Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 182 (2007) 455–465

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

O.R. Applications

A Lagrangean relaxation and decomposition algorithm for the video placement and routing problem

Tolga Bektaş [a,b,*], Osman Oğuz [a], Iradj Ouveysi [c]

[a] *Department of Industrial Engineering, Bilkent University, 06800 Ankara, Turkey*
[b] *Center for Research on Transportation, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Que., Canada H3C 3J7*
[c] *Department of Electrical and Electronic Engineering, The University of Melbourne, VIC 3010, Australia*

Received 14 April 2005; accepted 18 July 2006
Available online 7 November 2006

Abstract

Video on demand (VoD) is a technology used to provide a number of programs to a number of users on request. In developing a VoD system, a fundamental problem is load balancing, which is further characterized by optimally placing videos to a number of predefined servers and routing the user program requests to available resources. In this paper, an exact solution algorithm is described to solve the video placement and routing problem. The algorithm is based on Lagrangean relaxation and decomposition. The novelty of the approach can be described as the use of integer programs to obtain feasible solutions in the algorithm. Computational experimentation reveals that for randomly generated problems with up to 100 nodes and 250 videos, the use of such integer programs help greatly in obtaining good quality solutions (typically within 5% of the optimal solution), even in the very early iterations of the algorithm.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Integer programming; Video on demand; Placement; Routing; Lagrangean relaxation; Decomposition

1. Introduction

Video on demand (VoD) is a service that provides tens to hundreds of videos (programs) to hundreds to thousands of clients through a network. In other words, a VoD service can be described as a "virtual video rental store" in which the users have the option to choose and watch any program on request, in the convenience of their homes. Commercial VoD services are now being offered throughout the world due to the fact that multimedia technologies are developing very fast. With such services, users can select any video programs they like, and then after a short setup time, receive the video programs through the network. As for videotape, they have greater flexibility in scheduling the viewing time and have fine-grained control: enabling them to pause, resume, fast rewind and fast-forward the video.

[*] Corresponding author.
E-mail address: tolga@crt.umontreal.ca (T. Bektaş).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.07.031

An in-depth treatment of the subject is given by Little and Venkatesh [9]. Here, we only provide a brief description. A complete VoD system consists of three fundamental components which may be stated as the storage server, the network on which the system is built and the user interface (e.g. the keyboard, mouse, or voice commands, along with a translator). The requests made by the user through the interface is forwarded to the network. Once the requested program is fetched from an available resource, it is served to the user.

A central problem in structuring a VoD system is load balancing, which can be further separated into two subproblems, as pointed out by Little and Venkatesh [9]. The first consists of deciding on program allocation and the second is resource location and connection establishment.

Although with the advances in hardware technology the storage costs are getting cheaper, the relevant cost still "makes it more difficult to build and operate a service that can successfully compete with the local video hire store" [2] for many companies. A VoD application is not simply restoring data in a hard disk, and it needs much more advanced and complicated technology to provide any kind of a fast-forward, rewind and pause flexibility to the user. The functionality that is expected from a VoD server is beyond that of a personal computer (PC), hence VoD providers need special equipment capable of performing such tasks. Additional functionalities, coupled with basic overhead costs (such as maintenance and cooling), makes the hardware associated to this technology quite expensive (see [11]). Notice that as shown in Wu et al. [15], Barnett and Anido [1], and Liu et al. [10] the cost of a video server depends mainly on the server throughput, which is defined as the number of simultaneous video streams which the video server can support for the guaranteed quality of service. The number of simultaneous streams multiplied by the bandwidth requirement of each stream will give an approximate value for the server throughput.

Several studies exist that address the problem of developing a VoD system. To mention a few, Kim et al. [7] consider designing a VoD system on a network with storage capacity constraints on each node and no capacity limitations on the links that are used to connect each pair of nodes. They present an integer linear programming formulation of the problem and describe a tabu search algorithm for its solution. The authors present computational results for networks with up to 40 nodes and 200 programs. Wang et al. [14] study the optimal video distribution problem in VoD systems with multiple multicast sessions. Multicasting is performed when a set of clients require the same program at approximately the same time. In this case, clients are grouped as a multicast tree through which the server sends the program. The authors present a branch and bound algorithm to find the optimal solution of the problem when the network is a directed acyclic graph and propose an approximation algorithm for general graphs. Hwang and Chi [6] consider the problem of placing a number of programs on a number of servers such that the total installation cost that is composed of the network transmission cost and the video storage cost is minimized. Leung and Wong [8] address a different aspect of the problem which consists of determining what kind of a charging scheme a service provider should adopt in order to maximize the mean revenue. Ouveysi et al. [12] proposed an integer programming formulation to determine the location of the programs in a VoD network that is subject to storage and transmission capacity constraints, so as to minimize the total cost of storage and transmission. They refer to this problem as the video placement and routing problem (VPRP) and describe heuristic approaches for its solution. Finally, Huang and Fang [5] propose a dynamic load balancing algorithms among the servers in a multi-server VoD system. Through simulations, the authors demonstrate that their algorithms perform well on an example network.

The main motivation in this paper is to develop an exact solution algorithm for the integer linear programming formulation of the VPRP that is introduced by Ouveysi et al. [12]. The algorithm is based on Lagrangean relaxation and decomposition, coupled with some integer programming techniques to convert infeasible solutions to feasible solutions. Our approach differs from similar algorithms in the literature in which mainly heuristics are utilized for this purpose. Computational experimentation reveals that, although at the expense of relatively higher solution times, the use of such integer programs help greatly in obtaining good quality solutions even in the very early iterations of the algorithm.

