PERGAMON

Computers & Operations Research 26 (1999) 461–480

computers & operations research

Optimum multicast of multimedia streams

Chu-Fu Wang[1], Bo-Rong Lai[2], Rong-Hong Jan*[,3]

Department of Computer and Information Science, National Chiao Tung University, Hsinchu, 30050, Taiwan

Received February 1998; received in revised form August 1998

Scope and purpose

As multimedia service becomes more widely used through computer networks, conserving network resources becomes increasingly important. Multicast communications can save network bandwidths when delivering data to multiple destinations. Therefore, many researchers have given much attention to multicast routing problems. Most of the multicast routing problems only consider a single multicast session. However, in the real world, several multicast sessions will be broadcast, simultaneously. These multicast sessions will contend for the limited resources (such as bandwidth) of networks. This creates a new network optimization problem which is different from any other multicast routing problem. In this research, we studied an optimal video distribution problem which arises from video on demand (VOD) systems with multiple multicast sessions. We believe that the results are useful for improving video distribution methods in multimedia networks. In addition, this study would be applicable on the field of network flows, especially multicommodity flows.

Abstract

In a VOD system, multicast is a preferred method for saving network bandwidth. That is, customers requesting the same video program over a small time interval can be arranged in a multicast tree (group), and

* Corresponding author. Fax: 886-3-5721490; e-mail: rhjan@cis.nctu.edu.tw.

[1]Chu-Fu Wang received the B.S. degree in Applied Mathematics from National Cheng Kong University, and the M.S. degree in Computer and Information Science from National Chiao Tung University, Taiwan, 1995. Since 1995, he has been working toward the Ph.D. degree in Computer and Information Science at National Chiao Tung University, Taiwan. His research interests include multicast distribution, network reliability, and network optimization.

[2]Bo-Rong Lai received the B.S. and M.S. degrees in Computer and Information Science from National Chiao Tung Univeristy, Taiwan, in 1994, and 1996, respectively. His research interests include computer networks and multicast distribution.

[3]Rong-Hong Jan received the B.S. and M.S. degrees in Industrial Engineering, and the Ph.D. degree in computer Science from National Tsing Hua University, Taiwan, he is currently a Professor in the Department of Computer and Information Science, National Chiao Tung University. He has been a Visiting Associate Professor in the Department of Computer Science, University of Maryland. His research interests include computer networks, distributed systems, network reliability, and operations research.

PII: S0305-0548(98)00075-6

then the video server sends a video stream via this tree to customers. In this research, we considered how to arrange a set of multicast trees such that the number of customers served is maximized and the link capacity constraint is maintained. For a directed acyclic graph (DAG), we proposed a branch-and-bound algorithm to solve this problem and the solution method is illustrated by example. Next, we extended the algorithm to find an approximate solution for a general graph case. It is shown, through simulations on randomly generated graphs that the solution for our approximation method is very close to the optimal solution.

Keywords: Integer programming; Branch-and-bound algorithm; Network optimization; Multicast routing

1. Introduction

The demand for multimedia (such as video or audio) transmissions is increasing rapidly and the transmission of multimedia is time sensitive. In order to maintain quality of service (QoS), networks have to allocate enough bandwidth for transmitting multimedia data. Two types of multimedia transmissions are usually used. They are point-to-point transmission and point-to-multipoint transmission. The point-to-multipoint transmission, known as multicast, occurs when the multimedia data is to be delivered to a subset of nodes in the network. Broadcast is a special case of multicast in which the data is to be delivered to a set that includes all the network nodes.

In a multicast communication, the source sends identical data to all destinations. Since the data can be duplicated at switching nodes (the intermediate nodes), it is not necessary for the source node to send separate copies to all destinations. Thus, a good multicasting path can help to reduce the number of redundant streams flowing on the network. For example, consider a VOD system to which several users may subscribe an identical multimedia stream over a small time interval. Because the transmission of the video stream consumes the high bandwidth, it is a good idea for video server to collect the subscription information and then find a multicast tree to transmit a copy of the subscribed video stream to all subscribers.

Usually, two types of objective functions are considered in solving multicast transmission problems. One function minimizes the transmission cost. The other function minimizes transmission delay. For unconstrained case, a least cost multicasting path finding problem is known as the Steiner tree problem. In this problem, we are given a subset $S \subseteq N$ of nodes, called member nodes, and we wish to determine the minimum cost tree that has to contain all of the member nodes in S. The Steiner tree problem is known to be NP-complete [1]. Several exponential time algorithms have been developed for finding exact solution. Among these works, Hakimi [2] proposes a spanning tree enumeration algorithm that runs in $O(k^2 2^{n-k} + n^3)$ time, where $n = |N|$, and $k = |S|$. Levin [3] proposes a dynamic programming approach algorithm that runs in $O(3^k n + 2^k n^2 + k^2 n)$ time. Other exponential time algorithms have also appeared without time complexity analysis. Beasley [4] gives a Lagrangean relaxation algorithm. Another 0–1 linear programming algorithm in a slightly different version is given by Wong [5]. Also, there are many heuristic algorithms [6–8] that have been presented for solving Steiner tree problem. For detail description and more lectures on both exact and heuristic methods, one can refer two survey papers, Hwang and Richards [9] and Winter [10]. On the other hand, if the goal is to minimize the delay, a shortest path tree is the solution [11–14]. The problem can be solved by finding the

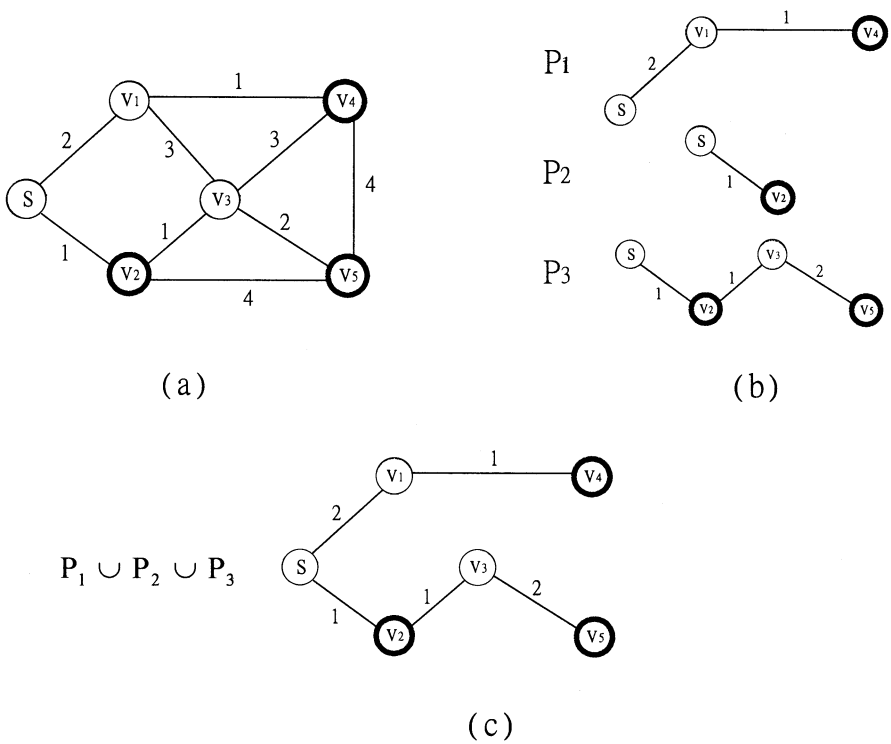

Fig. 1. An example for solving multicast transmission problem, which the objective function is to minimize the transmission delay.

shortest paths from the source node to all of the destination nodes and then merging these paths into a single multicast tree. For example, in Fig. 1a, node s is the source node and nodes v_2, v_4 and v_5 are the destination nodes. Each link is associated with a nonnegative integer to represent its transmission delay. Then the optimal solution for this problem (see Fig. 1c) can be obtained by merging three shortest paths, which are shown in Fig. 1b, into a tree.

In previous research, most of the multicast problems only considered a single multicast session. In the single multicast session, the major concern is how to find a minimum cost Steiner tree. Then use this tree to transmit the same data to every multicast member. However, in the real world, several multicast sessions will happen, simultaneously. For example, if there are a lot of video programs available in a VOD system for subscriptions, the server will establish several multicast sessions (i.e., each video has a multicast session) to transmit the video stream to customers. Note that the amounts of network resources are fixed. When multiple multicast sessions are setup simultaneously, they will contend for these network resources. If the resources are large enough, all multicast sessions can be setup. Otherwise, some of the sessions will fail. Therefore, the important issue is how to allocate network resources to each multicast session. The objective function for multisession problem can be the minimization of the total cost or the maximization of the number of VOD customers served. Note that for a VOD system, which provides pay-movies, the more number of customers served means the more revenues earned.

