A Lagrangean Relaxation and Decomposition Algorithm for the Video Placement and Routing Problem

Tolga Bektaş *, Osman Oğuz
Department of Industrial Engineering
Bilkent University
06800 *Ankara, Turkey*

Iradj Ouveysi
The University of Melbourne
VIC 3010, Australia

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, we develop an exact solution algorithm based on Lagrangean relaxation and decomposition to solve the video placement and routing problem. The novelty of the approach can be described as the use of integer programs to obtain feasible solutions to the problem. We also provide a modification to accelerate the proposed algorithm. Computational experimentation reveals that for randomly generated problems with up to 100 nodes and 100 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.

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. Commercial VoD services are now available in many areas that the multimedia technologies are developing very fast. With

*Corresponding author. Phone: +903122341010, Fax: +903122341051, E-mail: tolgab@bilkent.edu.tr

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.

An in-depth treatment of the subject is given by Little and Venkatesh [6]. Here, we only provide a brief explanation. A complete VoD system consists of three fundamental components which may be stated as the storage server, 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 user request for a 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 indicated by Little and Venkatesh [6]. The first consists of deciding on program allocation and the second is resource location and connection establishment.

Several studies exist addressing the problem of developing a VoD system. To mention a few, Kim et al. [4] consider designing a VoD system on a network with storage capacity constraints on each node and no capacity limitations on links between each pair of nodes. They offer an integer linear programming formulation of the problem along with a tabu search algorithm. The authors present computational results for networks with up to 40 nodes and 200 programs. Wang et al. [9] 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 and the server sends the program through this tree. The authors present a branch and bound algorithm to find an optimal solution when the network is a directed acyclic graph and propose an approximation algorithm for general graphs. Hwang

and Chi [3] 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. In the context of a VoD system, Leung and Wong [5] address a different aspect of the problem as what kind of a charging scheme should a service provider adopt in order to maximize the mean revenue. Ouveysi et al. [7] proposed an integer programming formulation to determine the location of the video programs so as to minimize the total cost of storage and transmission, subject to storage and transmission capacity constraints. They refer to this problem as the Video Placement and Routing Problem, for the solution of which the authors propose heuristic approaches. Finally Huang and Fang [2] propose a dynamic load balancing 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 model introduced by Ouveysi et al. [7]. Our algorithm is based on Lagrangean relaxation and decomposition, coupled with some integer programming techniques to convert infeasible solutions to feasible solutions. This approach in contrast to 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 define the problem and present the integer linear programming formulation. Section 3 provides the full details of the Lagrangean relaxation and decomposition algorithm, with the implementation results on randomly generated instances given in Section 4. Conclusions and further remarks are given in Section 5.

2 Problem Definition And Formulation

In defining the problem, we adhere to the notation of Ouveysi et al [7], given as follows. There exists a fully meshed network modelled by an undirected graph $G = (V, A)$, where $V = \{1, 2, ..., n\}$ is the set of nodes and A is the set of edges including the $n(n-1)/2$ links of the network. The set of programs (videos) to be placed and routed is denoted by $P = \{1, 2, ..., m\}$, where each program $k \in P$ has a capacity requirement denoted by m_k and a bandwidth requirement for transmission denoted by μ_k. Each node $j \in V$ corresponds to a potential location for storing the programs with capacity denoted by $C_s(j)$. In addition, each node has a demand for each program. The cost of storing a program $k \in P$ at node $j \in V$ is shown by $s_k(j)$ and the transmission cost of the program on the link $\{i, j\} \in A$ is shown by $t_k(i, j)$. Finally, each link $\{i, j\} \in A$ has a transmission capacity that is denoted by $C_t(i, j)$. A fully meshed VoD architecture with 5 servers is given in Figure 1.

Given the demand forecast of the programs, the *Video Placement and Routing Problem* (*VRPR*) is to find a placement scheme for the programs such that the total cost of storage and transmission of the programs in the network is minimized and the demand of each node for each program is satisfied.

