( t' \) is the last time slot (prior to \( t \)) in which \( p_j \) was broadcasted. In addition, \( B[t, 0, j] \) is the benefit of broadcasting \( p_j \) at time slot \( t \), where \( t \) is the first time slot where \( p_j \) was broadcasted. Otherwise, for \( t' \geq t \), \( B[t', t, j] = 0 \).
It is very convenient to represent the benefit function by $m$ matrices of size $(T + 1) \times (T + 1)$ where $m$ is the number of objects and $T$ is the number of time slots in the period. In this kind of representation, the benefit function of each object is represented by one matrix, and the elements in the matrix describe the benefit function of the object in the following way: $B[t', t, j]$ is the element in row $t'$ and column $t$ of matrix $j$. Note that according to Definition 2.1, $t' < t$, and therefore the benefit matrices are upper diagonal. For instance, Figure 2 depicts a benefit function of two objects for a period of 3 time slots. Assuming that the multicast channel is always available, one cannot lose by sending an object via multicast, hence the benefit function is non-negative (i.e., $B[t', t, j] \geq 0$ for every $0 \leq t' < t \leq T$ and $j$).

According to Definition 2.1, $B[t', t, j]$ stands for the benefit gained by broadcasting $p_j$ at time slot $t$, given that the previous multicast transmission of $p_j$ was done at time slot $t'$. Observe that the copy of $p_j$ that was sent at $t'$ may still be valid at time slot $t$. Hence, $B[t', t, j]$ should reflect this fact; the benefit gained by sending $p_j$ at time slot $t$ should only account for time slots, at which the copy of $p_j$ that was sent at time slot $t'$ is invalid, while the copy that was sent at time slot $t$ is valid. Furthermore, it may be the case that a client prefers a new copy of an object $p_j$, but will settle for an old copy of $p_j$. In this case the value of $B[t', t, j]$ should increase with the difference between $t$ and $t'$. Another possibility is that a client requires a copy of an object $p_j$ at time $t$ without any regards to its age. In this case, $B[0, t, j] > 0$ while $B[t', t, j] = 0$ for every $t' > 0$. Note that when dealing with statistical information about a sequence of requests, $B[t', t, j]$ may represent the average benefit gained by broadcasting $p_j$ at time $t$, given that the previous multicast transmission of $p_j$ was done at $t'$.

For example, consider a CDN application that uses a satellite based multicast channel. The CDN server distributes objects that have an expiration date or a max-age character and its objective function is to reduce usage of unicast dissemination, namely to reduce the amount of data that is sent via unicast. An expiration date of an object refers to a time at which the object should no longer be returned by a cache unless the cache gets a new (validated) version of this object from the server with a new expiration date. A max-age time of an object refers to the maximum time, since the last time this object has been acquired by a cache, at which the object should no longer be returned by the cache unless it gets a new (validated) version of this object from the server with a new max-age time (see [8, Section 13.2]). Given a set of objects and their sequence of requests for a period of $T$ time slots, the benefit function can be easily built as follows: for each object $p_j$ and for each time slot $t$, we calculate the benefit of broadcasting $p_j$ at time slot $t$, assuming that $t'$ was the last time slot before $t$, at which the object was broadcasted. This can be done (assuming that the sequence of requests is known) by simulating the case where the object is not sent via multicast vs. the case in which the object is sent via multicast. For instance, consider an object $p_j$ with max-age of five time slots and assume that this object is required by cache number one at
time slots 2, 6, and 8. We calculate $B[1,5,j]$ as follows. If $p_j$ is broadcasted at time slots 1 and 5, then there is no need to send it via unicast at all. On the other hand, if $p_j$ is not broadcasted at time slot 5 (i.e., only at time slot 1), then it must be sent via unicast at time slot 6, namely $B[1,5,j]$ is equal to one unicast transmission of $p_j$.

2.2 Problem Definition

The time dependent multi scheduling of multicast problem (TDMSM) is defined as follow: given a benefit function, find a feasible schedule with maximum benefit. An natural way to represent a schedule is by a function $S$ where $S(t)$ stands for the index of the object $j$ that was sent at time slot $t$. ($S$ may be undefined on part of the time slots.) The benefit of a schedule is the sum of the benefit values that are derived from it, namely if $t$ and $t'$ are two consecutive transmission time slots of object $p_j$, then $B[t',t,j]$ is added to the total benefit. For instance, assume that object $p_1$ is sent at time slot 2 and object $p_2$ is sent at time slots 1 and 3. This schedule can be represented by: $S(1) = 2, S(2) = 1$, and $S(3) = 2$, and the total benefit of this schedule is $B[0,2,1] + B[0,1,2] + B[1,3,2]$.

Another way to represent a schedule is by a set of triples, where each triple $(t',t,j)$ contains the index of the object $j$, the time it was sent $t$, and the last time before $t$ at which the object was sent $t'$ (or 0 if the object was not sent before). In this kind of representation, the triple $(t',t,j)$ is in the set if and only if $t$ and $t'$ are two consecutive transmissions of object $p_j$. Henceforth we assume that $t > t'$ in any given triple $(t',t,j)$. For instance, the schedule described above can be represented by the following set of triples: $\{(0,2,1),(0,1,2),(1,3,2)\}$. Although the notation of triples seems to be unnatural and more complex, it has the advantage that the benefit values of are derived directly from the triples. In other words, each triple contains the time slots of two consecutive transmissions, therefore the triple $(t',t,j)$ is in the set if and only if the benefit value $B[t',t,j]$ should be added to the total benefit (in the former representation, using a set of pairs, one should find a consecutive transmissions by parsing the whole set). Nevertheless, not every set of triples represent a feasible schedule. For instance if the triple $(4,6,3)$ is in the set it indicates that object $p_3$ was sent at time slots 4 and 6. Since other objects cannot be sent at these time slots, the triple $(6,7,9)$ cannot be in the set (i.e., object $p_9$ cannot be sent at time slot 6). Moreover the triple $(4,6,3)$ indicates that time slots 4 and 6 are to consecutive transmissions of object $p_3$, namely it was not sent at time slot 5, therefore the triple $(5,8,3)$ cannot be in the set. On the other hand it is clear that every schedule can be represented by a set of triples (every two consecutive transmissions of an object are a triple in the set). In Section 3 we use the notation of triples and the correlation between triples and benefit values to represent the scheduling problem as an independent set problem and to present efficient scheduling algorithm that are based on this representation.