In this research, we considered a VOD system with multiple multicast sessions. The problem was to find a set of mulitcast trees such that the number of VOD customers served was maximized under the bandwidth constraint. For the directed acyclic network, a branch-and-bound method was proposed to solve this problem. The solution method can be modified to solve the general graph case.

The remainder of this paper is organized as follows. Section 2 states the formulation for this problem. Section 3 presents a branch-and-bound method that can solve this problem over a directed acyclic network. Section 4 presents a DAG's based heuristic algorithm to solve the general graph using a modified branch-and-bound method. In Section 5, we give simulation results that illustrate the performance of our proposed method. Finally, concluding remarks are given in Section 6.

2. Statement of the problem

Consider the VOD system shown in Fig. 2. There is one VOD server and four switching nodes (e.g., ATM switches) in this system. The VOD server is responsible for providing the video

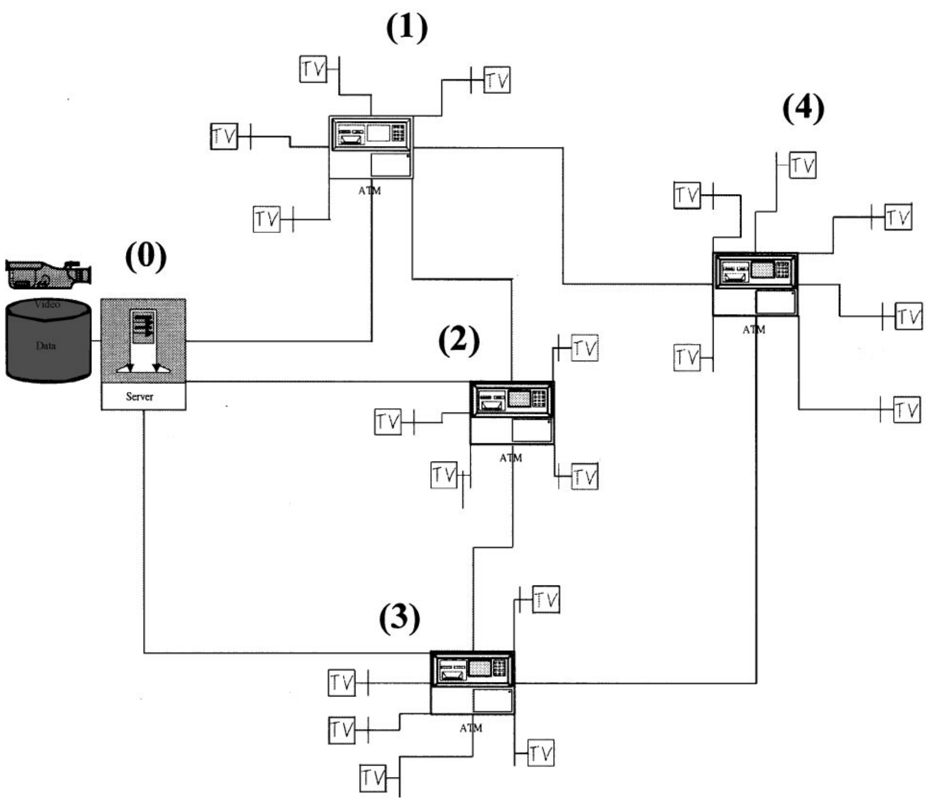

Fig. 2. A VOD system with one VOD Server and four switching nodes.

programs, and the switching node has the ability to duplicate the video stream that comes from the server. The customer's terminal device is connected to a switching node and the customer can receive the video program from the down link of the connected switching node. The switching nodes and the server are connected to each other in some fashion by directed links. The available capacity of a link is defined as the maximum number of video streams that can be served. In our model, each video program requires a unit of the link capacity for transmission. In order to guarantee the quality of service, we also assume that when a multicast session is established, the bandwidth allocated for this session is reserved until the session finishes.

Suppose there are m video programs $P_1, P_2, \dots, P_m$ available on the server. Because the switching node can duplicate the video stream, if a video stream flows into a switching node, all subscribers connected to that switching node can obtain the program in that stream. Therefore, the *bid* b_j^k of a video program P_k at a switching node j is defined as the number of customers who subscribe to video program P_k. The *bid vector* $(b_j^1, b_j^2, \dots, b_j^m)$ for switching node j is the collection of bid b_j^k at switching node j. We assume that any video program that flows in each link requires a unit of link capacity. Hence, we associate a nonnegative integer (link capacity) r_{ij} for each link (i, j) to represent the maximum number of video programs which can flow on that link (i, j) . Therefore, a VOD system shown in Fig. 2 is modeled as a network $G = (V, E)$ shown in Fig. 3a where V is a set of nodes containing a server node (node 0) and all switching nodes (nodes $1, 2, \dots, n-1$) and E is a set of communication links between these nodes. Let $v_j^k = 1$ denote the video stream k flowing into node j and $v_j^k = 0$, otherwise. Thus, the total number of subscribers being served (called source gain) is $\sum_{j=1}^{n-1}\sum_{k=1}^{m} v_j^k \cdot b_j^k$ where m is number of streams and $n-1$ is number of switching nodes. For example, the source gain for the multicast scheme in Fig. 3b is 23. Fig. 3c shows that the video distribution scheme in Fig. 3b is composed using a set of multicast trees $\{T_1, T_2, T_3\}$ where T_1, T_2 and T_3 can be used for multicast sessions of video programs P_1, P_2 and P_3, respectively.

In this research, the problem was formulated to find a multicast scheme such that the source gain is maximized and the bandwidth constraint is maintained. We call this problem an optimum source gain multicast problem (OSGMP).

Notation:

n: number of nodes.
m: number of video streams.
r_{ij}: capacity of link (i, j) .
b_j^k: switch j's bid for stream k.
v_j^k: the decision variable, $v_j^k = \begin{cases} 1, & \text{stream } k \text{ flows into node } j, \\ 0, & \text{otherwise.} \end{cases}$
x_{ij}^k: the decision variable, $x_{ij}^k = \begin{cases} 1, & \text{stream } k \text{ flows in link}(i, j), \\ 0, & \text{otherwise.} \end{cases}$
M: big M; a very large number.

Assumptions:

(1) Each stream requires one unit of link capacity.
(2) Each link has integral units of capacity.
(3) Node 0 is the source node and nodes $1, 2, \dots, n-1$ are switching nodes.
(4) The video server has bid information.