The format of our paper is as follows. In the next section, we state a formal definition of the problem and present the integer linear programming formulation. Section 3 provides the full details of the Lagrangean relaxation and decomposition algorithm. Results of computational experiments with the proposed solution approach are given in Section 4. Conclusions are stated in Section 5.

2. Problem definition and formulation

The problem considered here is formally defined as follows. There exists a fully meshed network modelled by an undirected graph $G = (V, A)$, where $V = \{1, 2, \dots, n\}$ is the set of nodes and A is the set of edges consisting of the $n(n-1)/2$ links of the network. Each link $\{i,j\} \in A$ has a transmission capacity that is denoted by S_{ij}. The set of programs (videos) is denoted by $P = \{1, 2, \dots, m\}$, where each program $k \in P$ has a capacity requirement that is denoted by m_k and a bandwidth requirement for transmission that is denoted by μ_k. Each node $j \in V$ corresponds to a potential location for storing the programs (i.e., the video server) with capacity denoted by S_j. In addition, each node has a unit demand for each program. The cost of storing a program $k \in P$ at node $j \in V$ is denoted by c_j^k and the cost of transmitting program $k \in P$ over link $\{i,j\} \in A$ is denoted by c_{ij}^k. A fully meshed VoD architecture with five servers is given in Fig. 1, where VS(i) denotes the ith video server.

Each user of the system is connected to a single video server. If a user requested program is not found at the corresponding server, the user can watch the program transparently from other servers in the system, however, at the expense of additional cost.

Given the demand forecast of the programs, the problem considered in this paper consists of finding a placement scheme for the programs (*video placement*) and deciding on which video server will address the demand of a specific user (*routing*). The overall aim is to minimize total storage and transmission cost of the system such that the demand of each node for each program is satisfied. We hereafter refer to this problem as the *video placement and routing problem* (*VPRP*).

Ouveysi et al. [12] have proposed an integer linear programming formulation for the VPRP, using the following binary decision variables:

$$x_{ij}^k = \begin{cases} 1 & \text{if program } k \in P \text{ is transmitted to node } j \in V \text{ from node } i \in V, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_j^k = \begin{cases} 1 & \text{if program } k \in P \text{ is stored at node } j \in V, \\ 0 & \text{otherwise.} \end{cases}$$

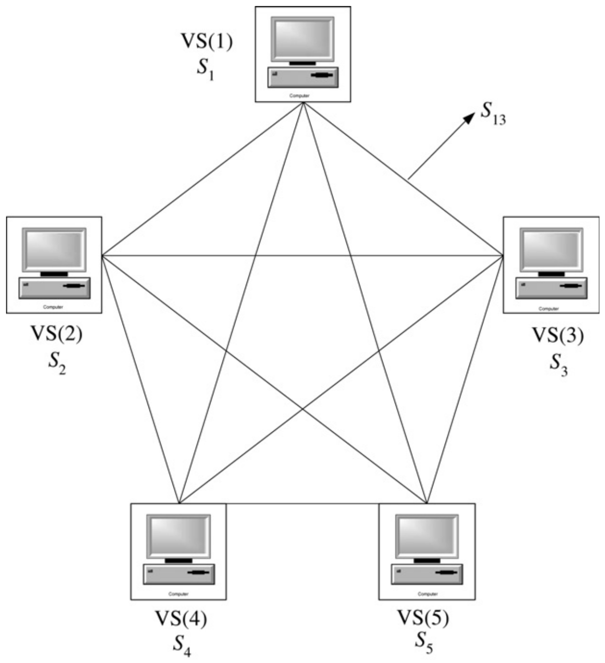

Fig. 1. A fully meshed VoD architecture with five servers.

The formulation is given as follows (denoted by **F**):

$$(\mathbf{F}) \quad \text{minimize} \quad \sum_{k\in P}\sum_{j\in V} c_j^k y_j^k + \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j} c_{ij}^k x_{ij}^k \tag{1}$$

$$\text{subject to} \quad \sum_{k\in P} m_k y_j^k \leqslant S_j \quad \forall j\in V, \tag{2}$$

$$\sum_{k\in P} \mu_k x_{ij}^k \leqslant S_{ij} \quad \forall i\neq j\in V, \tag{3}$$

$$\sum_{i\in V, i\neq j} x_{ij}^k + y_j^k = 1 \quad \forall j\in V,\ k\in P, \tag{4}$$

$$x_{ij}^k \leqslant y_i^k \quad \forall i\neq j\in V,\ k\in P, \tag{5}$$

$$x_{ij}^k, y_j^k \in \{0,1\} \quad \forall i\neq j\in V,\ k\in P. \tag{6}$$

In this model, constraints (2) and (3) correspond to the capacity constraints related to the storage nodes and transmission links, respectively. Constraints (4) state that each node either stores a program or receives it from another node that stores it. Finally, constraints (5) imply that a program can be transmitted from a node only if the program is stored at that node. Constraints (6) impose integrality restrictions on the decision variables. Problem **F** has in general $(n^2m + nm)/2$ binary variables and $(n^2m + n^2 + n)/2$ constraints.