Ouveysi et al. [7] provide a integer linear programming formulation for this problem, using the following binary decision variables:

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

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

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

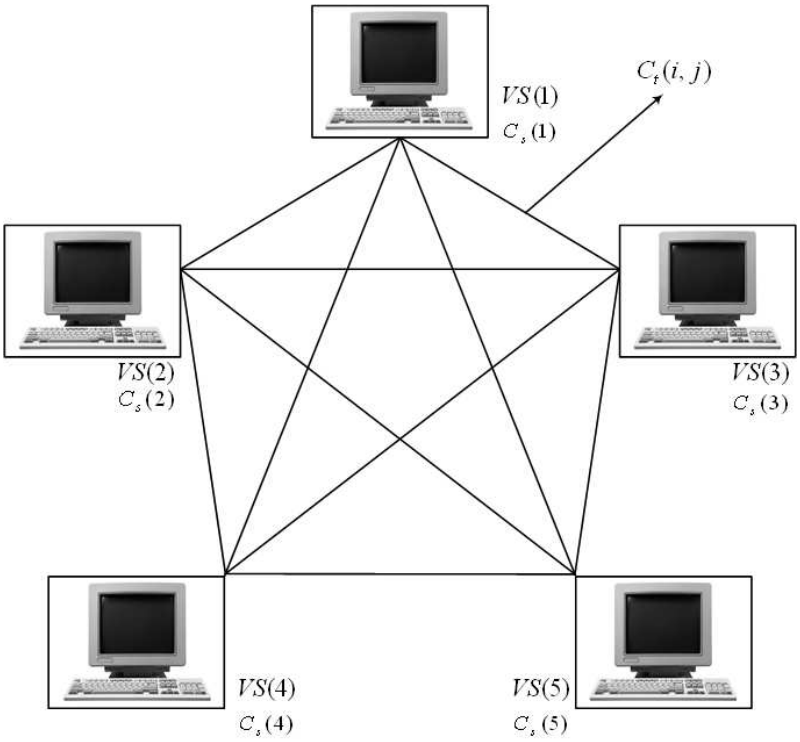

Figure 1: A fully meshed VoD architecture with 5 servers

$$\textbf{(F)} \qquad \text{minimize} \sum_{k\in P}\sum_{j\in V} s_k(j)y_j^k + \sum_{k\in P}\sum_{j\in V}\sum_{i\in V, i\neq j} t_k(i,j)x_{ij}^k \tag{1}$$

s.t.

$$\sum_{k\in P} m_k y_j^k \leq C_s(j), \quad \forall j \in V \tag{2}$$

$$\sum_{k\in P} \mu_k x_{ij}^k \leq C_t(i,j), \quad \forall i \neq j \in V \tag{3}$$

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

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

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

In this model, constraints (2) and (3) correspond to the capacity constraints related with 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. It is clear that this formulation only allows one-hop paths in transmitting a program. The reader is referred to Ouveysi et al. [8] for the generalization of this problem to two-hop paths. 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.

We now study the complexity of the VPRP. Ouveysi et al. [7] 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 single program (i.e., $P = \{1\}$) and let $\mu_1 \leq min_{\{i,j\}\in A} C_t(i,j)$ and $m_1 \leq min_{i\in V} C_s(i)$. 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 $V = I \bigcup J$ where $y_j \leq 0$ 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 $x_{ij} \leq y_i, \forall i \in I, j \in J$. But then **F** reduces to the well-known uncapacitated facility location problem (see [1]) 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. Therefore, in this paper, we propose an exact solution algorithm for problem **F** based on Lagrangean relaxation and decomposition. We present the details of the algorithm in the next section.

3 A Lagrangean Relaxation and Decomposition Algorithm

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

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

$$\text{s.t.}$$

$$(4), (5), (6)$$

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

$$(F_{k^*}(\beta,\alpha)) \qquad \text{minimize} \sum_{j\in V}(s_{k^*}(j)+\beta_j m_k)y_j^{k^*} + \sum_{j\in V}\sum_{i\in V, i\neq j}(t_{k^*}(i,j)+\alpha_{ij}\mu_{k^*})x_{ij}^{k^*}$$

s.t.

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

$$x_{ij}^{k^*} \leq y_i^{k^*}, \quad \forall i \neq j \in V$$

$$x_{ij}^{k^*}, y_j^{k^*} \in \{0,1\}, \quad \forall i \neq j \in V$$

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**.