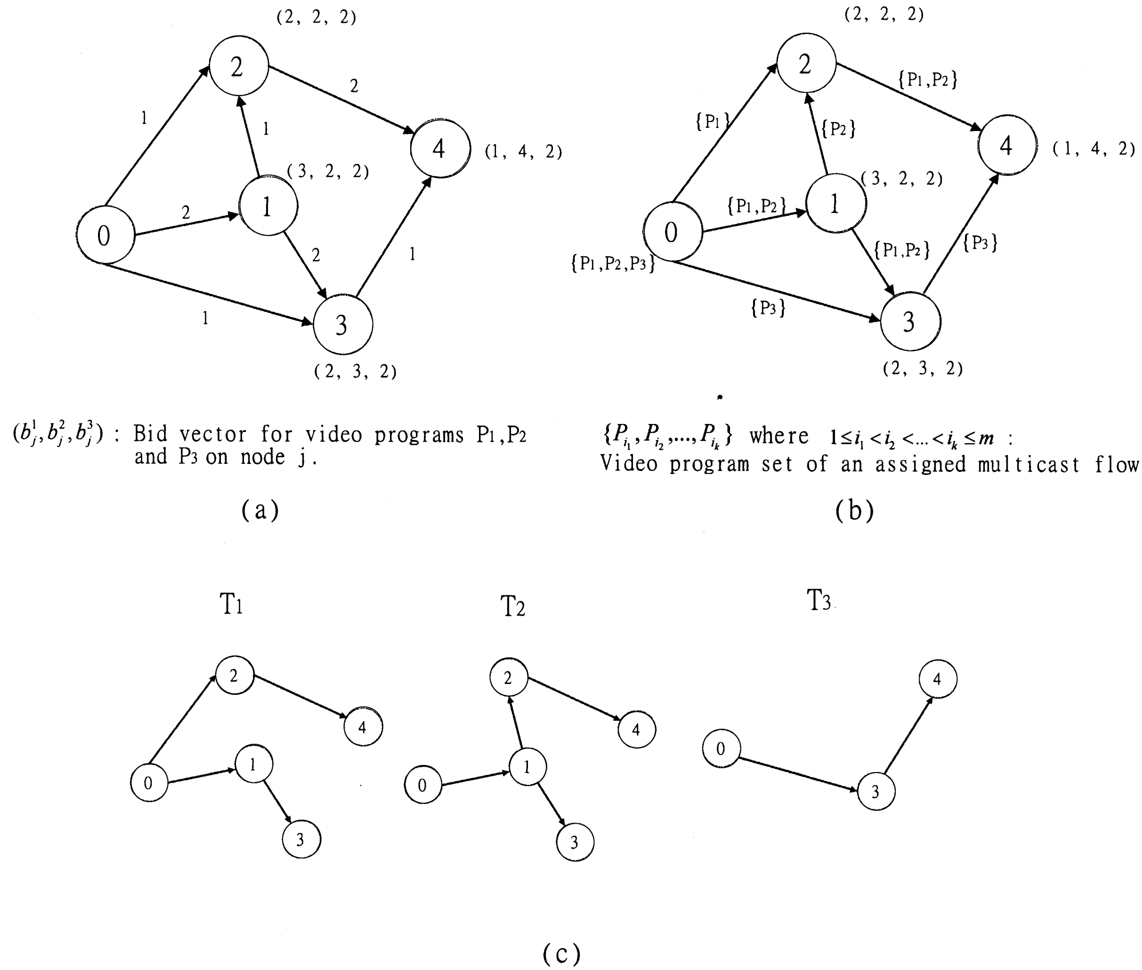

Fig. 3. An example for video distribution scheme.

The problem can be mathematically stated as follows.

Maximize $\sum_{j=1}^{n-1} \sum_{k=1}^{m} v_j^k b_j^k$

subject to

$$\sum_{k=1}^{m} x_{ij}^k \leqslant r_{ij}, \quad 0 \leqslant i, j \leqslant n-1, \tag{1}$$

$$\sum_{i=0}^{n-1} x_{i0}^k = 0, \quad 1 \leqslant k \leqslant m, \tag{2}$$

$$\sum_{i=0}^{n-1} x_{ij}^k = v_j^k, \quad 1 \leqslant j \leqslant n-1, 1 \leqslant k \leqslant m, \tag{3}$$

$$\sum_{s=0}^{n-1} x_{js}^k \leqslant M v_j^k, \quad 1 \leqslant j \leqslant n-1, 1 \leqslant k \leqslant m, \tag{4}$$

$$x_{ij}^k \in \{0, 1\} \quad 0 \leqslant i, j \leqslant n-1, 1 \leqslant k \leqslant m; \quad v_j^k \in \{0, 1\} \quad 0 \leqslant j \leqslant n-1, 1 \leqslant k \leqslant m. \tag{5}$$

Constraint (1) ensures that the multicast streams in each link do not exceed the bandwidth boundaries. Constraint (2) ensures that no stream can flowback to the server. The remaining constraints indicate that a video stream can flow out from a switch node only if it has received the video stream from its upstream neighbors. Furthermore, for each node, all video sets which were sent by its upstream neighbors are mutually exclusive.

3. A branch-and-bound algorithm for directed acyclic graphs

In order to solve the OSGMP, we use a tree, called a state-space tree to represent all of the feasible solutions, and apply the branch-and-bound algorithm on the state-space tree to search for the optimal solution.

Note that for a DAG there exists a topological order for its nodes. That is, we can label the nodes such that $i < j$ for every directed link (i, j). For example, Fig. 4 shows a topological order for a DAG. Based on the topological order, we can find all feasible solutions for each node. For example, in Fig. 5, links (1, 3) , (2, 3) are the only incoming links of node 3. Let node 1 can receive video programs P_1, P_2 and P_3. That is, node 1 has video program set $\{P_1, P_2, P_3\}$. Similarly, let node 2 have video program set $\{P_2, P_3\}$. The capacity of link (1, 3) is 2, it means two video programs can be chosen from set $\{P_1, P_2, P_3\}$ and send them to node 3. Similarly, we can choose one video program from set $\{P_2, P_3\}$ at node 2 and then send it to node 3. Therefore, all possible video program sets for node 3 are $\{P_1, P_2\}$, $\{P_1, P_2, P_3\}$, $\{P_1, P_3\}$, $\{P_2, P_3\}$. Note that we only keep dominating video program sets for node 3. For example, $\{P_1, P_2\}$, $\{P_1, P_3\}$, $\{P_2, P_3\}$ are dominated by $\{P_1, P_2, P_3\}$ at node 3 because $\{P_1, P_2, P_3\}$ for node 3 has a better gain than others. Thus, we keep $\{P_1, P_2, P_3\}$ as a dominating video program set at node 3. In the following, we will present how to generate dominating video set for each node. Let

$$v_j^k = \begin{cases} 1, & \text{video program } P_k \text{ can be received at switching node } j, \\ 0, & \text{otherwise.} \end{cases}$$

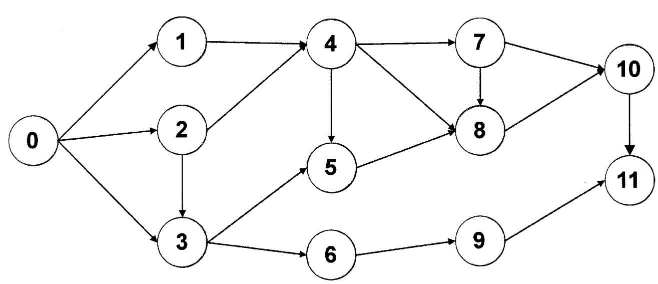

Fig. 4. Topological order of a DAG.

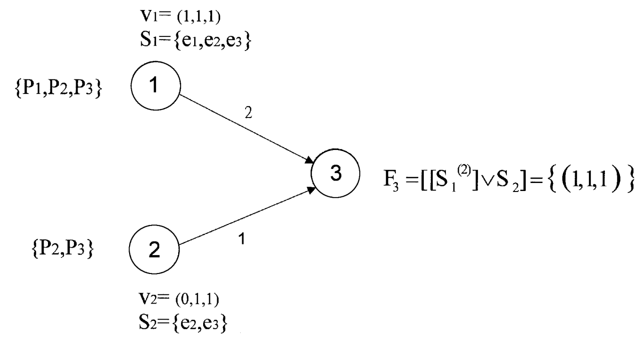

Fig. 5. Example of computing the feasible video received set of a node.

Let vector $\mathbf{v}_j = (v_j^1, v_j^2, \ldots, v_j^m)$ denote the video vector received at the switching node j. The *or* operation for vector $\mathbf{v} = (v_1, v_2, \ldots, v_m)$ and $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ is defined as

$$\mathbf{v} \vee \mathbf{u} = (v_1 \vee u_1, v_2 \vee u_2, \ldots, v_m \vee u_m)$$

where $\vee$ *is logic or* operation. Let $\mathbf{e}_j$ be a unit vector with all entries at zero except entry j, which is a one. Thus, $\mathbf{v} = \sum_{i=1}^{m} v_i \mathbf{e}_i$. Let S_x be the received video set for $\mathbf{v}_x$ at node x, such that if $v_x^j = 1$ then $\mathbf{e}_j$ is an element in S_x. For example, in Fig. 5, node 2 has video vector $\mathbf{v}_2 = (0, 1, 1)$, then $\mathbf{v}_2 = \mathbf{e}_2 + \mathbf{e}_3$ and $S_2 = \{\mathbf{e}_2, \mathbf{e}_3\}$. Suppose that link capacities of (x, z) and (y, z) are 1, then the set that includes all possible video vectors received at node z is defined as

$$S_x \vee S_y = \{\mathbf{e}_i \vee \mathbf{e}_j \mid \mathbf{e}_i \in S_x, \mathbf{e}_j \in S_y\}.$$

For example, $S_x = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ and $S_y = \{\mathbf{e}_2, \mathbf{e}_3\}$, then the set that includes all possible video vectors received at node z are $S_x \vee S_y = \{(1,1,0), (1,0,1), (0,1,0), (0,1,1), (0,1,1), (0,0,1)\}$. Let $S^{(2)}$ denote $S \vee S$ and $S^{(i)} = S^{(i-1)} \vee S$. For any node j, assume that links $(i_1, j), (i_2, j), \ldots, (i_a, j)$ are the incoming links for node j and node i_s $(s = 1, 2, \ldots, a)$ has video vector received $(v_{i_s}^1, v_{i_s}^2, \ldots, v_{i_s}^m)$. Let F_j be the set that includes all of the possible received video vectors for node j. Thus,

$$F_j = S_{i_1}^{(r_{i_1 j})} \vee S_{i_2}^{(r_{i_2 j})} \vee \cdots \vee S_{i_a}^{(r_{i_a j})},$$

where $r_{i_s j}$ is the link capacity of (i_s, j), $s = 1, 2, \ldots, a$.