2.3 NP-hardness

We prove that TDMSM is NP-hard using a reduction from the satisfiability problem [7]. Let $\varphi$ be a CNF formula such that $\varphi = C_1 \land \ldots \land C_k$, where $C_j = l_{j1} \lor \ldots \lor l_{jn_j}$, for $1 \leq j \leq k$, and $l_{jp} \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$, for $1 \leq p \leq n_j$. We construct a TDMSM instance that consists of $2n$ objects and a period of $k+n$ time slots. For every $i \in \{1, \ldots, n\}$ we define two objects, one for the literal $x_i$ and one for the literal $\bar{x}_i$. Each object is described by a $(k+n+1) \times (k+n+1)$ benefit matrix. The first $j-1$ elements in the $j$th column of $x_i$’s matrix are 1 if $C_j$ contains $x_i$, otherwise they are 0. Similarly, the first $j-1$ elements in the $j$th column of $\bar{x}_i$’s matrix are 1 if $C_j$ contains
\( \bar{x}_i \), and otherwise they are 0. Also, the value of the first element in column \( k + i \) of matrices \( x_i \) and \( \bar{x}_i \) is set to 1, that is \( B[0, k + i, x_i] = B[0, k + i, \bar{x}_i] = 1 \). All other values are set to 0. The benefit matrices corresponding to the formula \((x_1 \lor \bar{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor \bar{x}_4)\) are given in Figure 3. Note that in this case \( k = 2 \) and \( n = 4 \).

Clearly, the TDMSM instance can be constructed in polynomial time. It remains to show that \( \varphi \) is satisfiable if and only if the maximum benefit of the corresponding TDMSM instance is \( n + k \).

First, the values in the benefit matrices are binary and in each time slot only one object can be broadcasted. Hence, each time slot can contribute at most one to the total benefit, so the total benefit is not greater than \( n + k \).

If there exist a truth assignment \( \tau \) that satisfies \( \varphi \) we use the following schedule. If \( \tau(x_i) = \text{TRUE} \) we broadcast the object that corresponds to \( \bar{x}_i \) only at time slot \( k + i \). If \( \tau(x_i) = \text{FALSE} \) (or \( \tau(\bar{x}_i) = \text{TRUE} \)) we broadcast the object that corresponds to \( x_i \) only at time slot \( k + i \). For each time slot \( i \leq k \) pick one of the literals in \( C_i \) whose assignment is \( \text{TRUE} \), and broadcast the object corresponding to this literal. It is not hard to verify that the total benefit is \( n + k \).

If the total benefit is \( n + k \) then every time slot contributes a benefit of one. Hence for every \( 1 \leq i \leq n \) either \( x_i \)’s object or \( \bar{x}_i \)’s object are set via multicast for the first time at time slot \( k + i \). The schedule at time slot \( k + i \) determines the assignment of \( x_i \) that we denote by \( \tau(x_i) \). If \( x_i \)’s object is broadcasted at time slot \( k + i \) then \( \tau(x_i) = \text{FALSE} \). On the other hand, if \( \bar{x}_i \)’s object is broadcasted at time slot \( k + i \) then \( \tau(x_i) = \text{TRUE} \). For every \( j \leq k \) column \( j \) contributes a benefit of one as well. Assume that \( l_i \in \{x_i, \bar{x}_i\} \) is broadcasted at time slot \( j \). \( \tau(l_i) = \text{TRUE} \) since it is broadcasted before time slot \( k + i \). Moreover, the \( j \)th clause contains \( l_i \), and therefore the clause is satisfied. Since all the clauses are satisfied, \( \varphi \) is satisfied.

We note that we use the benefit values of 0 and 1 in the reduction, namely the problem is hard even in the special case where the benefit function is binary. Observe that a similar reduction can be obtained using the values 1 and 2.

## 3. Unit Size Objects

In Section 2.2 we showed that a feasible schedule can be represented by a set of triples. In this section we use this notation and the relations between triples to devise a 4-approximation algorithm for TDMSM. This algorithm uses the local-ratio technique [4, 3] and is based on the representation of TDMSM as special case of the maximum weight independent set problem.

### 3.1 Relations Between Triples

It is very convenient to represent a triple \((t', t, j)\) as an interval, where the end-points of the interval are \( t' \) and \( t \). Using this kind of representation we can describes the relations between triples according to the location of the corresponding intervals. We say that a triple \((t', t, j)\) intersects a triple \((s', s, j)\) if the intervals \([t', t]\) and \([s', s]\) intersect. In other words, the triples intersect if \( s' < t \) and \( s > t' \) (see Figure 4a). The triples \((t', t, j)\) and \((s', s, k)\) overlap, if \( j \neq k \) and the intervals \((t', t)\) and \((s', s)\) have at least one common end point that is not zero. That is, \((t', t, j)\) and \((s', s, k)\) overlap, if \( j \neq k \) and \( t = s \) or \( t' = s \) or \( t = s' \) or \( t' = s' \neq 0 \). (See example in Figure 4b.) A set of triples is called feasible if it does not contain overlapping triples or intersecting triples.

**Lemma 3.1** A set of triples that describes a feasible schedule is feasible.
Figure 3: The benefit matrices corresponding to the formula $(x_1 \lor \bar{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor \bar{x}_4)$. 