It is clear that **F** only allows one-hop paths in transmitting a program. Ouveysi et al. [13] have studied the generalization of this problem to two-hop paths. However, as one inevitably needs to figure out the working paths between all pair of nodes in determining a routing scheme, it would be redundant to consider 2, 3 or 4 hop paths. Therefore, the approach taken here can be regarded as a more logical one compared to that proposed in [13].

We now study the complexity of the VPRP. Ouveysi et al. [12] mention the possibility that the VPRP may be $\mathcal{NP}$-hard. In the proposition stated below, we prove that it indeed is.

Proposition 1. *The video placement and routing problem (VPRP) is $\mathcal{NP}$-hard.*

Proof. The proof is based on the following restriction. Consider a special case of the problem with $P = \{1\}$, and let $\mu_1 \leqslant \min_{\{i,j\}\in A}\{S_{ij}\}$ and $m_1 \leqslant \min_{i\in V}\{S_i\}$. Since there is a single program, we can drop the index k in the formulation. In this case, constraints (2) and (3), pertaining to node and link capacities, become redundant. Now, partition the node set such that $V = I \cup J$ where $y_j \leqslant 0$ for all $j \in J$. Then, constraints (4) and (5) can be written as $\sum_{i\in I} x_{ij} = 1$, $\forall j \in J$ and $x_{ij} \leqslant y_i$, $\forall i \in I, j \in J$, respectively. But then **F** reduces to the well-known uncapacitated facility location problem (see [3]) with I as the set of potential facility locations and J as the set of customers. Since this problem is known to be $\mathcal{NP}$-hard, the VPRP is also $\mathcal{NP}$-hard. □

The complexity of the VPRP implies that the solution of **F** using standard off-the-shelf software will not be practical, especially with the increasing size of the problem. In what follows, we describe an exact solution algorithm for problem **F** that is based on Lagrangean relaxation and partitions the problem into smaller problems that are easier to solve.

3. A Lagrangean relaxation and decomposition algorithm

The algorithm is based on relaxing the capacity constraints (2) and (3) in a Lagrangean fashion, by associating the Lagrange multipliers β_j and α_{ij}, respectively. As a result, we obtain the following relaxed problem (denoted by $F(\beta, \alpha)$):

$$(F(\beta,\alpha)) \quad \text{minimize} \quad \sum_{k\in P}\sum_{j\in V}(c_j^k + \beta_j m_k) y_j^k + \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j}(c_{ij}^k + \alpha_{ij}\mu_k) x_{ij}^k - C_0$$

$$\text{subject to} \quad (4)\text{–}(6),$$

where $C_0 = \sum_{j\in V}\beta_j S_j + \sum_{i\in V}\sum_{j\in V}\alpha_{ij}S_{ij}$. Next, we observe that $F(\beta, \alpha)$ decomposes into $|P|$ subproblems, one for each program $k \in P$, where each one is denoted by $F_k(\beta, \alpha)$ and is shown for a specific program k^* as follows:

$$
\begin{aligned}
(F_{k^*}(\beta,\alpha)) \quad \text{minimize} \quad & \sum_{j\in V}(c_j^{k^*}+\beta_j m_k)y_j^{k^*}+\sum_{j\in V}\sum_{i\in V,i\neq j}(c_{i,j}^{k^*}+\alpha_{ij}\mu_{k^*})x_{ij}^{k^*}\\
\text{subject to} \quad & \sum_{i\in V,i\neq j}x_{ij}^{k^*}+y_j^{k^*}=1 \quad \forall j\in V,\\
& x_{ij}^{k^*}\leqslant y_i^{k^*} \quad \forall i\neq j\in V,\\
& x_{ij}^{k^*},y_j^{k^*}\in\{0,1\} \quad \forall i\neq j\in V.
\end{aligned}
$$

Each subproblem has $(n^2+n)/2$ binary variables and $(n^2+n)/2$ constraints. Let $v(F)$ denote the optimal objective function value of problem F. Then, as a result of the decomposition procedure, the optimal objective function of $F(\beta,\alpha)$ can be calculated as $v(F(\beta,\alpha))=\sum_{k\in P}v(F_k(\beta,\alpha))-C_0$. We are now ready to provide a general outline of the algorithm for the solution to problem **F**. This algorithm is based on the traditional subgradient optimization scheme (see [4]).

3.1. The solution algorithm

- Start with an initial vector of multipliers β^1, α^1. Let the incumbent lower bound be $\text{lb}=-\infty$, incumbent upper bound be $\text{ub}=\infty$ and $t=1$.
- Perform the following until $gap=\frac{\text{ub}-\text{lb}}{\text{ub}}<1.00$ or the maximum amount of iterations have been reached:
  - Solve $F(\beta^t,\alpha^t)$. Set $\text{lb}=v(F(\beta^t,\alpha^t))$ if $v(F(\beta^t,\alpha^t))>\text{lb}$.
  - Modify the solution of $F(\beta^t,\alpha^t)$ into a feasible solution $\widehat{F}(\beta^t,\alpha^t)$ using the two-stage procedure that will be explained shortly. If $v(\widehat{F}(\beta^t,\alpha^t))<\text{ub}$, set $\text{ub}=v(\widehat{F}(\beta^t,\alpha^t))$.
  - Update the multipliers as follows:

    $$\beta^{t+1}=\max\{0,\beta^t+s_1^t\cdot g_1^t\},$$
    $$\alpha^{t+1}=\max\{0,\alpha^t+s_2^t\cdot g_2^t\}.$$

    Here, g_1^t and g_2^t are the subgradient vectors. The jth component of g_1^t is defined as

    $$(g_1^t)_j=\sum_{k\in P}m_k y_j^k-S_j.$$

    Similarly, the (i,j)th component of g_2^t is defined as

    $$(g_2^t)_{ij}=\sum_{k\in P}\mu_k x_{ij}^k-S_{ij}.$$

    In updating the multipliers, the steplengths s_1^t and s_2^t are calculated as follows:

    $$s_i^t=\lambda\frac{1.05\cdot\text{ub}-v(V(\beta^t,\alpha^t))}{\|g_i^t\|^2},\quad i=1,2. \tag{7}$$
  - Increment t as $t+1$.
- Output ub as the best feasible solution.