We say a vector $\mathbf{v}$ is dominated by vector $\mathbf{u}$ if and only if $\mathbf{v} \vee \mathbf{u} = \mathbf{u}$. For example, (0, 1, 1) is dominated by (1, 1, 1) because $(0,1,1) \vee (1,1,1) = (1,1,1)$. Let $[F]$ denote the set of vectors in which all the dominated vectors in F are deleted. That is, any two vectors in $[F]$ are not dominated by one another. Fig. 5 shows an example for how to compute all possible video vectors received for a node. The video vectors received at nodes 1 and 2 are (1, 1, 1) and (0, 1, 1), respectively. Then, set F_3 can be generated by computing the result of $[[S_1^{(2)}] \vee S_2]$. The possible video sets received, S_1 and S_2 with respect to F_1 and F_2 are $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ and $\{\mathbf{e}_2, \mathbf{e}_3\}$, respectively. Therefore, $S_1^{(2)} = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} \vee \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = \{(1,0,0), (1,1,0), (1,0,1), (1,1,0), (0,1,0), (0,1,1), (1,0,1), (0,1,1), (0,0,1)\}$, and $[S_1^{(2)}] = \{(1,1,0), (1,0,1), (0,1,1)\}$ by deleting the dominated vectors from set $S_1^{(2)}$. In the final computation, $[[S_1^{(2)}] \vee S_2] = [\{(1,1,0), (1,0,1), (0,1,1)\} \vee \{(0,1,0), (0,0,1)\}] = \{(1,1,1)\}$. This means that node 3 can receive video programs P_1, P_2 and P_3 from its incoming links.

The state-space tree is generated from the network G. The node for the state-space tree is labeled by a received video vector $(v_j^1, v_j^2, \ldots, v_j^m)$, which specifies that switching node j can receive video programs P_k whenever $v_j^k = 1$, $k = 1, 2, \ldots, m$. The path from the root to a leaf node in the state-space tree will be defined to represent a sequence of possible video vectors, i.e., a sequence of received video vectors $(v_0^1, v_0^2, \ldots, v_0^m), (v_1^1, v_1^2, \ldots, v_1^m), \ldots, (v_{n-1}^1, v_{n-1}^2, \ldots, v_{n-1}^m)$. Note that the feasible solution $x_{ij}^k, \forall i, j, k$ can be found easily from the sequence of received video vectors.

For example, Fig. 7 is a partial state-space tree generated from Fig. 6. As shown in Fig. 6, node 0 is the server which provides three video programs P_1, P_2, and P_3, hence the received video vector for the root x_0 in the state-space tree is (1, 1, 1) (i.e., $F_0 = \{(1, 1, 1)\}$). For switching node 1, because link from node 0 to node 1 is the only incoming link and the boundary capacity for this link is 2, hence, the possible video vectors that can be received by node 1 can be derived using $F_1 = [S_0^{(2)}] = \{(1, 1, 0), (1, 0, 1), (0, 1, 1)\}$. Thus, the second level for the state-space tree is generated and labeled using (1, 1, 0), (1, 0, 1) and (0, 1, 1). Similarly, for switching node 2, because link from node 0 to node 2 and link from node 1 to node 2 are the only incoming links and $r_{02} = 1$ and $r_{12} = 1$, the children of node 1 in state-space tree can be determined by $[\{\mathbf{e}_1, \mathbf{e}_2\} \vee \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}] = \{(0, 1, 1), (1, 0, 1), (1, 1, 0)\}$. Therefore, we labeled these nodes using (0, 1, 1), (1, 0, 1) and (1, 1, 0). By continuing the same process, the entire state-space tree for this OSGMP can be obtained.

In fact, to find an optimal solution, we will not consider all of the feasible sequences since it is very time consuming. We will apply a best-first branch-and-bound algorithm to find the optimal

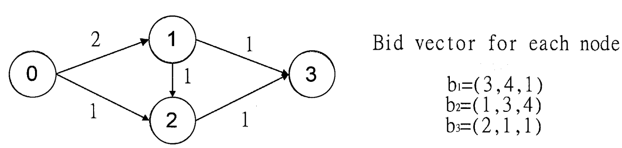

Fig. 6. Example of a OSGMP.

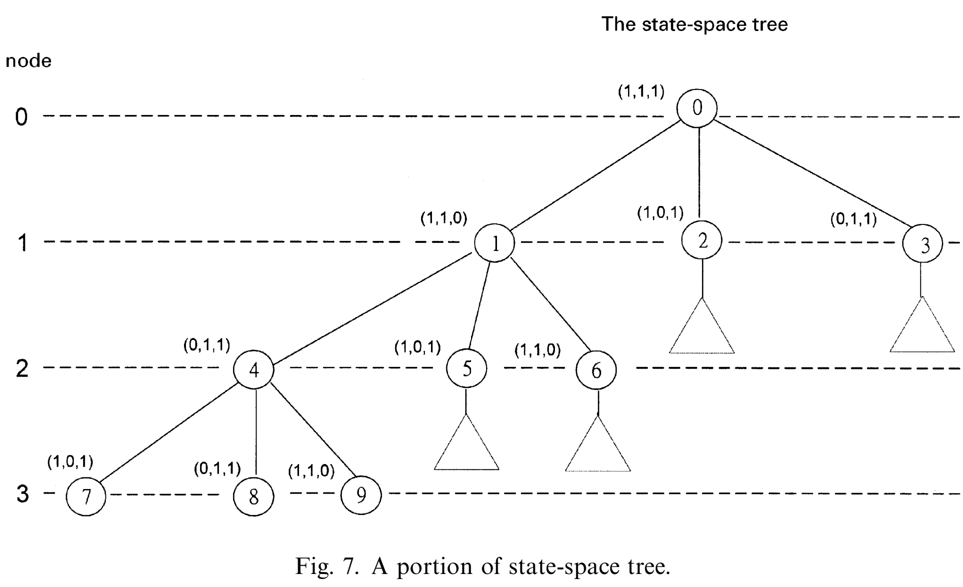

Fig. 7. A portion of state-space tree.

solution by traversing only a portion of the state-space tree. The branch-and-bound method is basically characterized by two decision rules. One provides the method to estimate the upper bound of an objective function at every node of the state-space tree. The other specifies a choice criterion for the selection of a node for the next branch.

3.1. The estimation of the upper bound for the objective function at node x

Assume that the current by visited node is x_i in the state-space tree. Then the partial sequence for the received video vectors can be defined by the path A_{x_i} from the root x_0 to node x_i is $(v_0^1, v_0^2, \ldots, v_0^m), (v_1^1, v_1^2, \ldots, v_1^m), \ldots, (v_i^1, v_i^2, \ldots, v_i^m)$. That is, there are i switching nodes being assigned to received video vectors. The current value of the objective function at node x_i is:

$$f(A_{x_i}) = \sum_{j=1}^{i} \sum_{k=1}^{m} b_j^k v_j^k.$$

Note that there are remaining $n-(i+1)$ switching nodes unvisited. Let switching node j be a node in G with bid vector $(b_j^1, b_j^2, \ldots, b_j^m)$ and incoming links $(i_1, j), (i_2, j), \ldots, (i_a, j)$. We sort the bids at node j such that $b_j^{i_1} \geqslant b_j^{i_2} \geqslant \cdots \geqslant b_j^{i_m}$. An upper bound ub_j for gain contributed from node j can be estimated by $ub_j = \sum_{k=1}^{h} b_j^{i_k}$ where $h = \sum_{s=1}^{a} r_{i_s j}$. The value $UB_i = f(A_{x_i}) + \sum_{j=i+1}^{n-1} ub_j$ is an upper bound for the objective function value in the complete assignment generated from partial assignment $(v_1^1, v_1^2, \ldots, v_1^m), (v_2^1, v_2^2, \ldots, v_2^m), \ldots, (v_i^1, v_i^2, \ldots, v_i^m)$.