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) Object 1 ($x_1$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Object 2 ($\bar{x}_1$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(c) Object 3 ($x_2$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(d) Object 4 ($\bar{x}_2$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(e) Object 5 ($x_3$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(f) Object 6 ($\bar{x}_3$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(g) Object 7 ($x_4$)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(h) Object 8 ($\bar{x}_4$)
Proof. Consider a feasible schedule in which the \( j \)th object, \( p_j \), is sent via multicast at time slots \( t_{j_1}, \ldots, t_{j_n} \), and let \( S \) be the set of triples that describes this feasible schedule. The scheduling of object \( p_j \), for every \( j \), is represented by the set of triples \( \{(0, t_{j_1}, j), (t_{j_1}, t_{j_2}, j), \ldots, (t_{j_{n-1}}, t_{j_n}, j)\} \) which does not contain intersecting triples. Also, in a feasible schedule at most one object can be sent during every time slot, namely object \( p_i \), where \( i \neq j \), cannot be sent at time slots \( t_{j_1}, \ldots, t_{j_n} \). Therefore, \( S \) does not contain overlapping triples.

Note that a feasible set of triples does not always induce a feasible schedule. For example, the set \( \{(0, 1, 1), (2, 3, 1)\} \) is a feasible set of triples, since it does not contain intersecting or overlapping triples. However, it does not induce a feasible schedule, since the interval between two consecutive transmissions of Object 1 is not represented by a triple. Clearly, we can fix the problem by adding the triple \((1, 2, 1)\) to the set. This leads us to the following definition.

Definition 3.1 A feasible set of triples \( S \) is called maximal (with respect to set inclusion) if the set \( S' = \{(t', t, j)\} \cup S \) is not feasible for every \((t', t, j) \notin S\). In other words, \( S \) is maximal if one cannot add a triple to the set without losing feasibility.

Next, we show that if a feasible set is also maximal, then it represents a feasible schedule.

Lemma 3.2 A feasible maximal set of triples \( S \) induces a feasible schedule.

Proof. Since \( S \) does not contain overlapping triples, at most one object is sent at every time slot. Since \( S \) does not contain intersecting triples, if \((t', t, j) \in S\), then \( t \) and \( t' \) are two consecutive transmissions of \( p_j \). The maximality requirement ensures that if \( t \) and \( t' \) are two consecutive broadcastings of \( p_j \) then \((t', t, j) \in S\), since otherwise \((t', t, j) \) can be added to the \( S \).

3.2 Approximation Algorithm

Given a benefit function, we can construct a weighted graph \( G = (V, E) \), in which every triple \((t, t', j)\) is represented by a vertex \( v[t', t, j] \) whose weight is \( w(v[t', t, j]) = B[t', t, j] \). An edge \((v[t', t, j], v[s', s, k]) \in E \) if \((t', t, j) \) intersects or overlaps \((s', s, k) \). Henceforth we say that \( v[t', t, j] \) and \( v[s', s, k] \) intersect (overlap) if the triples \((t', t, j) \) and \((s', s, k) \) intersect (overlap). By the construction of \( G \), finding a maximum (and maximal) independent set is equivalent to finding a solution to TDMSM.

In general, maximum independent set is hard to approximate [11]. Nevertheless, the graph that is derived from a TDMSM instance is a special case, in which the edges of the graph are characterized by the intersecting and overlapping notation. We describe a recursive algorithm that
finds an independent set in \( G \), and prove that the algorithm computes 4-approximations. We note that unlike other scheduling problems that were approximated using this kind of technique (see, e.g., [3]), the output of our algorithm does not necessarily induce a feasible schedule. Namely, our algorithm finds an independent set and not necessarily a maximal independent set. To obtain a feasible schedule one should extend the output of the algorithm to a maximal independent set. Since the benefit function is non-negative, the benefit of the extended solution in not less than the benefit of the original independent set. Thus, the corresponding schedule is 4-approximate.

Algorithm \( IS \) is a local ratio algorithm that computes a 4-approximate solutions. It is recursive and works as follows. If there are no vertices, then it returns \( \emptyset \). Otherwise, it chooses a vertex \( v[t', t, j] \) with the smallest end point, namely such that \( t \leq s \) for every \( v[s', s, k] \in V \). It constructs a new weight function \( w_1 \), and solves the problem recursively on \( w_2 = w - w_1 \) and the set of vertices with positive \( w_2 \)-weight that is denoted by \( V^+ \). Note that \( v[t', t', j] \notin J^+ \), since \( w_2(v[t', t', j]) = 0 \). Then, it adds \( v[t', t', j] \) to the solution that was computed recursively, if feasibility is maintained.

**Algorithm 1 : \( IS(G = (V, E), w) \)**

1: If \( V = \emptyset \) return \( \emptyset \).
2: Choose \( v[t', t, j] \in V \), such that \( t \leq s \) for every \( v[s', s, k] \in V \)
3: Define the weight functions

\[
  w_1(v[s', s, k]) = \begin{cases} 
    w(v[t', t, j]) & v[s', s, k] = v[t', t, j] \text{ or } (v[t', t, j], v[s', s, k]) \in E, \\
    0 & \text{otherwise}
  \end{cases}
\]

and \( w_2 = w - w_1 \)
4: Define \( V^+ = \{v[s', s, k] \mid w_2(v[s', s, k]) > 0\} \) and \( E^+ = E \cap (V^+ \times V^+) \)
5: \( S^+ \leftarrow IS(G^+ = (V^+, E^+), w_2) \)
6: If \( S^+ \cup \{v[t', t, j]\} \) is an independent set \( S \leftarrow S^+ \cup \{v[t', t, j]\} \)
7: Else \( S \leftarrow S^+ \)
8: return \( S \)

The number of vertices in the graph depends on the number of objects and the number of time slots, namely \(|V| = \frac{1}{2}mT(T - 1)\). In Line 2 the algorithm picks a vertex \( v[t', t, j] \) such that \( t \) is minimal. If more than one vertex can be selected, it is efficient to pick the vertex with maximum weight. By doing so we make sure that there are \( O(T) \) recursive calls, since the minimal end point \( t \) increases with every recursive call. In a naive implementation every iteration requires \( O(mT^2) \) operations (finding the maximum weight, subtracting weights, and removing vertices) thus the complexity of the algorithm is \( O(mT^3) \).