In calculating the steplengths, Eq. (7) is used where λ is a convergence parameter. More details on this parameter will be provided in Section 4. The *gap* calculated at each iteration of the algorithm shows how far the current feasible solution may be from the optimal solution. Therefore, in the case that the algorithm is unable to find the optimal solution, it is capable of indicating the quality of the final solution.

At any step of the algorithm, the optimal solution of $F(\beta^t,\alpha^t)$ will be integer and also feasible with respect to constraints (4) and (5), but may not necessarily satisfy the capacity constraints (2) and (3). This (infeasible) solution needs to be converted into a feasible solution with respect to problem **F** in order to be able to provide the algorithm with an upper bound. The usual way to accomplish this is to use some fast heuristics to convert the infeasible solution to a feasible solution, however, at the expense of a possibly bad feasible solution. In contrast, we will use a reverse approach here and use integer programs (IPs) to obtain feasible solutions.

Our motivation is to make use of the information provided by the relaxed solution as much as possible. Such an approach, although at the expense of a higher computational effort, will be proven to be effective in quickly obtaining feasible solutions of good quality. The details of our procedure are given as follows:

3.2. Obtaining feasible solutions

Let $\hat{y}_j^k$ and $\hat{x}_{ij}^k$ be the optimal solution of the $F(\beta,\alpha)$. Using this solution, we attempt to achieve a feasible solution to **F** using a two-stage procedure (named as 2*SP*). In brief terms, the first stage of the 2*SP* attempts to obtain a feasible configuration of y variables by using an IP model named *FeasY*. Using the result of the first stage, we construct another IP model named *FeasX* in the second stage, whose solution provides a feasible configuration of x variables. Details are provided below:

3.2.1. Stage 1

The first stage of the 2*SP* consists of converting the $\hat{y}_j^k$'s so as to satisfy constraint (2) with a minimal amount of modification. The modification is performed for each node $j \in V$ such that any program that violates the capacity constraint is repositioned. For this purpose, we define the set $\mathrm{O}(j,k) = \{j \in V, k \in P \mid \hat{y}_j^k = 1\}$. The feasibility problem can then be solved by using the following IP model (henceforth denoted by *FeasY*):

$$(FeasY)\quad \text{minimize} \quad \sum_{k\in P}\sum_{j\in V} c_j^k y_j^k + \sum_{k\in P}\sum_{j\in V} R m_j^k \tag{8}$$

$$\text{subject to} \quad \sum_{k\in P} m_k y_j^k \leqslant S_j \quad \forall j \in V,$$

$$y_j^k \geqslant 1 - m_j^k \quad \forall j,k \in \mathrm{O}(j,k), \tag{9}$$

$$\sum_{j\in V} y_j^k \geqslant 1 \quad \forall k \in P, \tag{10}$$

$$y_j^k \in \{0,1\} \quad \forall j \in V,\ k \in P,$$

$$m_j^k \in \{0,1\} \quad \forall j,k \in \mathrm{O}(j,k). \tag{11}$$

In *FeasY*, the additional binary variable m_j^k is equal to one if program k on node j is repositioned to another node. To ensure that a minimal amount of modification is performed, a penalty is associated to each repositioning, which is reflected in the second summation of the objective function of *FeasY* by the penalty coefficient R. The motivation for such an approach is to benefit as much as possible from the information provided by the relaxed Lagrangean solution. In *FeasY*, constraints (9) together with the objective function ensure that if a program k already located at node j is repositioned to another node, then $y_j^k = 0$. Constraints (10) are used to ensure that after the modification, each program is available on at least one node. The optimal solution of *FeasY* yields a placement scheme for the programs such that no node constraint is violated. Below, we state a proposition that eases the solution of *FeasY*.

Proposition 2. Let $\overline{FeasY}$ denote the formulation where constraints (11) are replaced by $0 \leqslant m_j^k \leqslant 1$ for every pair $(j,k) \in \mathrm{O}(j,k)$. Then, in an optimal solution to $\overline{FeasY}$, no m_j^k will attain a fractional value.

Proof. Let $\bar{y}_j^k$ and $\overline{m}_j^k$ denote the optimal solution to $\overline{FeasY}$. We will consider two cases.

1. Let $\bar{y}_j^k = 1$ for a given pair $(j,k) \in \mathrm{O}(j,k)$. If $\overline{m}_j^k > 0$, then it is always possible to reduce $\overline{m}_j^k$ to 0 to obtain a solution that has a value $R\overline{m}_j^k$ less than that of the current one, while still satisfying constraint (9).
2. Let $\bar{y}_j^k = 0$ for a given pair $(j,k) \in \mathrm{O}(j,k)$. Then, $\overline{m}_j^k$ should be equal to 1 in order to satisfy constraint (9). □

The result of Proposition 2 implies that, instead of *FeasY*, one may solve $\overline{FeasY}$ which has a fewer number of binary variables. As will be seen shortly, our computational experience confirms that *FeasY* is easily solved to optimality with standard optimization software.