3.2. The selection process in a branching node.

To facilitate the generation of the state-space tree, a data structure heap called live-node heap is used to record all live nodes that are waiting to be branched. The search strategy of the proposed branch-and-bound algorithm is a best-first search. That is, the node, say node x, selected for the next branch is the live node whose UB_x value is the largest among all of the nodes in the live-node heap. Note that the maximal value is at the top of the heap. Traversing the state-space tree begins from the root and stops when the live-node heap is empty. In addition, a current maximal source gain (G_{max}) is associated with the branch-and-bound algorithm. Initially, G_{max} is set at 0 and updated to be $G_{max} = \max(G_{max}, f(A_y))$ whenever a leaf node y is reached. If a node x satisfies $UB_x \leqslant G_{max}$, then node x is bounded since further branching from x will not lead to a better solution. When the live-node heap becomes empty, we obtain the optimal solution $(v_1^1, v_1^2, \ldots, v_1^m)$, $(v_2^1, v_2^2, \ldots, v_2^m), \ldots, (v_{n-1}^1, v_{n-1}^2, \ldots, v_{n-1}^m)$ with optimal value $\sum_{j=1}^{n-1} \sum_{k=1}^{m} b_j^k v_j^k = G_{max}$. The detailed algorithm is presented in Fig. 8, and the method is introduced using a recursive version.

A simple numerical example is given in Fig. 9. In Fig. 9a, each switching node is associated with a bid vector and each link is associated with a capacity. Fig. 9b shows the generation of the state-space tree. Initially, set the current maximal source gain (G_{max}) to be 0 and the root node x_0 of the state-space tree is (1, 1, 1) (i.e., $S_0 = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$). The upper bound for gain contributed from switching node 1 is $ub_1 = 7$. This is because $b_1^2 > b_1^3 > b_1^1$ and $r_{01} = 2$. Then, $ub_1 = b_1^2 + b_1^3 = 7$. Similarly, we have $ub_2 = 7$ and $ub_3 = 3$. Therefore, the possible received video vectors set for switching node 1 can be computed and obtained using $[S_0^{(2)}] = \{(0, 1, 1), (1, 0, 1), (1, 1, 0)\}$. This generates nodes 1, 2 and 3 of the state-space tree.

```
algorithm Optimal-stream-distribution;
begin
    set G_max : = 0;
    set live-node heap : = {v_0};
    while live-node heap ≠ ∅ do
    begin {generate the state-space tree}
        v : = remove-top(live-node heap);
        compute f(A_v) and UB_v;
        if G_max < UB_v then
        begin
            if v is a leaf node in state-space tree then G_max : = f(A_v);
            else generate all branching nodes from v, and
                add them to live-node heap;
        end;
    end;
end;
```

Fig. 8. Algorithm of optimal-stream distribution.

Now, we estimate the upper bound for the objection function (UB) at each node. We observed that at node 1, $UB_1 = (0, 1, 1)\cdot(1, 4, 3) + ub_2 + ub_3 = 17$ was the biggest value among UB_1, UB_2 and UB_3. Therefore, we selected node 1 to be our next branching node. Continuing the same process, we can reach the first leaf node 7 and update the current maximal source gain (G_{max}) to be 17. Finally, G_{max} bounds node 4 because $(0, 1, 1)\cdot(1, 4, 3) + (0, 1, 1)\cdot(4, 3, 1) + ub_3 = 14 < 17$. Similarly, nodes 5, 2 and 3 are bounded by G_{max}. Hence, no more nodes are waiting for branching and the algorithm terminates.

4. A DAG's-based heuristic algorithm for general graphs

In order to solve the OSGMP for a general graph case, we propose a two phased algorithm to find an approximate solution. At the first phase, we properly choose a directed acyclic subgraph from the given general graph and then apply the branch-and-bound algorithm to find an initial solution. In the second phase, the residual capacities for links are considered so that the source gain of the initial solution will be increased. To achieve this goal, for each residual link (i, j), we can select a set of video programs which appear on node i but not on node j, and send them along the link until the capacities are exhausted. The details are given as follows.

4.1. Phase I: choose a directed acyclic subgraph from a general graph

We can choose a directed acyclic subgraph from a general graph by labeling the node order. Based on the node order, we can obtained a DAG by removing all links (i, j) if $i > j$. Different node orders produce different subgraphs. A *good* directed acyclic subgraph is a subgraph with greater source gain at the first phase. A subroutine **Find-node-order** is proposed to determine a node order for a general graph such that a *good* directed acyclic subgraph can be produced. Subroutine **Find-node-order** gives each node with a value called **gain-lost** to estimate the maximum source gain that may be lost in the first phase when a node order is given.

Initially, we set the **gain-lost** for each node to be a big number, say 999 except for the source node. The **gain-lost** for the source node is set to be 0. For each iteration, **Find-node-order** will choose a node whose number of incoming links (in-degree) is zero and then label it. If no zero in-degree

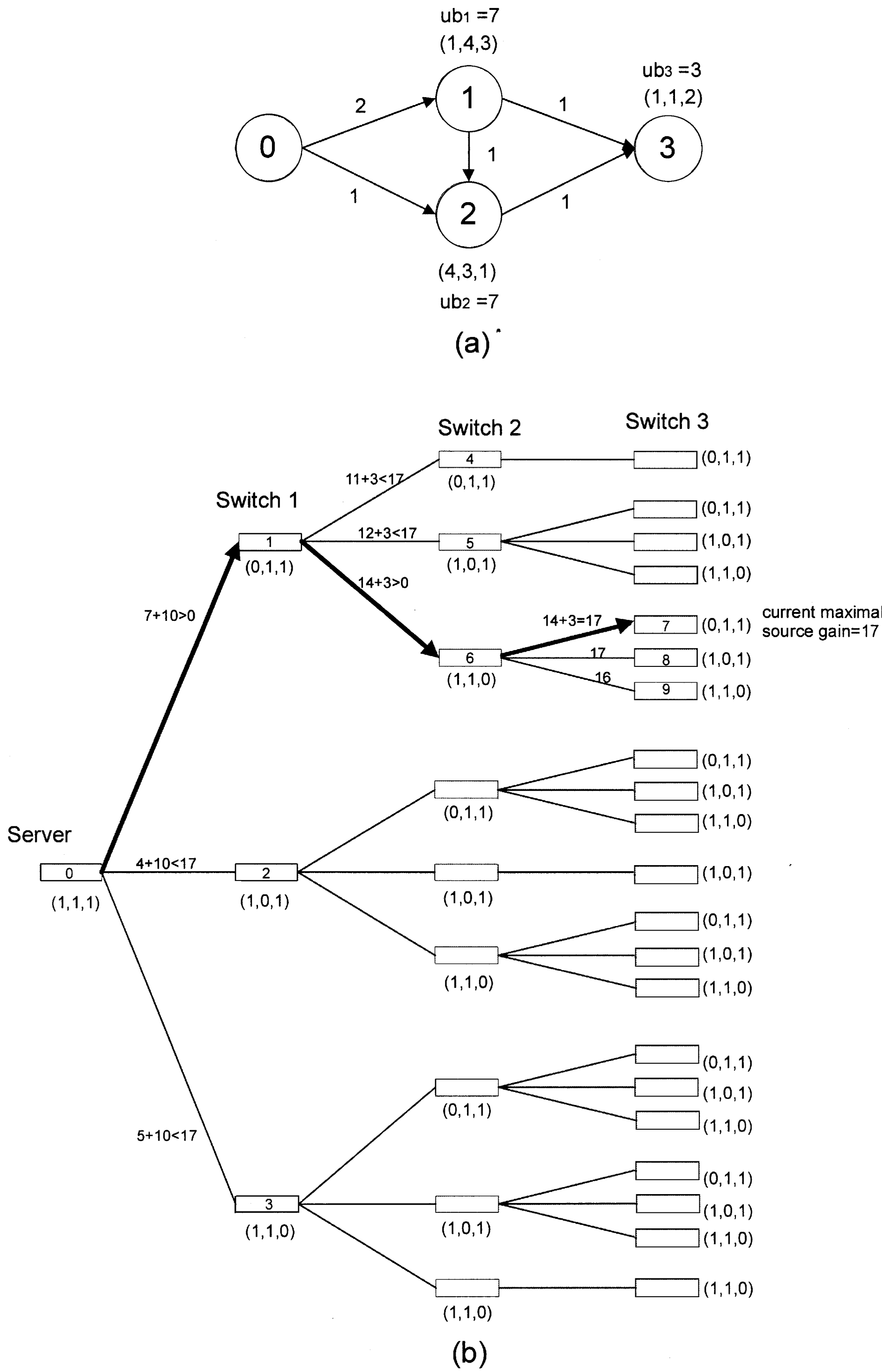

Fig. 9. A numerical example for branch-and-bound algorithm.

```
procedure Find-node-order;
begin
    initialize glost(s) : = 0;
    initialize glost(j) : = 999 for all j ∈ G − {s};
    order : = 0;
    while G ≠ ∅ do
    begin
        select a node j such that in-degree(j)=0;
        if no in-degree zero node was found then
            select a node j such that glost(j) is the smallest
              value among every other nodes in G;
        label(j) : = order;
        update bid vector b_k for every descendant k of j;
        G := G − {j};
        recompute glost(k) for every descendant k of j;
        order : = order+1;
    end
end;
```