Next, we prove that Algorithm \( IS \) is a 4-approximation algorithm using the Local Ratio Theorem for maximization problems (see [3] for more details).

**Theorem 3.1 (Local Ratio [3])** Let \( F \) be a set of constraints and let \( w, w_1 \) and \( w_2 \) be benefit functions such that \( w = w_1 + w_2 \). If \( x \) is \( r \)-approximate with respect to \( w_1 \) and \( w_2 \) then \( x \) is \( r \)-approximate with respect to \( w \).

We show that any independent set weighs at most \( 4w_1(v[t', t, j]) \) with respect to \( w_1 \).

**Lemma 3.3** \( \text{OPT}(w_1) \leq 4 \cdot w_1(v[t', t, j]) \).
Proof. By construction of $w_1$ vertices that overlap or intersect $v[t',t,j]$ (including $v[t',t,j]$) are given a weight of $w_1(v[t',t,j])$ (Line 3). All other vertices have a weight of zero and their contribution to the weight of the independent set is null. Therefore, we consider only vertices that intersect or overlap $v[t',t,j]$. We divide these vertices into two sets. Denote by $I(v[t',t,j])$ the set of vertices that intersect $v[t',t,j]$ (including $v[t',t,j]$) and by $O(v[t',t,j])$ the set of vertices that overlap $v[t',t,j]$ (including $v[t',t,j]$).

Consider two vertices $v[t'_1,t_1,j], v[t'_2,t_2,j] \in I(v[t',t,j])$. Both vertices intersect $v[t',t,j]$, namely $t'_1 < t$ and $t'_2 < t$. Also, $t \leq t_1$ and $t \leq t_2$ since $t$ is minimum (Line 2). Thus, $v[t'_1,t_1,j]$ and $v[t'_2,t_2,j]$ intersect. This means that $I(v[t',t,j])$ induces a clique, and hence any independent set cannot contain more than one vertex from $I(v[t',t,j])$.

Observe that any $v[s',s,k] \in O(v[t',t,j])$ satisfies $s = t$, $s' = t$, or $s' = t'$ ($s' = t'$ is not possible since $t$ is minimum). Thus, we divide $O(v[t',t,j])$ into three subsets:

\[
\begin{align*}
O_1(v[t',t,j]) &= \{v[s',s,k] \mid v[s',s,k] \in O(v[t',t,j]) \text{ and } s = t\} \\
O_2(v[t',t,j]) &= \{v[s',s,k] \mid v[s',s,k] \in O(v[t',t,j]) \text{ and } s' = t\} \\
O_3(v[t',t,j]) &= \{v[s',s,k] \mid v[s',s,k] \in O(v[t',t,j]) \text{ and } s' = t' \neq 0\}
\end{align*}
\]

Consider two vertices $v[t_1,t,k], v[t_2,t,l] \in O_1(v[t',t,j])$. If $k = l$ the vertices intersect (since $t_1,t_2 < t$), otherwise the vertices overlap. Thus, $O_1(v[t',t,j])$ induces a clique, and an independent set cannot contain more than one vertex from $O_1(v[t',t,j])$. Similarly, it can be shown that $O_2(v[t',t,j])$ and $O_3(v[t',t,j])$ induce cliques. Hence, an independent set cannot contain more than one vertex from $O_2(v[t',t,j])$ and one vertex from $O_3(v[t',t,j])$.

Putting it all together, an independent set contains at most four vertices. The weight of each vertex is $w_1(v[t',t,j])$, thus its maximum weight is $4 \cdot w_1(v[t',t,j])$. ■

Theorem 3.2 Algorithm IS is a 4-approximation algorithm.

Proof. The proof is by induction on the recursion. At the induction base (Line 1 of IS) $V = \emptyset$, and the algorithm returns an optimal solution. For the induction step, we assume that $S^+$ is 4-approximate with respect to $(G^+,w_2)$. We show that $S$ is 4-approximate with respect to $(G,w_1)$ and $(G,w_1)$. This proves, by Local Ratio Theorem, that $S$ is 4-approximate with respect to $(G,w)$. $V \setminus V^+$ contains vertices with non-positive weight due to Line 4, thus the optimum with respect to $(G,w_2)$ is not greater than the optimum with respect to $(G^+,w_2)$. It follows that $S^+$ is 4-approximate with respect to $(G,w_2)$. Due to Line 3, $w_2(v[t',t,j]) = 0$, therefore adding $v[t',t,j]$ to $S$ (in Line 6) does not reduce the weight of the independent set. Hence $S$ is 4-approximate with respect to $(G,w_2)$.

Next, $S$ contains $v[t',t,j]$ or at least one of its neighbors due to Line 6. Hence, $S$ weighs at least $w_1(v[t',t,j])$ with respect to $(G,w_1)$ where the maximum independent set is $4 \cdot w_1(v[t',t,j])$ by Lemma 3.3. It follows that $S$ is 4-approximate with respect to $(G,w_1)$. ■

4 Variable Size Objects

In this section we consider the general case where different objects may have different sizes. In this case we divide the time axis such that the transmission of one object requires an integral number of time slots. One way to satisfy this requirement is to find the greatest common divider of the objects’ sizes and use this value as the length of the time slot. We denote by $\tau_j$ the number of time
slots it takes to broadcast \( p_j \). For instance, if the transmission times of \( p_1 \), \( p_2 \) and \( p_3 \) are 0.4, 1 and 1.4 millisecond, respectively, the length of the time slot can be 0.2 millisecond and \( \tau_1 = 2 \), \( \tau_2 = 5 \) and \( \tau_3 = 7 \).

In this section we present a 10-approximation algorithm for this version of TDMSM using the same technique that has been used in the previous section. Before doing so we formally define the problem and the benefit function for the case of variable sizes. We also discuss the relations between triples for this case.