3.2.2. Stage 2

In the second stage of the $2SP$, we attempt to find a feasible configuration of x_{ij}^k variables, based on the optimal solution $\bar{y}_j^k$ of $FeasY$. In other words, we would like to obtain a vector of x variables satisfying the following IP model (henceforth referred to as $FeasX$):

$$(FeasX) \quad \text{minimize} \quad \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j} c_{ij}^k x_{ij}^k \tag{12}$$

$$\text{subject to} \quad \sum_{k\in P} m_k x_{ij}^k \leqslant S_{ij} \quad \forall i\neq j\in V : \bar{y}_i^k = 1,\ \bar{y}_j^k = 0, \tag{13}$$

$$\sum_{i\in V, i\neq j, \bar{y}_i^k=1} x_{ij}^k = 1 \quad \forall j\in V,\ \ k\in P : \bar{y}_j^k = 0, \tag{14}$$

$$x_{ij}^k \in \{0,1\} \quad \forall i\neq j\in V : \bar{y}_i^k = 1,\ \bar{y}_j^k = 0,\ \ k\in P. \tag{15}$$

Note that in model $FeasX$, the binary variables x_{ij}^k are only defined if $\bar{y}_i^k = 1$ and $\bar{y}_j^k = 0$ for every triplet $(i\in V, j\in V, k\in P)$. Therefore, the size of the model is greatly reduced as compared to problem **F**. The optimal solution of the $FeasX$ yields a feasible configuration of x_{ij}^k variables, since constraints (13) will ensure that the solution will obey the link capacities.

As a result of stages 1 and 2, we obtain the optimal objective values for the formulations $FeasY$ and $FeasX$. The objective value of the corresponding solution for problem **F** is then found through $v(FeasY) + v(FeasX) - \sum_{k\in P}\sum_{j\in V} Rm_j^k$.

4. Computational results

In this section, we describe our computational results with the proposed algorithm on randomly generated test problems. The Lagrangean relaxation and decomposition algorithm has been implemented in C and all the tests are performed on a Sun UltraSPARC 12×400 MHz with 3 GB RAM, using CPLEX 9.0 as the optimization package to solve the IPs.

For the computational experiments, a batch of 16 random problems have been generated with the number of nodes (n) ranging from 50 to 80, and the number of programs (m) ranging from 20 to 50. As for the parameters, μ_k, c_{ij}^k, c_j^k are randomly generated from a continuous uniform distribution between 50 and 100. m_k is modelled as $m_k = \mu_k T_k$, where T_k is the total transmission time for program k. In the experiments, $T_k = 10$ minutes for all $k \in P$. S_{ij} values have been chosen from the uniform distribution between $\max_{k\in K}\{\mu_k\}$ and $\sum_{k\in K}\mu_k$. The capacity of each node (S_j) is set to be 40% of the total size of all the programs and the penalty parameter R is set to $10 \cdot \max_{k\in P, j\in V}\{c_j^k\}$.

The parameters for the algorithm are chosen as follows. The convergence parameter λ is initially set to 2.00 and multiplied by 0.87 if there is not any improvement in the best known upper bound for five consecutive iterations.

It was previously stated that the algorithm proposed here is a solution procedure that is capable of providing both upper and lower bounds at every iteration. This, in turn, outputs an integrality gap that is an indicator of the quality of the solution found. Therefore, we do not compare our algorithm with the heuristic procedure proposed by Ouveysi et al. [12]. However, we do compare it with CPLEX 9.0, a powerful commercial optimization package. To be fair in comparisons, we impose a common time limit of 300 seconds on both algorithms, considering the dynamic nature of the problem requiring repeated resolving to adopt to the changes in the demand pattern and available programs.

We present the computational results in Table 1. Each row of the table contains the average values of five randomly generated instances. The columns of the table are explained below:

- n: number of nodes;
- m: number of programs;
- n_{L}: number of iterations required by the algorithm;
- t_{sub}: average time required to solve all the subproblems to optimality (in seconds);
- t_{FeasY}: average time required to solve $FeasY$ to optimality (in seconds);

Table 1
Computational results for the Lagrangean relaxation and decomposition algorithm

n	m	n_L	t_{sub}	t_{FeasY}	t_{FeasX}	*igap*	*gap*	d_{CPLEX}
50	20	13.2	13.78	0.12	0.53	3.23	2.03	0.82
60	20	12.4	23.91	0.15	0.78	4.04	2.20	0.91
70	20	7	36.45	0.17	1.08	3.37	1.82	0.12
80	20	6.4	58.20	0.20	1.39	2.92	2.45	−0.30
50	30	16.4	15.93	0.18	0.97	4.01	2.70	0.98
60	30	10	30.02	0.23	1.37	3.01	2.27	0.72
70	30	6.4	52.20	0.25	2.02	2.95	2.58	−2.74
80	30	4	76.55	0.31	2.47	2.65	1.99	−5.08
50	40	10.6	26.72	0.24	1.49	4.43	2.81	0.53
60	40	7.6	40.64	0.27	2.02	2.95	2.38	−0.56
70	40	5.4	61.94	0.34	2.77	2.56	2.26	−3.63
80	40	3.4	98.26	0.41	3.43	2.15	1.72	−4.80
50	50	9.4	33.75	0.30	2.01	3.28	2.67	−0.10
60	50	5.6	58.94	0.35	2.93	3.38	2.60	−2.61
70	50	4	90.00	0.40	3.83	2.92	2.56	−3.89
80	50	3	139.18	0.47	4.89	2.24	2.02	−4.07