Fig. 10. Procedure of find-node order.

nodes are found, then the node with the smallest **gain-lost** value among the nodes in graph G will be selected and labeled. The chosen node is then deleted from graph G. Next, we update the bid vectors of its outgoing neighbors. Let $(j, i_1), (j, i_2), \ldots (j, i_a)$ be nodes j's outgoing links and $(b^1_{i_1}, b^2_{i_1}, \ldots, b^m_{i_1}), (b^1_{i_2}, b^2_{i_2}, \ldots, b^m_{i_2}), \ldots, (b^1_{i_a}, b^2_{i_a}, \ldots, b^m_{i_a})$ be the bid vectors of node j's outgoing neighbors $i_1, i_2, \ldots i_a$, respectively. Note that, after link (j, i_k) is deleted, the largest gain that can be achieved is the sum of first r_{ji_k} biggest bids among bid vector $(b^1_{i_k}, b^2_{i_k}, \ldots, b^m_{i_k})$. We set the bid values of the first r_{ji_k} biggest bids to 0. Also, we re-estimate the **gain-lost** values for nodes i_k using the updated bid vector. A complete description of this subroutine is shown in Fig. 10.

For node j with bid vector $(b^1_j, b^2_j, \ldots, b^m_j)$ and incoming links $(i_1, j), (i_2, j), \cdots, (i_a, j)$, if we label node j as $l(j) < \min\{l(i_1), l(i_2), \ldots, l(i_a)\}$, then all incoming links for node j will not appear on the directed acyclic subgraph and the total capacities $r = r_{i_1 j} + r_{i_2 j} + \cdots + r_{i_a j}$ will not be used for the first phase. Therefore, we let the **gain-lost** for node j be the gain contributed from the total lost capacities r. That is, we sort the bids at node j such that $b^{i_1}_j \geqslant b^{i_2}_j \geqslant \cdots \geqslant b^{i_m}_j$. A **gain-lost,** denoted as $glost(j)$ for node j can be estimated by $glost(j) = \sum^r_{k=1} b^{i_k}_j$. Obviously, when in-degree for node j is 0 then the value of glost(j) should be set to 0.

Fig. 11 shows a numerical example for how to determine the node order for a given graph. At first, the algorithm initializes the **gain-lost** value for each node. At the first iteration, node i was depicted and labeled by 0, for in-degree(i) is equal to zero. Now, execute the update process to modify $\mathbf{b}_j$ and $\mathbf{b}_k$. For example, updates bid vectors $\mathbf{b}_j$. Because the capacity of link (i, j) is 2, therefore, we choose the first 2 biggest bids among $\mathbf{b}_j$ and reset them to 0. This secures a modified bid vector $\mathbf{b}_j = (0, 0, 4)$. Similarly, we can obtain the modified bid vector $\mathbf{b}_k = (1, 3, 0)$. Next, delete node i from graph G and then update glost(k) and glost(j). For updating the value for glost(k), (j, k) is the only incoming link of node k, hence, the total capacities r which will not be used for phase I is equal to 1. Now, we recompute glost(k) using the summation of the first r (= 1) biggest bids among $\mathbf{b}_k$, and obtain glost(k) = 3. Similarly, we can secure the value glost(j) = 4. Repeat the same iteration until graph G becomes empty. Hence, a node order is produced by this algorithm.

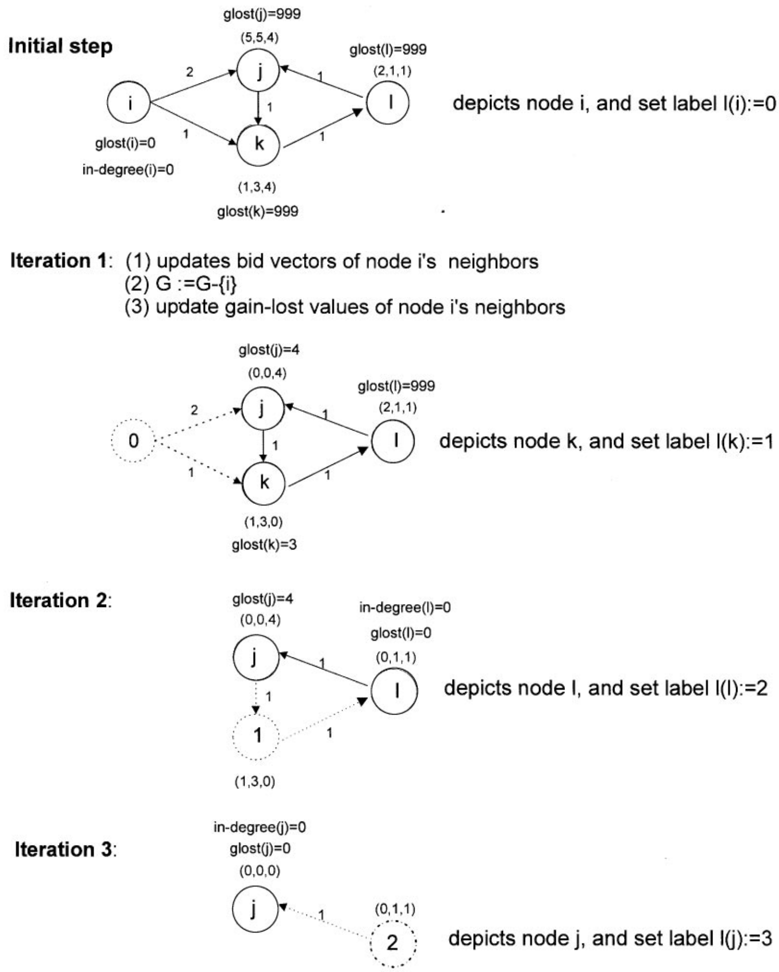

Fig. 11. Determine the nodes order of graph G.

Based on the node order, we can obtained a DAG by removing all links (i, j) if $i > j$. At the same time, we collect the removed links to form a subgraph called a residual subgraph. Note that, if the given graph is a DAG, obviously the residual subgraph will be empty. Now, we apply the branch-and-bound algorithm on the directed acyclic subgraph, and obtain an optimal received video vector for each node of the resulting DAG.

4.2. Phase II: video streams distribution on residual graph

In order to increase the source gain, for every link (i, j) in the residual subgraph, node i can send video streams to node j. Let link (i, j) have capacity r_{ij} in the residual subgraph, and $(b_j^1, b_j^2, \ldots, b_j^m)$ be the bid vector of node j. Let $(v_i^1, v_i^2, \ldots, v_i^m)$ and $(v_j^1, v_j^2, \ldots, v_j^m)$ be the received video vectors for node i and node j, respectively. A video can contribute to the source gain when it is already received