The Algorithm:

- Start with an initial vector of multipliers β^1, α^1. Let the incumbent lower bound be $lb = -\infty$, incumbent upper bound be $ub = \infty$ and $t = 1$.
- Perform the following until $gap = \frac{ub-lb}{ub} < 1.00$ or the maximum amount of iterations have been reached.
  - Solve $F(\beta^t, \alpha^t)$. Set $lb = v(F(\beta^t, \alpha^t))$ if $v(F(\beta^t, \alpha^t)) > lb$.
  - 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 later on. If $v(\widehat{F}(\beta^t, \alpha^t)) < ub$, set $ub = v(\widehat{F}(\beta^t, \alpha^t))$.
  - 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 j^{th} component of g_1^t is defined as:

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

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

$$(g_2^t)_{ij} = \sum_{k \in P} \mu_k x_{ij}^k - C_t(i,j)$$

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

$$s_i^t = \lambda \frac{1.05 \cdot ub - v(V(\beta^t, \alpha^t))}{\|g_i^t\|^2}, \qquad i = 1, 2 \tag{7}$$

– Increment t as $t + 1$.

- Output ub as the best feasible solution.

In calculating the steplengths, the equation (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 is 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 solution of $F(\beta^t, \alpha^t)$ provides an integral solution that is feasible with respect to constraints (4) and (5), but do 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. In general, in such circumstances, some fast heuristics are used to convert the infeasible relaxed solution to a feasible solution, at the expense of a possibly bad feasible solution. However, we consider a reverse approach and utilize integer programs (IPs) techniques to obtain feasible solutions. Our motivation is to make use of the information provided by the current 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 providing feasible solutions of good quality. The details of our procedure is as follows:

3.1 Obtaining Feasible Solutions

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

3.1.1 Stage 1

The first stage of the $2SP$ consists of converting the $\widehat{y}_j^k$'s so as to satisfy constraint (2) with a minimal amount of modification. The modification is done for each node $j \in V$, through repositioning any program that violates the capacity constraint. To achieve this, we define the set $O(j,k) = \{j \in V, k \in P | \widehat{y}_j^k = 1\}$. Then, the feasibility is accomplished through the use of the following feasibility IP model (henceforth denoted by *FeasY*):

$$\textit{(FeasY)} \qquad \text{minimize} \sum_{k\in P}\sum_{j\in V} s_k(j)y_j^k + \sum_{k\in P}\sum_{j\in V} Rm_j^k \tag{8}$$

s.t.

$$\sum_{k\in P} m_k y_j^k \le C_s(j), \quad \forall j \in V$$

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

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

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

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

In *FeasY*, the additional binary variable m_j^k is equal to one if program k on node j is repositioned to another node, with, however, a penalty for each repositioning. This is reflected in the second summation of the objective function of *FeasY* with a penalty coefficient R and ensures that the modification performed to the solution is minimal. The motivation for such an approach is to benefit as much as possible from the information provided by the relaxed Lagrangean solution. Constraints (9) stipulate 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 located on at least one node. In short, the solution to *FeasY* yields a placement scheme for the programs such that no node constraint is violated. Our computational experience shows that *FeasY* is easily solved to optimality with standard optimization software.

We also note that in *FeasY*, it is possible to relax the binary variables m_j^k in the interval $[0,1]$, since it is easy to see that in any optimal solution to *FeasY*, no m_j^k will attain a fractional value.

3.1.2 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 values of the variables $\overline{y}_j^k$ 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*):

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

s.t.

$$\sum_{k\in P} m_k x_{ij}^k \leq C_t(i,j), \quad \forall i\neq j\in V : \overline{y}_i^k = 1, \overline{y}_j^k = 0$$

$$\sum_{i\in V, i\neq j, \overline{y}_i^k=1} x_{ij}^k = 1, \quad \forall j\in V, k\in P : \overline{y}_j^k = 0 \tag{13}$$

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

Note that in model $FeasX$, the binary variables x_{ij}^k are only defined if $\overline{y}_i^k = 1$ and $\overline{y}_j^k = 0$. Therefore, the size of the model is greatly reduced as compared to problem **F**. The optimal solution to the *FeasX* yields a feasible configuration of x_{ij}^k variables.

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 (feasible) 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 proposed for the solution of **F** has been implemented in C and all the tests are performed on a Sun UltraSPARC 12x400 MHz with 3 GB RAM, using CPLEX 9.0 as the optimization

package to solve the IPs.