- t_{FeasX}: average time required to solve *FeasX* to optimality (in seconds);
- *igap*: initial gap obtained at the beginning of the algorithm (%);
- *gap*: final gap obtained at the end of the algorithm (%);
- d_{CPLEX}: comparison of the algorithm with CPLEX, which shows average percent difference between the best solution found by the proposed algorithm (denoted by v_{opt}) and that of CPLEX (denoted by v_C) within the given time limit, and calculated as $\frac{v_{opt}-v_C}{v_{opt}} \cdot 100$.

The results presented in Table 1 indicate that the algorithm presented here is able to produce good quality solutions (typically around 2% of the optimal), even in the first iteration for most of the problems. In addition, the proposed algorithm is observed to be capable of providing better solutions than those found by CPLEX in the same amount of time, especially as the instances grow in size. As can also be seen in Table 1, the time required to obtain a feasible solution at every step of the algorithm is quite small. However, one drawback of the algorithm lies in obtaining lower bounds, where the required computation time t_{sub} increases heavily with the number of nodes. This is due to the fact that, at every iteration, the algorithm needs to solve m integer subproblems to optimality. This, in turn, makes the algorithm computationally inefficient for large size instances since the number of subproblems will increase with the number of programs. To overcome this drawback, we propose a simple modification to the algorithm that appears to be quite efficient and is as described below in detail.

4.1. A modified algorithm

Since solving m integer subproblems at every iteration of the algorithm is costly, we propose a modification to the algorithm that consists of solving the LP-relaxation of each integer subproblem $F_{k^*}(\beta, \alpha)$ as opposed to solving it as a binary program. In this case, the lower bound obtained will surely be below the lower bound obtained by the original algorithm, but solving linear programs instead of binary programs at each iteration will expectedly help in speeding up the algorithm. The only complication with this modification is that the optimal solutions of the LP-relaxations of the subproblems will in general be fractional, if not always. As the two-stage procedure requires integer variables as input, such fractional solutions can not be used in obtaining feasible solutions. However, this situation can be fixed through rounding up (to 1) every fractional variable with value greater or equal to 0.50, and rounding down (to 0) the rest. Using the rounded solution, a feasible solution can then be computed using $2SP$, as discussed previously.

Our computational experience with this version of the algorithm shows that such a modification greatly helps in reducing the solution time for lower bound calculation at each iteration of the algorithm. We demonstrate this in Table 2, where the original and the modified algorithms are compared on some test problems. The first seven columns of this table are as explained above. In the next three columns, we report the average computation time required to find lower bounds (denoted by $\hat{t}_{\text{sub}}$), to solve problem *FeasY* (denoted by $\hat{t}_{FeasY}$) and to solve *FeasX* (denoted by $\hat{t}_{FeasX}$) with the modified algorithm.

In the modified algorithm, one may expect that the performance in terms of the final gaps will deteriorate since one is solving linear programming problems instead of integer programs. To see how much is lost in terms of the gaps produced by this modification, we provide two additional columns $\widehat{igap}$ and $\widehat{gap}$ that report the initial and final gaps found by the modified algorithm, respectively. Finally, the last column denoted by *imp* in Table 2 shows the amount of improvement one obtains in the solution time of lower bound computation with the use of the suggested modification (calculated as $\frac{t_{\text{sub}}-\hat{t}_{\text{sub}}}{t_{\text{sub}}} \cdot 100$).

The numerical values given in Table 2 show that our modification proposal does not have any effect on reducing the computation times to obtain feasible solutions. However, it does have a tremendous effect in reducing the necessary computation times to find lower bounds. As indicated under column *imp*, the time savings can be as high as 90%. Furthermore, these results indicate that not much is lost with respect to the final gaps output by the algorithm since the solutions obtained are still near-optimal (typically around 5% of the optimal solution). Based on these results, we proceed on solving larger instances with the modified algorithm, where the number of nodes range from 50 to 100, and the number of programs range from 50 to 90. In solving these instances, we keep all the algorithm parameters as previously explained. This time 30 problems have been generated. The results are presented in Table 3, where each row contains the average values calculated over four random instances. The columns of Table 3 are as explained previously. The only additional column

Table 2
Comparison of the original and modified algorithm in terms of solution time and gap