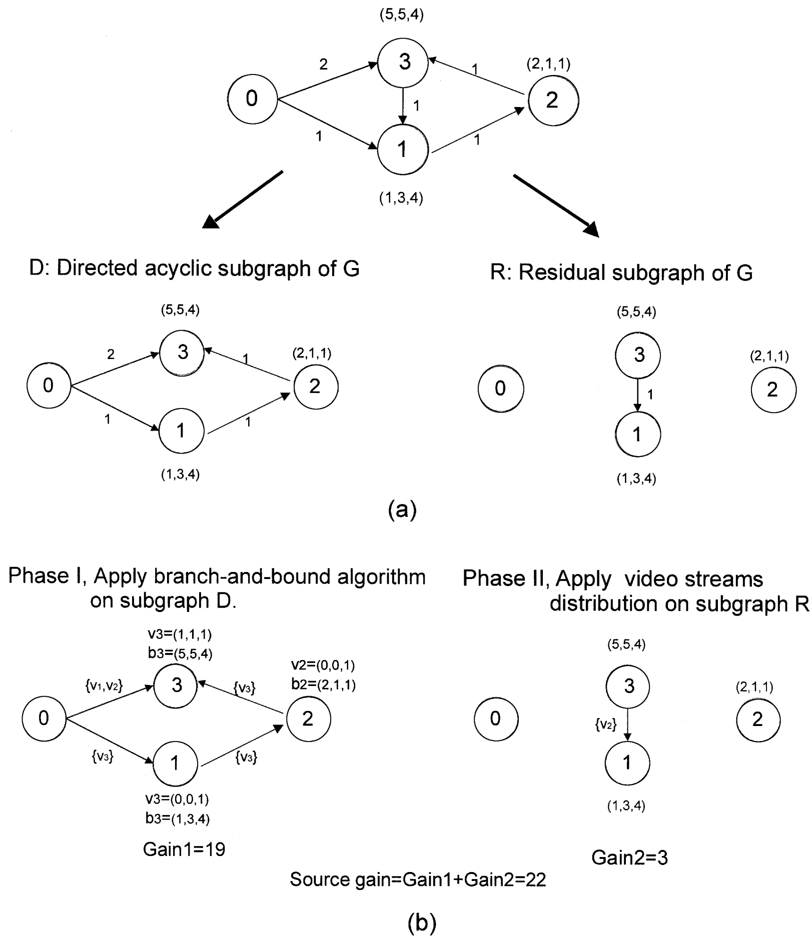

Fig. 12. General graph's streams distribution method. (a) graph partition using a given node order, (b) streams distribution policy which is found by heuristic algorithm.

at node i but not received at node j. That is, we want to choose a video q such that $v_i^q = 1$ and $v_j^q = 0$. However, link (i, j) only has capacity r_{ij}. Let $Q = \{q | v_i^q = 1 \wedge v_j^q = 0\}$. Thus, we select a subset $Q' \subseteq Q$, where $|Q'| = r_{ij}$, such that $\sum_{q \in Q'} b_j^q$ is maximized. This can be done by selecting the first r_{ij} biggest b_j^q, $q \in Q$.

Fig. 12a shows the graph partition by giving the node order which is obtained from Fig. 11. Fig. 12b shows the results from applying the two-phased heuristic algorithm. The two-phased heuristic algorithm for solving the OSGMP over a general graph is stated in Fig. 13.

5. Simulation results

In this section, the performance of the branch-and-bound algorithm for solving OSGMP over DAG and the two-phased heuristic algorithm for a general graph case are studied. The criterion we

algorithm. Two-phased heuristic;
begin
  apply subroutine **Find-nodes-order;**
  partition graph G into a subgraph pairs (D, R) where D is the
    directed acyclic subgraph, and R is the residual subgraph;
  apply DAG's Optimal-stream-distribution (D);
  apply video streams distribution on residual graph R;
end;

Fig. 13. Algorithm of two-phase heuristic.

adopted to evaluate the branch-and-bound algorithm was the traversing ratio of the state-space tree, which is defined as n_b/N, where n_b is the number of nodes that are traversed by the branch-and-bound algorithm, and N is the number of nodes in the complete state-space tree. On the other hand, the criterion we adopted to evaluate the two-phased algorithm was the gap ratio, which is defined as $1 - X_{heu.}/X_{opt.}$, where $X_{heu.}$ is the source gain of the two-phased heuristic algorithm, and $X_{opt.}$ is the optimal source gain.

The performance of these algorithms are influenced by the following five factors:

(1) number of nodes,
(2) the capacity constraint for each link,
(3) the bid vector with respect to each node,
(4) number of video programs available for customer subscription, and
(5) the shapes of the given graph.

Hence, the following assumptions were made about the experiment to address these factors.

(1) The capacity constraint for each link was randomly assigned to either 5 or 6.
(2) The bid of each video program at any switching node was randomly generated between intervals [0, 10].
(3) Number of video programs m, available for customers subscription $m = 8, 10, 12$ were considered.
(4) Twenty instances of bid vectors were run for each given graph, and the traversing ratio (gap ratio) was also computed when the branch-and-bound algorithm (two-phased heuristic algorithm) was applied.
(5) Five DAGs were depicted from randomly generated graphs, then run for the branch-and-bound algorithm. The number of nodes $n = 10, 12$ were considered. Fig. 14a shows these graph topologies.
(6) Five general graphs were depicted from randomly generated graphs, then run for the two-phased algorithm. The number of nodes $n = 10$ was considered. Fig. 14b shows these graph topologies.

The results of the average traversing ratios are listed in Table 1, with the number of nodes $n = 10, 12$ and number of video programs available for customers subscription $m = 8, 10$ and 12, respectively. On average, the traversing ratios were no more than 12% (3%) for 10-node (12-node) networks.

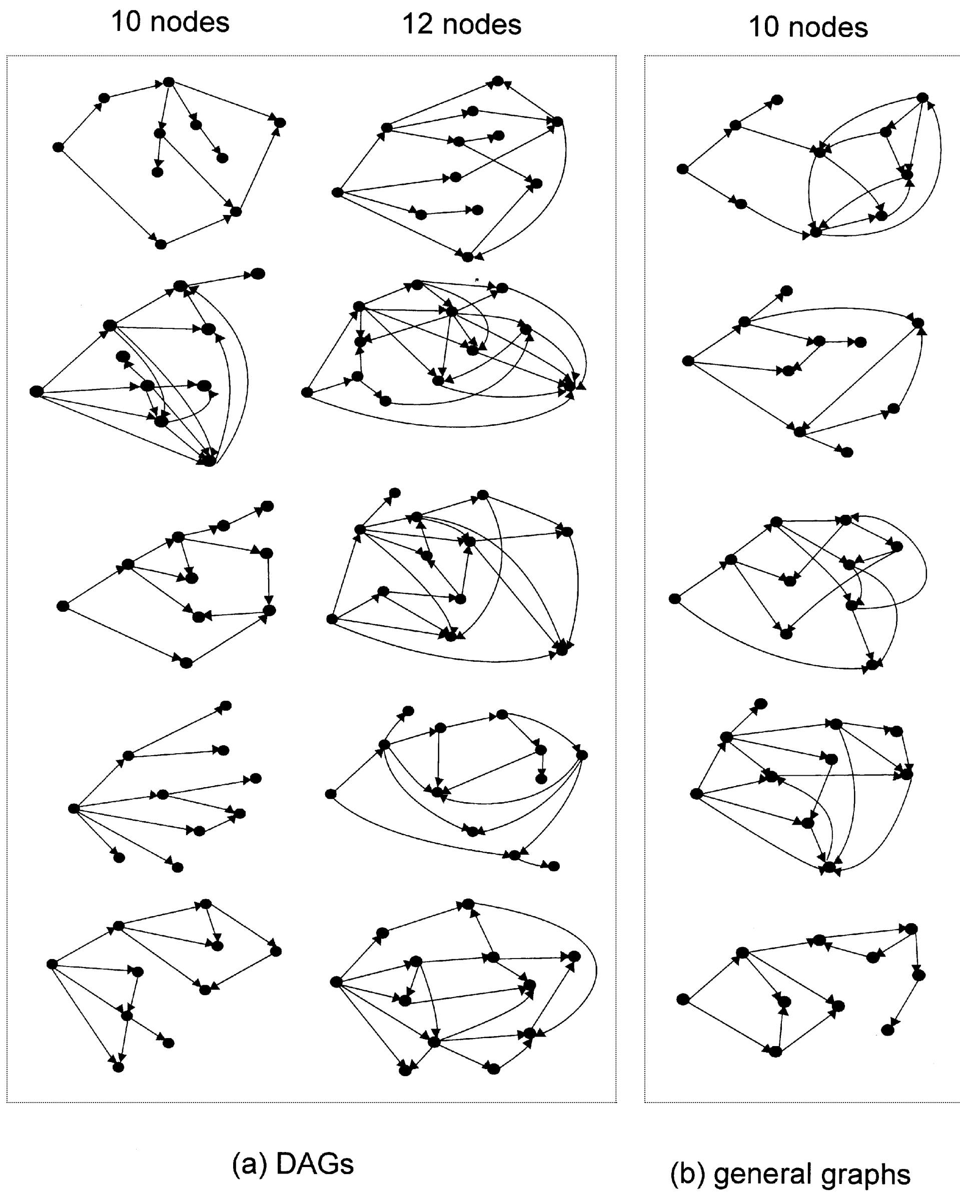

Fig. 14. Network topologies used in the performance evaluation. (a) DAGs run for branch-and-bound algorithm (b) General graphs run for two-phased algorithm.