The random problems have been generated with the following parameters:

- $m_k \sim U(1, 100)$
- $\mu_k \sim U(1, 100)$
- $t_k(i, j) \sim U(1, 100)$
- $s_k(j) \sim U(1, 100)$
- $C_t(i, j) \sim U(max_{i,j}\{t_k(i, j)\}, \sum_{i,j} t_k(i, j))$

The capacity of each node ($C_s(j)$) is set to be 40% of the total size of all the programs. For the Lagrangean algorithm, the convergence parameter is initially set to 2.00 and multiplied by 0.87 if there is not any improvement in the best known upper bound for 5 consecutive iterations. The algorithm is stopped if one of the following conditions is satisfied: the $gap < 1.00$, $t > 100$ or there is no improvement in the best known upper bound for 9 successive iterations. The last criteria has been set based on our observations during preliminary experiments. All the subproblems are solved to optimality using CPLEX 9.0.

It is previously stated that the proposed algorithm herein is an exact solution procedure as it provides 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. [8]. We present the computational results in Table 1, where each row presents a randomly generated problem. 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
- t_{FeasY}: average time required to solve $FeasY$ to optimality
- t_{FeasX}: average time required to solve $FeasX$ to optimality
- t_L: overall solution time required by the algorithm
- *igap*: initial gap obtained at the beginning of the algorithm
- *gap*: final gap obtained at the end of the algorithm

Insert here Table 1.

The results presented in Table 1 indicate that the algorithm presented here is able to produce good quality solutions, even in the first iteration, for most of the problems. As also indicated in Table 1, the time required to obtain feasible solution at every step of the algorithm is quite short. However, one drawback of the algorithm lies in obtaining lower bounds. The lower bound computation time, as shown under t_{sub} column, 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 larger sized instances, as the size and the number of subproblems will increase as well. To overcome this drawback, we propose a simple modification to the algorithm that appears to be quite efficient, as described in detail below.

4.1 A Modified Algorithm

Since solving m integer subproblems at every iteration of the algorithm is costly, we propose to solve the LP-relaxation of each integer subproblem. The lower bound obtained in this case will surely be below the lower bound obtained by solving integer subproblems, but will help in speeding up the algorithm. The only complication with this modification is

that the solutions of the subproblems will in general be fractional, if not always. However, the fractional solutions can easily be converted to integer solutions that will be used to obtain feasible solutions to the original problem. This is done through rounding up (to 1) every fractional variable with value greater or equal to 0.50, and rounding down the rest. Using this solution, a feasible solution can easily be computed using formulations *FeasY* and *FeasX*, as discussed previously.

Our computational experience with this version of the algorithm shows that such a modification greatly helps in reducing the solution time of calculation of the lower bound at each iteration of the algorithm. To see this, we present Table 2, where we compare the original and the modified algorithm on some test problems. The first five columns are self explanatory. In the next three columns, we report the average computation time required to find lower bounds (denoted by $\widehat{t}_{sub}$), to solve problem *FeasY* (denoted by $\widehat{t}_{FeasY}$) and to solve *FeasX* (denoted by $\widehat{t}_{FeasX}$), respectively. Finally , the last column denoted by *imp*, shows the improvement in the solution time in finding lower bounds (calculated as $\frac{t_{sub}-\widehat{t}_{sub}}{t_{sub}} * 100$.)

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*, these improvements can be up to 90%. Based on these results, we proceed on solving larger instances with this modification and the results are presented in Table 3. In solving these instances, we keep all the algorithm parameters as explained previously.

Looking at the results in 1 we see that the modified algorithm provides good quality solutions (typically with a gap below 5%) in a reasonable amount of time. However, as problem size increases, the difference between the initial gap and the final gap tends to decrease.

5 Concluding Remarks

In this paper, we have presented an hybrid Lagrangean relaxation-decomposition algorithm for the resolution of the Video Placement and Routing Problem (VRPR). Our algorithm provides a benchmark to measure the quality of the output results and is capable of producing good quality solutions in considerably short running times. 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 would result in good quality solutions even at the earlier iterations of the algorithm. However, obtaining a lower bound at each step of the algorithm seems to be the main bottleneck as one has to solve a number of integer programs at each step. For this reason, we have also proposed a modification to our algorithm that considerably accelerates its running time.

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 or metaheuristics such as tabu search can be of use. However, one must be aware that such approaches are incapable of providing the quality of the solution found unless additional lower bounding techniques are employed.