n	*m*	Original algorithm					Modified algorithm					*imp*
		t_{sub}	t_{FeasY}	t_{FeasX}	*igap*	*gap*	$\hat{t}_{\text{sub}}$	$\hat{t}_{FeasY}$	$\hat{t}_{FeasX}$	$\widehat{igap}$	$\widehat{gap}$	
50	10	11.31	0.07	0.23	13.51	7.78	2.15	0.07	0.25	13.71	8.63	81.03
60	10	22.75	0.08	0.32	7.91	6.91	3.05	0.08	0.33	8.59	6.42	86.59
70	10	41.12	0.09	0.48	7.73	4.76	4.29	0.09	0.46	8.16	5.37	89.58
80	10	51.54	0.11	0.58	5.47	3.70	6.02	0.10	0.53	6.69	6.10	88.32
90	10	22.71	0.11	0.76	3.63	2.44	7.93	0.11	0.78	4.69	2.24	65.07
100	10	64.31	0.13	0.90	3.82	3.64	9.83	0.12	0.87	7.00	6.38	84.71
50	20	18.08	0.11	0.52	5.01	2.31	4.31	0.12	0.58	8.27	3.89	76.18
60	20	30.34	0.16	0.77	3.19	3.10	6.25	0.15	0.80	4.75	3.71	79.39
70	20	29.33	0.16	1.11	3.46	2.36	8.60	0.16	1.13	6.26	5.04	70.68
80	20	35.42	0.18	1.31	1.30	1.12	11.42	0.19	1.33	4.01	3.47	67.76
90	20	74.56	0.20	1.88	2.32	1.91	15.87	0.24	1.70	3.40	3.34	78.72
100	20	157.08	0.23	2.13	2.66	2.12	21.29	0.29	2.22	4.09	3.78	86.45
50	30	12.96	0.16	0.88	3.89	3.26	6.49	0.18	0.88	6.22	4.94	49.94
60	30	19.88	0.26	1.46	3.80	2.49	9.41	0.22	1.22	5.25	4.95	52.68
70	30	31.16	0.24	2.46	2.42	2.41	13.17	0.34	1.79	3.56	2.59	57.73
80	30	112.63	0.23	2.20	2.49	1.95	17.26	0.43	2.19	4.29	3.29	84.68
90	30	136.87	0.31	3.13	2.87	2.29	24.40	0.33	2.88	6.37	5.21	82.17
100	30	143.16	0.34	3.68	2.15	2.11	30.93	0.36	3.55	4.62	4.16	78.39
50	40	25.67	0.25	1.77	3.45	2.27	8.76	0.24	1.27	5.72	3.77	65.90
60	40	36.17	0.24	1.85	3.78	2.45	12.93	0.30	1.92	6.47	5.28	64.25
70	40	43.29	0.31	2.93	2.64	2.62	17.96	0.35	2.80	4.49	4.05	58.52
80	40	111.22	0.40	3.47	2.31	2.04	24.92	0.40	3.50	4.67	4.09	77.60
90	40	132.14	0.42	4.53	2.61	2.07	32.48	0.35	4.41	5.82	5.07	75.42
100	40	142.76	0.47	5.23	1.05	1.05	41.22	0.47	4.89	3.24	3.24	71.13

Table 3
Comparison results of the modified Lagrangean relaxation and decomposition algorithm with CPLEX

n	m	n_{L}	$\hat{t}_{\text{sub}}$	$\hat{t}_{FeasY}$	$\hat{t}_{FeasX}$	*igap*	*gap*	d_{CPLEX}	n_{s}
50	50	20.25	10.94	0.32	4.36	6.25	5.41	2.55	4/4
60	50	15.75	16.02	0.41	3.47	5.84	5.35	−0.35	4/4
70	50	11	23.04	0.45	6.68	5.36	5.12	−2.34	4/4
80	50	9	31.09	0.44	4.96	4.33	4.29	−2.19	4/4
90	50	6.25	40.46	0.65	6.78	3.93	3.82	−2.52	1/4
100	50	5.25	51.72	0.72	7.96	4.07	3.82	–	0/4
50	60	17.5	13.27	0.47	3.70	6.86	6.22	0.23	4/4
60	60	12.5	19.66	0.47	4.62	5.33	5.01	−1.99	4/4
70	60	9.25	27.51	0.54	5.85	5.35	4.85	−1.47	4/4
80	60	7	36.86	0.67	7.16	3.98	3.92	−2.14	3/4
90	60	5.75	48.60	0.68	9.40	4.65	4.56	–	0/4
100	60	4.5	62.12	0.69	17.33	3.71	3.63	–	0/4
50	70	15.25	15.81	0.53	3.90	6.70	5.90	−0.40	4/4
60	70	10.5	23.40	0.52	6.59	4.82	4.72	−0.64	4/4
70	70	7.25	31.86	0.61	11.80	5.43	5.27	−1.44	4/4
80	70	6	43.86	0.83	11.75	4.90	4.81	–	0/4
90	70	4.75	56.46	0.75	14.68	4.39	4.31	–	0/4
100	70	4	72.83	0.88	19.48	4.82	4.70	–	0/4
50	80	12	17.93	0.61	7.86	5.57	5.03	−0.59	4/4
60	80	8.75	25.84	1.05	10.23	5.48	4.96	−1.62	4/4
70	80	7	36.42	0.77	8.48	5.11	4.89	–	0/4
80	80	5	49.00	0.79	24.38	4.43	4.43	–	0/4
90	80	4	66.66	0.97	18.51	4.40	4.30	–	0/4
100	80	3	82.40	0.88	21.07	4.07	4.07	–	0/4
50	90	9.75	20.78	0.67	27.61	6.46	5.72	−0.99	4/4
60	90	6.5	29.05	0.75	26.30	5.10	5.06	−1.06	4/4
70	90	4.25	40.61	0.94	42.21	5.38	5.28	–	0/4
80	90	3.5	56.13	1.10	81.74	5.27	4.95	–	0/4
90	90	4	73.68	1.02	18.23	4.28	4.25	–	0/4
100	90	3	95.17	1.21	22.85	4.40	4.35	–	0/4

is n_{s}, which denotes the number of instances out of four for which CPLEX was able to find an integer feasible solution within the given time limit.