### 4.1 Problem Definition and Benefit Function

The problem definition and the benefit function in this case are similar to those defined in Section 2. However, the different sizes of the objects induce some changes that are reflected in the feasibility of the schedule and in the exact definition of the benefit function.

In the variable size case when an object \( p_j \) is said to be transmitted at time slot \( t \), it actually means that \( \tau_j \) time slots are used for this transmission starting with time slot \( t \). Thus, in a feasible schedule these \( \tau_j \) time slots cannot be used for other transmissions. Also, if \( t \) and \( t' \) are two consecutive transmissions of \( p_j \), then \( t \) cannot be smaller than \( t' + \tau_j \), namely the gap between two consecutive transmissions is at least \( \tau_j \). Henceforth we assume that every triple we consider satisfy the inequality \( t \geq t' + \tau_j \) if \( t' > 0 \). Note that in Section 2.1, where \( \tau_j = 1 \) for every \( j \), we implicitly considered this restriction by defining the benefit function to be 0 where \( t' \geq t \).

### 4.2 Relations Between Triples

In order to characterize a feasible set of triples we extend the definitions from Section 3 as follows. We say that the triples \((t', t, j)\) and \((s', s, j)\) intersect if \( t \neq s', s \neq t' \) and the intervals \([t', t + \tau_j]\) and \([s', s + \tau_j]\) intersect. For example, in Figure 5a the triple \((t', t, j)\) intersects both \((t'_1, t_1, j)\) and \((t'_3, t_3, j)\) and does not intersect \((t'_2, t_2, j)\). Two triples \((t', t, j)\) and \((s', s, k)\), where \( j \neq k \), overlap if one of two intervals \([t, t + \tau_j]\) and \([t', t' + \tau_j]\) intersects with one of the two intervals \([s, s + \tau_k]\) and \([s', s' + \tau_k]\). In this case we require that either \( t' \) or \( s' \) are not equal to zero. (Recall that in Definition 2.1 \( B[0, t, j] \) represents the benefit of sending \( p_j \) at time slot \( t \) assuming that it was not sent earlier.) For example, in Figure 5b the intervals \([t, t + \tau_j]\) and \([s', s' + \tau_k]\) intersect and therefore the triples overlap.

![Figure 5: Relations between triples.](image-url)
As mentioned above, the gap between two consecutive transmissions of \( p_j \) is at least \( \tau_j \). Thus, not every triple can describe two consecutive transmissions in a feasible schedule. The following definitions consider this restriction. A triple \((t', t, j)\) is called a feasible if \( t > t' = 0 \) or \( t \geq t' + \tau_j \). A set of triples is called feasible if it does not contain overlapping, intersecting, or infeasible triples. (In Section 3 we implicitly consider this restriction since all objects had transmission time of one time slot and we referred to a triple \((t', t, j)\), in which \( t > t' \).) A feasible set of triples \( S \) is called maximal if \( \forall (t', t, j) \not\in S \), the set \( S' = \{(t', t, j)\} \cup S \) is not feasible.

The proofs of the following lemmas are similar to these given in Section 3.

**Lemma 4.1** A set of triples that describes a feasible schedule is feasible.

**Lemma 4.2** A maximal set of triples induces a feasible schedule.

### 4.3 Approximation Algorithm

Given a benefit function, we can construct a weighted graph \( G = (V, E) \) as we did in Section 3. In this graph every feasible triple \((t', t, j)\) represents a vertex \( v[t', t, j] \in V \) that has the weight of \( w(v[t', t, j]) = B[t', t, j] \). An edge \((v[t', t, j], v[s', s, k]) \in E \) if \((t', t, j)\) intersects or overlaps \((s', s, k)\).

We use the same correlation between a maximal independent set and a feasible schedule and we show an algorithm that finds an independent set in \( G \). This algorithm is a modified version of Algorithm IS in which in every recursive call we choose a vertex that corresponds to an object with the smallest transmission time, namely the following lines should replace Line 2 in Algorithm IS:

2a: Choose \( j \) such that \( \tau_j \leq \tau_k \) for every \( k \)

2b: Choose \( v[t', t, j] \in V \), such that \( t \leq s \) for every \( v[s', s, j] \in V \)

As in the unit size case, in order to obtain a feasible schedule, one should extend the output of the algorithm to a maximal independent set.

The analysis on the modified algorithm is similar to the one used in the previous section. The only difference is in the weight of the maximum weight independent set with respect to \( w_1 \).

**Lemma 4.3** \( \text{OPT}(w_1) \leq 10 \cdot w_1(v[t', t, j]) \).

**Proof.** According to the construction of \( w_1 \) only vertices that overlap or intersect \( v[t', t, j] \) (including \( v[t', t, j] \)) have a weight of \( w_1(v[t', t, j]) \). All other vertices have a weight of zero and their contribution to the weight of the independent set is null. Therefore, we refer only to vertices that intersect or overlap \( v[t', t, j] \) (including \( v[t', t, j] \)). We divide the vertices into two sets using the following notation. Denote by \( I(v[t', t, j]) \) the set of vertices that intersect \( v[t', t, j] \) including \( v[t', t, j] \). Denote by \( O(v[t', t, j]) \) the set of vertices that overlap \( v[t', t, j] \) including \( v[t', t, j] \).

We partition \( I(v[t', t, j]) \) into two subsets:

\[
I_1(v[t', t, j]) = \{v[s', s, j] \mid v[s', s, j] \in I(v[t', t, j]) \text{ and } s' < t\}
\]

\[
I_2(v[t', t, j]) = \{v[s', s, j] \mid v[s', s, j] \in I(v[t', t, j]) \text{ and } s' > t\}
\]

In other words, \( I_1(v[t', t, j]) \) is the set of vertices that intersect \( t \) while \( I_2(v[t', t, j]) \) is the set of vertices that intersect \( t + \tau_j \) and do not intersect \( t \). See example in Figure 6.