The other experiment was done by applying the two-phased algorithm to five general graphs with $n = 10, m = 10$, and for each given graph, 20 instances of bid vectors were considered. The experimental results are listed in Table 2. It shows the source gains which were found using the two-phased heuristic algorithm are very close to the optimal source gains and most of the gap ratios were less than 0.07. Therefore, the proposed two-phase algorithm is very efficient.

Table 1
Variation in average traversing ratios for branch-and-bound algorithm

	$n = 10$ $m = 8$	$n = 10$ $m = 10$	$n = 10$ $m = 12$	$n = 12$ $m = 8$	$n = 12$ $m = 10$	$n = 12$ $m = 12$
n_b/N	11.4%	10.2%	10.8%	1.6%	2.6%	$\ll$2.1%

Table 2
Comparision of two-phase algorithm and optimal algorithm on source gain and computational time

Sample	Source gain			Computational time (s)[a]		Sample	Source gain			Computational time (s)[a]	
	$X_{heu.}$	$X_{opt.}$	Gap-ratio	$T_{heu.}$	$T_{opt.}$		$X_{heu.}$	$X_{opt.}$	Gap-ratio	$T_{heu.}$	$T_{opt.}$
1	218	226	0.035	6	78	51	235	237	0.008	9	364
2	214	220	0.027	14	118	52	253	259	0.023	12	254
3	203	210	0.033	1	645	53	221	225	0.018	10	178
4	210	218	0.037	116	4411	54	243	243	0.000	10	154
5	209	211	0.009	1	56	55	232	235	0.013	8	472
6	201	205	0.020	1	48	56	233	241	0.033	10	526
7	209	221	0.054	106	111	57	212	216	0.019	14	3121
8	218	221	0.014	1	598	58	238	247	0.036	10	797
9	210	221	0.050	200	2464	59	225	228	0.013	9	112
10	231	231	0.000	6	313	60	235	251	0.064	13	17
11	202	204	0.010	2	7351	61	217	218	0.005	1	4
12	243	249	0.024	1	74	62	216	217	0.005	1	3
13	215	222	0.032	42	2580	63	194	194	0.000	2	6
14	212	217	0.023	3	224	64	210	210	0.000	1	5
15	214	215	0.005	66	3387	65	202	202	0.000	2	11
16	220	230	0.043	43	104	66	192	195	0.015	1	1
17	191	201	0.050	61	3005	67	211	212	0.005	2	6
18	232	237	0.021	4	1412	68	215	216	0.005	1	4
19	199	203	0.020	26	380	69	216	216	0.000	1	5
20	228	232	0.017	7	345	70	214	214	0.000	1	12
21	188	188	0.000	1	28	71	195	195	0.000	1	4
22	188	188	0.000	1	16	72	241	243	0.008	1	1
23	166	173	0.340	19	32	73	206	209	0.014	1	2
24	186	186	0.000	1	15	74	211	214	0.014	3	9
25	192	192	0.000	1	23	75	212	212	0.000	1	3
26	172	174	0.011	20	23	76	224	224	0.000	1	4
27	178	178	0.000	1	22	77	188	191	0.016	2	4
28	192	197	0.025	1	27	78	231	231	0.000	1	6
29	198	198	0.000	1	15	79	202	202	0.000	3	9
30	190	190	0.000	1	18	80	226	226	0.000	1	5
31	172	185	0.070	1	28	81	185	185	0.000	10	141
32	209	217	0.037	1	22	82	183	183	0.000	13	229
33	188	188	0.000	8	31	83	172	172	0.000	17	213

Table 2. (*Continued*)

Sample	Source gain			Computational time (s)		Sample	Source gain			Computational time (s)	
	$X_{heu.}$	$X_{opt.}$	gap-ratio	$T_{heu.}$	$T_{opt.}$		$X_{heu.}$	$X_{opt.}$	gap-ratio	$T_{heu.}$	$T_{opt.}$
34	190	190	0.000	1	16	84	189	189	0.000	12	336
35	186	186	0.000	1	17	85	197	197	0.000	9	451
36	195	195	0.000	1	30	86	178	178	0.000	9	293
37	167	170	0.018	24	95	87	182	182	0.000	11	405
38	197	201	0.020	1	27	88	199	199	0.000	11	2148
39	178	178	0.000	1	18	89	184	184	0.000	23	1619
40	193	193	0.000	1	20	90	196	196	0.000	15	226
41	220	234	0.060	24	926	91	189	189	0.000	11	827
42	245	244	0.004	9	1246	92	197	197	0.000	10	1148
43	217	217	0.000	11	303	93	182	182	0.000	15	625
44	236	241	0.021	8	188	94	189	189	0.000	13	829
45	240	240	0.000	2	483	95	190	190	0.000	13	432
46	220	219	0.005	17	253	96	189	189	0.000	20	1425
47	231	231	0.000	10	1848	97	167	167	0.000	13	468
48	247	249	0.008	7	1006	98	201	201	0.000	14	223
49	229	241	0.050	10	137	99	179	179	0.000	13	125
50	256	256	0.000	5	156	100	188	188	0.000	14	50

[a]These samples were run on IBM PC with Pentium-PRO-S CPU of 200 MHz.

6. Concluding remarks

In this paper, we formally modelled the optimum source gain multicast problem as an integer programming problem. A branch-and-bound method was proposed to solve this problem for a DAG case. We presented a two-phased algorithm to find an approximate solution for a general graph case. From the computation results, it is shown that the objective function value of the approximation solution is very close to the optimal value. On the other hand, in our design issue of the two-phased heuristic algorithm, a *good* directed acyclic subgraph was determined by choosing a node order for the given graph to achieve higher gain in the first phase. We used **gain-lost** values as the criterion for labeling the node order. In the update process for each iteration, we updated the bid vector of the chosen node's outgoing neighbors by setting first r biggest bids to be 0, where r denoted the capacity for the outgoing link. This was based on assuming that the chosen node would receive every kind of video from the server node. However, this assumption is not accurate enough and it will cause an under-estimate of the **gain-lost** values. Therefore, a more accurate estimation of the **gain-lost** values to enhance the performance of the two-phase algorithm will be our future research.

The postoptimality analysis is also a possible future research. For example, it may happen that a customer decide to discontinue viewing a particular video and switches to some other video stream. One way to accomplish this is to solve the problem anew, but this may be computationally inefficient. If one makes use of the properties of the multicast tree solution, it is possible to reduce additional computations.

References

[1] Garey MR, Johnson DS. Computers and intractability, San Francisco: W.H. Freeman, 1979.
[2] Hakimi SL. Steiner's problem in graphs and its implications, Networks 1971;1:113–33.
[3] Levin A. Ju. Algorithm for the shortest connection of a group of graph vertices. Soviet Mathematical Doklady 1971;12:1477–81.
[4] Beasley JE. An algorithm for the Steiner problem in graphs. Networks 1984;14:147–59.
[5] Wong RT. A dual ascent approach for Steiner tree problems on a directed graph. Mathematical Programming, 1984;28:271–87.
[6] Shore ML, Foulds LR, Gibbons PB. An algorithm for the Steiner problem in graph. Networks, 1982;12:323–33.
[7] Wu YF, Widmayer P, Wong CK, A fast approximation algorithm for the Steiner problem in graphs. Acta Information, 1986;23:223–9.
[8] Takahashi H, Matsuyama A. An approximate solution for the Steiner problem in graphs. Mathematics of Japan 1980;24:573–7.
[9] Hwang FK, Richards DS. Steiner tree problems. Networks 1992;22:55–9.
[10] Winter P. Steiner problem in networks: a survey. Networks 1987;17:129–67.
[11] Noronha Jr. CA, Tobagi FA. Optimum routing of multicast streams. Proceedings of IEEE Infocom'94, 1994;2:865–73.
[12] Shacham N, Meditch JS. An algorithm for optimal multicast of multimedia streams. Proceedings of IEEE Infocom'94, 1994;2:856–63.
[13] Zhu Q, Parsa M, Garcia-Luna-Aceves JJ. A source-based algorithm for delay-constrained minimum-cost multicasting. Proceedings of IEEE Infocom'95, 1995;1:377–85.
[14] Kompella VP, Pasquale JC, Polyzos GC. Multicast for multimedia applications. Proceedings of IEEE Infocom'92, 1992;3:2078–85.