Looking at the results given in Table 3 we see that the modified algorithm provides good quality solutions (typically with a gap below 5%) in a reasonable amount of time. CPLEX, on the other hand, fails to find an integer feasible solution within the given time limit, as problems grow larger in size.

We have also carried out additional experiments in comparing the proposed algorithm and CPLEX on problems of larger size. We compare the algorithm with two variants of CPLEX, i.e., one with CPLEX's emphasis on optimality and the other with an emphasis on feasibility. The second variant will make sure that better feasible solutions will be generated earlier during the solution process.

As far as the results are concerned, for an instance with $n = 50$ and $m = 200$, the proposed algorithm was able to produce an integer feasible solution within 2 minutes of solution time, whereas both variants of CPLEX are only able to output an integer feasible solution after 10 minutes of solution time. In addition, CPLEX can produce a solution dominating that of the proposed algorithm only after 20 minutes of solution time for this instance. For an even larger problem with $n = 100$ and $m = 250$, the proposed algorithm was able to produce an integer feasible solution within 5 minutes of computing time, whereas CPLEX could not discover a feasible solution even after 60 minutes (1 hour). These results suggest that the proposed algorithm is a viable alternative to CPLEX, especially when good quality solution needed in a short amount of time. This is a desirable characteristic in a solution algorithm for such a problem as the VPRP, where the parameters of the problem (e.g., demand) may change on an hourly basis.

5. Concluding remarks

In this paper, we have presented a Lagrangean relaxation and decomposition algorithm for the resolution of the video placement and routing problem (VPRP). Our algorithm is capable of producing good quality solutions in considerably short running times and provides a benchmark to measure the quality of the output results. The algorithm proposed in this paper is different from similar existing algorithms because we achieve the feasible solutions through the use of integer programming techniques and this is the reason that our solution methodology results in good quality solutions even at the earlier iterations of the algorithm. Computational results indicate that the proposed algorithm is able to outperform a state-of-the art commercial optimization package with respect to obtaining near-optimal solutions.

It is clear that obtaining the optimal solution of the model considered here will get harder as the problem sizes increase. In such cases, fast heuristic algorithms can be of use. However, one must be aware that such algorithms are incapable of providing the quality of the solution found unless additional lower bounding techniques are employed.

References

[1] S.A. Barnett, G.J. Anido, A cost comparison of distributed and centralized approaches to video-on-demand, IEEE Journal on Selected Areas in Communications 14 (6) (1996) 1173–1182.
[2] J. But, D. Egan, Designing a scalable video-on-demand system, in: International Conference on Communications, Circuits and Systems (ICCCAS02), 2002, pp. 559–565.
[3] G. Cornuejols, G.L. Nemhauser, L.A. Wolsey, The uncapacitated facility location problem, in: P.B. Mirchandani, R.L. Francis (Eds.), Discrete Location Theory, John Wiley & Sons, 1990, pp. 119–171 (Chapter 3).
[4] M. Held, P. Wolfe, H.P. Crowder, Validation of subgradient optimization, Mathematical Programming 6 (1974) 62–88.
[5] Y.-F. Huang, C.-C. Fang, Load balancing for clusters of VOD servers, Information Sciences 164 (2004) 113–138.
[6] R.-H. Hwang, P.-H. Chi, Fast optimal video placement algorithms for hierarchical video-on-demand systems, IEEE Transactions on Broadcasting 47 (4) (2001) 357–366.
[7] Y.K. Kim, J.Y. Kim, S.S. Kang, A tabu search approach for designing a non-hierarchical video-on-demand network architecture, Computers and Industrial Engineering 33 (3–4) (1997) 837–840.
[8] Y.-W. Leung, E.W.M. Wong, An incentive charging scheme for video-on-demand, Journal of the Operational Research Society 52 (2001) 55–63.
[9] T.D.C. Little, D. Venkatesh, Prospects for interactive video-on-demand, IEEE Multimedia 1 (3) (1994) 14–24.
[10] J.C.L. Liu, J. Hsieh, D.H.C. Du, M.J. Lin, Performance of a storage system for supporting different video types and qualities, IEEE Journal on Selected Areas in Communications 14 (7) (1996) 1314–1331.
[11] X. Liu, S.T. Vuong, Supporting low-cost video-on-demand in heterogeneous peer-to-peer networks, in: Seventh IEEE International Symposium on Multimedia, 2005, pp. 523–533.
[12] I. Ouveysi, L. Sesana, A. Wirth, The video placement and routing problem, in: Operations Research/Management Science at Work: Applying Theory in the Asia Pacific RegionInternational Series in Operations Research and Management Science, vol. 43, Kluwer Academic Publishers, 2002, pp. 53–71.
[13] I. Ouveysi, K.-C. Wong, S. Chan, K.T. Ko, Video placement and dynamic routing algorithms for video-on-demand networks, in: Proceedings of the IEEE Globecom'98 Conference, vol. 2, 1998, pp. 658–663.
[14] C.-F. Wang, B.-R. Lai, R.-H. Jan, Optimum multicast of multimedia streams, Computers and Operations Research 26 (1999) 461–480.
[15] T.H. Wu, I. Korpeglu, B.C. Cheng, Distributed interactive video system design and analysis, IEEE Communications Magazine 35 (3) (1997) 100–108.