According to the definition of \( I_1(v[t', t, j]) \) every two vertices \( v[s'_1, t_1, j], v[s'_2, t_2, j] \in I_1(v[t', t, j]) \) satisfy \( s'_1, s'_2 < t \) and \( t \leq s_1, s_2 \) since \( t \) is minimum. Hence, \( v[s'_1, s_1, j] \) and \( v[s'_2, s_2, j] \) intersect.
Figure 6: Partition of $I(v'[t', t, j])$ into $I_1(v'[t', t, j])$ and $I_2(v'[t', t, j])$: $v'[t_1', t_1, j], v'[t_2', t_2, j] \in I_1(v'[t', t, j])$, while $v[t_3', t_3, j] \in I_2(v'[t', t, j])$.

It follows that $I_1(v'[t', t, j])$ induces a clique, and therefore an independent set cannot contain more than one vertex from $I_1(v'[t', t, j])$. We use a similar argument on $I_2(v'[t', t, j])$. Every two vertices $v[s_1', s_1, j], v[s_2', s_2, j] \in I_2(v'[t', t, j])$ satisfy $t < s_1', s_2' < t + \tau_j$. In addition, $s_1, s_2 > t + \tau_j$, which means that $v[s_1', s_1, j]$ and $v[s_2', s_2, j]$ intersect. Hence, $I_2(v'[t', t, j])$ induces a clique, and an independent set cannot contain more than one vertex from it.

We now turn to deal with $O(v'[t', t, j])$. We partition this set into four subsets, each corresponds to a different relation between the $v'[t', t, j]$ and an overlapping vertex $v[s', s, k]$. The four subsets are:

\[
O_1(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } s < t' + \tau_j \text{ and } t' < s + \tau_k\}, \\
O_2(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } s' < t' + \tau_j \text{ and } t' < s' + \tau_k\}, \\
O_3(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } s < t + \tau_j \text{ and } t < s + \tau_k\}, \\
O_4(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } s' < t + \tau_j \text{ and } t' < s' + \tau_k\}
\]

In other words, $O_1(v'[t', t, j])$ contains vertices that correspond to triples whose right side overlaps the left side of $(t', t, j)$. $O_2(v'[t', t, j])$ contains vertices that correspond to triples whose left side overlaps the left side of $(t', t, j)$. $O_3(v'[t', t, j])$ and $O_4(v'[t', t, j])$ are defined similarly with respect to the right side of $(t', t, j)$. See examples in Figure 7. Observe that $O(v'[t', t, j]) = \bigcup_{i=1}^{4} O_i(v'[t', t, j])$.

In addition, $O_1(v'[t', t, j])$ can be partitioned to the following two subsets:

\[
O'_1(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } t' < s < t' + \tau_j \text{ and } t' < s + \tau_k\}, \\
O''_1(v'[t', t, j]) = \{v[s', s, k] \mid v[s', s, k] \in O(v'[t', t, j]) \text{ and } s \leq t' \text{ and } t' < s + \tau_k\}
\]

In other words, $O''_1(v'[t', t, j])$ is the set of vertices that intersect $t'$ while $O'_1(v'[t', t, j])$ is the set of vertices that intersect $t' + \tau_j$ and does not intersect $t'$.\footnote{\(O'_1(v'[t', t, j])\) contains redundant information since $t' \leq t_1$ implies that $t' < t_1 + \tau_k$. However, we did not remove this redundant information since it emphasizes the division of $O'_1(v'[t', t, j])$}

For instance, in Figure 7a, $(t'_2', t_2, j) \in O'_1(v'[t', t, j])$ while $(t_1', t_1, j) \in O'_1(v'[t', t, j])$. A similar partition can be done with $O_2(v'[t', t, j])$, $O_3(v'[t', t, j])$ and $O_4(v'[t', t, j])$ as well. (See Figure 7.)

The definition of $O'_1(v'[t', t, j])$ implies that every two vertices $v[s_1', s_1, k]$ and $v[s_2', s_2, l]$ in $O'_1(v'[t', t, j])$ satisfy $t' \leq s_1 < t' + \tau_j$ and $t' \leq s_2 < t' + \tau_j$. In addition, $\tau_j \leq \tau_k$, $\tau_l$, therefore $t' + \tau_j \leq s_1 + \tau_k$ and $t' + \tau_j \leq s_2 + \tau_l$. Hence, $s_1 < s_2 + \tau_l$ and $s_2 < s_1 + \tau_k$, namely if $k = l$ then $v[s_1', s_1, k]$ intersects $v[s_2', s_2, j]$ and if $k \neq l$ then $v[s_1', s_1, k]$ overlap $v[s_2', s_2, j]$. In other words, $O'_1(v'[t', t, j])$ induces a clique and an independent set cannot contain more than one vertex.
from it. A similar analysis can be done for one vertex from each subset. Its maximum weight is $10 \cdot w_{1}(v[t', t, j])$.

Theorem 4.1
The modified version of Algorithm IS is a 10-approximation algorithm.

5 Using Several Multicast Channels

In this section we consider the variant on TDMSM in which $k$ multicast channels are available. In this problem we are given a benefit function and our goal is to find a feasible schedule to $k$-channels with maximum benefit. We refer to this variant as $k$-TDMSM.

We first show that $k$-TDMSM is NP-hard even on unit size objects. We do so by a simple reduction from TDMSM. Given an instance of TDMSM with $n$ objects, we add $k-1$ new objects $p_{n+1}, \ldots, p_{n+k-1}$ such that

$$B[t', t, n + j] = \begin{cases} M & t = t' + 1, \\ 0 & \text{otherwise}, \end{cases}$$

for every $j \in \{1, \ldots, k-1\}$ and $t \geq 1$, where $M = \sum_{j=1}^{n} \sum_{t} \sum_{t' < t} B[t', t, j]$. Clearly, every optimal solution will transmit the new $k-1$ objects at every time slot. Hence, the optimum value of the $k$-TDMSM instance is larger than the optimum value of the original TDMSM instance by an additive

![Figure 7: Partition of $O(v[t', t, j])$.](image-url)
factor of \((k - 1) \cdot |T| \cdot M\). It follows that if one is able to compute an optimal solution for \(k\)-TDMSM, one is also able to compute an optimal solution for TDMSM.

Next, we show how to compute a 4-approximate solution for \(k\)-TDMSM in the case of unit sizes. We do this by using a modified version of Algorithm IS in which \(w_1\) is replaced by the following weight function:

\[
    w_1(v[s', s, k]) = w(v[t', t, j]) \cdot \begin{cases} 
        1 & v[s', s, k] \in I(v[t', t, j]), \\
        1/k & v[s', s, k] \in O(v[t', t, j]), \\
        0 & \text{otherwise.}
    \end{cases}
\]

The analysis of the algorithm is similar to the one used for Algorithm IS. We only need to prove that (i) the optimum with respect to \(w_1\) is at most \(4 \cdot w_1(v[t', t, j])\), and (ii) \(w_1(S) \geq w_1(v[t', t, j])\), where \(S\) is the solution returned by the recursive call.

We prove that \(\text{OPT}(w_1) \leq 4 \cdot w_1(v[t', t, j])\) using a proof similar to the proof of Lemma 3.3. We use the terms vertices and triples interchangeably. First, only one triple from \(I(v[t', t, j])\) can be contained in a feasible solution. Furthermore, since there are \(k\) multicast channels, a solution can contain up to \(k\) triples from \(O_i(v[t', t, j])\) for every \(i \in \{1, 2, 3\}\). It follows that the total weight of the solution is at most \((1 + 3k \cdot 1/k) \cdot w_1(v[t', t, j]) = 4 \cdot w_1(v[t', t, j])\).

Next, we show that \(w_1(S) \geq w_1(v[t', t, j])\). If \(v[t', t, j] \not\in S\) this is clearly true. Otherwise, \(v[t', t, j]\) cannot be added to \(S\). It follows that either \(S \cap I(v[t', t, j]) \neq \emptyset\) or \(|S \cap O(v[t', t, j])| = k\), and in both cases \(w_1(S) \geq w_1(v[t', t, j])\).

In order to obtain a 10-approximation algorithm for \(k\)-TDMSM with variable sizes one needs to use the Algorithm from Section 4 with the new \(w_1\). The solution returned is 10-approximate since \(\text{OPT}(w_1) \leq 10 \cdot w_1(v[t', t, j])\).

6 Greedy Algorithm

The approximation results described so far hold for any non-negative benefit function. However, when using this scheme in practical scenarios, the benefit function may obey several non-trivial limitations. In this section we consider certain limitations on the benefit function, and we show that in this case a greedy algorithm computes 3-approximate solutions.

6.1 Restricted Benefit Function

Consider a scenario in which a source sends object \(p_j\) via multicast at two consecutive time slots \(t_1\) and \(t_3\), where \(t_3 > t_1\). The benefit of such a schedule is \(B[t_1, t_3, j]\). If the source transmits \(p_j\) also at time slot \(t_2\), where \(t_1 < t_2 < t_3\), then the total benefit that of these transmissions is \(B[t_2, t_3, j] + B[t_1, t_2, j]\). In general, this additional transmission may decrease the total benefit (i.e., \(B[t_1, t_2, j] + B[t_2, t_3, j] < B[t_1, t_3, j]\)). However, this does not happen in practice. Recall that \(B[t_1, t_3, j]\) represents the benefit of broadcasting \(p_j\) at time slot \(t_3\) given that the previous multicast transmission was performed at time slot \(t_1\). Therefore, a multicast transmission at \(t_2\) should not add unicast transmissions of \(p_j\). Therefore, in this section we assume that for any object \(p_j\), and time slots \(t_1 < t_2 < t_3\) the following triangle inequality holds: \(B[t_1, t_2, j] + B[t_2, t_3, j] \geq B[t_1, t_3, j]\).

Observe that due to the triangle inequality if we are given a solution that does not use the multicast channel in some time slot \(t\), we can send an arbitrary object in time slot \(t\) without decreasing the
benefit. Therefore, we may assume that there exists an optimal solution that uses the multicast channel in every time slot.

Next we consider a restriction that indicates that benefit cannot appear suddenly. Assume that a source sends $p_j$ via multicast at time slots $t_2$ and $t_3$ where $t_3 > t_2$. The benefit of such schedule is $B[t_2, t_3, j]$. In many practical scenarios this benefit comes from the fact that the object was sent at $t_3$ (regardless of the previous time it was sent) and from the fact that it was already sent at $t_2$. Thus, for any time slot $t_1$ such that $t_1 < t_2$ the following skewed triangle inequality should hold:

$$B[t_1, t_2, j] + B[t_1, t_3, j] \geq B[t_2, t_3, j].$$

TDMSM remains NP-hard even under both restrictions. This can be proven by slightly modifying the reduction from Section 2.3. Instead of using binary values we use the values 1 and 2. In this case, $B[t_1, t_2, j] + B[t_2, t_3, j] \geq 1+1 = 2 \geq B[t_1, t_3, j]$, and $B[t_2, t_3, j] \leq 2 = 1+1 \leq B[t_1, t_2, j] + B[t_1, t_3, j]$ for every $t_1, t_2, t_3$ such that $t_1 < t_2 < t_3$.

6.2 The Greedy Algorithm

In this section it would be convenient to treat a schedule $S$ as a function, where $S(t)$ is the index $j$ of the object that was sent at time slot $t$. For the purposes of the algorithm we denote by $\text{prev}(j, t)$ the last time object $p_j$ was sent before time slot $t$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\epsilon$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) Object 1.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2$\epsilon$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) Object 2.

Figure 8: The table on the left represents the benefit function of Object 1, and the one on the right the benefit function of Object 2.

We start our discussion by showing that Algorithm Greedy does not guaranty any approximation ratio in the general case. For instance, consider the benefit function that is described in Figure 8. In this case the benefit of the optimal schedule is $B[0,1,1] + B[1,2,1] = 1 + \epsilon$, while the benefit of the schedule returned by the algorithm is $B[0,2,2] + B[1,2,2] = 2\epsilon$.

Next, we show that Algorithm Greedy computes 3-approximate solutions for the special case where the benefit function satisfies the above mentioned restrictions.

Theorem 6.1 Algorithm Greedy computes 3-approximate solutions for the case where the benefit function is restricted.

Proof. We use the following notation. Let $\text{prev}_S(j, t)$ be the time slot before time $t$ in which object $j$ was last broadcasted in solution $S$. If object $j$ was not broadcasted before time $t$ we
define \( \text{PREVS}(j, t) = 0 \). Also, let \( \text{NEXTS}(j, t) \) be the first time slot after time \( t \) in which object \( j \) was broadcasted in solution \( S \). If object \( j \) was not broadcasted after time \( t \) we define \( \text{NEXTS}(j, t) = T + 1 \).

We denote by \( B(S) \) the benefit gained by the solution \( S \).

Let \( G \) be the greedy solution and let \( \text{OPT} \) be an optimal solution. We prove that Algorithm Greedy is a 3-approximation algorithm by showing that \( B(G) \geq \frac{1}{3} \cdot B(\text{OPT}) \). We note that it is shown in [5] that \( B(G) \geq \frac{1}{3} \cdot B(\text{OPT}^*) \), where \( \text{OPT}^* \) is the fractional optimum of an LP relaxation of the problem.

We define a series of hybrid solutions denoted by \( H_0, \ldots, H_T \), where

\[
H_t(i) = \begin{cases} 
G(i) & i \leq t, \\
\text{OPT}(i) & i > t.
\end{cases}
\]

By the triangle inequality,

\[
\Delta_t = B[\text{PREVS}_t(j, t), t, j] + B[t, \text{NEXTS}_t(j, t), j] - B[\text{PREVS}_t(j, t), \text{NEXTS}_t(j, t), j] - B[t, \text{NEXTS}_t(j, t), \ell] + B[t, \text{NEXTS}_t(j, t), j, \ell].
\]

By the skewed triangle inequality

\[
\Delta_t \leq B[t, \text{NEXTS}_t(j, t), j, \ell] - B[\text{PREVS}_t(j, t), \text{NEXTS}_t(j, t), j] \leq B[\text{PREVS}_t(j, t), t, j]
\]

which means that \( \Delta_t \leq 2B[\text{PREVS}_t(j, t), t, j] \). \( H_t(i) = G(i) \) for every \( i < t \) by the definition of \( H_t \), thus \( \Delta_t \leq 2B[\text{PREVS}_G(j, t), t, j] \). Furthermore, \( B[\text{PREVS}_G(j, t), t, j] \leq B[\text{PREVS}_G(\ell, t), t, j] \) since Algorithm Greedy picks the object that maximizes the benefit, and therefore \( \Delta_t \leq 2B[\text{PREVS}_G(\ell, t), t, \ell] \). It follows that

\[
B(\text{OPT}) - B(G) = B(H_0) - B(H_T) = \sum_{t=1}^{T} \Delta_t \leq 2 \cdot \sum_{t=1}^{T} B[\text{PREVS}_t(G(t), t), t, G(t)] = 2 \cdot B(G).
\]

Hence, \( B(G) \geq \frac{1}{3} \cdot B(\text{OPT}) \) as required.

7 Online Problem

The application that distributes the data usually does not know which objects will be required in advance, and it is required to determine which object to transmit via multicast at any given time according to information that has been received up to this time. For instance, CDN servers distribute web objects according to requests that are received from clients. In a reliable multicast
protocol the retransmissions are required due to unpredicted packet loss. Hence, the application must use an online algorithm.

In this section we examine the online case in which an algorithm should determine at any given time, which object should be sent via multicast according to current information, namely without having information regarding future requests. We specifically consider the problem in which the objective function is to maximize the number of unicast transmissions that were avoided due to multicast transmissions. That is, the benefit is the difference between the number of unicast transmissions with and without the availability of the multicast channel. We prove that any deterministic online algorithm is not competitive for this problem.

Let $A$ be a deterministic online algorithm. Given $A$ we define a nemesis instance $I$ consisting of $n$ unit size objects $p_1, \ldots, p_n$ and one client. (The same ideas work with multiple clients.) In the first $\frac{n}{2}$ time slots no object is required by the client. Hence, the online algorithm $A$ can broadcast up to $\frac{n}{2}$ different objects (one object per time slot). Without loss of generality, we assume that $A$ did not broadcasted the objects $p_1, p_2, \ldots, p_{\frac{n}{2}}$ during the first $\frac{n}{2}$ time slots. In time slot $\frac{n}{2}+1$, the client requires objects $p_1, p_2, \ldots, p_{\frac{n}{2}}$.

We show that $A(I) = 1$ while $\text{OPT}(I) = \frac{n}{2}$. Algorithm $A$ can broadcast only one object at time slot $\frac{n}{2}+1$ while the rest of the $\frac{n}{2} - 1$ objects are sent via unicast. Therefore, only one unicast transmission is avoided by $A$, or $A(I) = 1$. On the other hand, the adversary can broadcast all objects without using unicast dissemination at all by broadcasting object $p_i$ at time slot $i$. Therefore, $\text{OPT}(I) = \frac{n}{2}$. It follows that for every $n$ the ratio between $\text{OPT}(I)$ and $A(I)$ is $\frac{n}{2}$. We note that this sequence can be repeated periodically by assuming that the objects expire after every $n$ time slots.

For the case of $k$ channels, we can use an instance consisting of $n \cdot k$ objects, where half of them, that were not transmitted by $A$, are required at slot $\frac{n}{2}$. In this case $A$ can save up to $k$ unicast transmission, while $\text{OPT}(I) = k \cdot \frac{n}{2}$. It follows that in this case the ratio between $\text{OPT}(I)$ and $A(I)$ is also $\frac{n}{2}$.

**Acknowledgments**

We thank the anonymous referees for their helpful comments and suggestions.

**References**