References

[1] G. Cornuejols, G.L. Nemhauser, and L.A. Wolsey. The uncapacitated facility location problem. In P.B. Mirchandani and R.L. Francis, editors, *Discrete Location Theory*, chapter 3, pages 119–171. John Wiley & Sons, 1990.

[2] Y.-F. Huang and C.-C. Fang. Load balancing for clusters of VOD servers. *Information Sciences*, 164:113–138, 2004.

[3] R.-H. Hwang and P.-H. Chi. Fast optimal video placement algorithms for hierarchical video-on-demand systems. *IEEE Transactions on Broadcasting*, 47(4):357–366, December 2001.

[4] Y.K. Kim, J.Y. Kim, and S.S. Kang. A tabu search approach for designing a non-hierarchical video-on-demand network architecture. *Computers and Industrial Engineering*, 33(3-4):837–840, 1997.

[5] Y.-W. Leung and E.W.M. Wong. An incentive charging scheme for video-on-demand. *Journal of the Operational Research Society*, 52:55–63, 2001.

[6] T.D.C. Little and D. Venkatesh. Prospects for interactive video-on-demand. *IEEE Multimedia*, 1(3):14–24, Autumn/Fall 1994.

[7] I. Ouveysi, L. Sesana, and A. Wirth. *Operations Research / Management Science at Work: Applying Theory in the Asia Pacific Region*, volume 43 of *International Series in Operations Research and Management Science*, chapter The video placement and routing problem, pages 53–71. Kluwer Academic Publishers, 2002.

[8] I. Ouveysi, K.-C. Wong, S. Chan, and K.T. Ko. Video placement and dynamic routing algorithms for video-on-demand networks. In *Proceeding of the IEEE Globecom'98 Conference*, volume 2, pages 658–663, 1998.

[9] C.-F. Wang, B.-R. Lai, and R.-H. Jan. Optimum multicast of multimedia streams. *Computers and Operations Research*, 26:461–480, 1999.

Table 1: Computational results for the Lagrangean relaxation and decomposition algorithm

n	m	n_L	t_{sub}	t_{FeasY}	t_{FeasX}	t_L	$igap$	gap
50	10	15	9.07	0.06	0.25	141	13.51	7.78
60	10	15	15.84	0.07	0.34	244	7.91	6.91
70	10	16	21.44	0.09	0.50	352	7.73	4.76
80	10	16	25.65	0.10	0.62	422	5.47	3.70
90	10	13	30.91	0.11	0.79	414	3.63	2.44
100	10	9	131.87	0.12	0.88	1196	3.82	3.64
50	20	15	14.15	0.11	0.58	223	5.01	2.31
60	20	9	23.31	0.13	0.84	218	3.19	3.10
70	20	22	24.68	0.16	1.20	573	3.46	2.36
80	20	11	35.27	0.18	1.32	404	1.30	1.12
90	20	25	69.01	0.23	1.93	1779	2.32	1.91
100	20	31	80.55	0.23	2.14	2572	2.66	2.12
50	30	11	12.54	0.18	1.03	152	3.89	3.26
60	30	28	46.24	0.25	1.35	1339	3.80	2.49
70	30	9	26.37	0.23	2.08	258	2.42	2.41
80	30	13	94.47	0.33	2.11	1260	2.49	1.95
90	30	18	188.64	0.32	2.97	3455	2.87	2.29
100	30	9	168.30	0.33	3.51	1550	2.15	2.11
50	40	17	23.62	0.23	1.64	433	3.45	2.27
60	40	14	49.34	0.30	2.62	732	3.78	2.45
70	40	9	58.85	0.42	2.70	558	2.64	2.62
80	40	21	120.21	0.38	4.17	2620	2.31	2.04
90	40	17	218.63	0.34	4.97	3807	2.61	2.07
100	40	9	211.71	0.37	4.88	1953	1.05	1.05

Table 2: Comparison of the original and modified algorithm

		Original Algorithm			Modified Algorithm			
n	m	t_{sub}	t_{FeasY}	t_{FeasX}	$\widehat{t}_{sub}$	$\widehat{t}_{FeasY}$	$\widehat{t}_{FeasX}$	imp
50	10	9.07	0.06	0.25	2.24	0.06	0.25	75.33
60	10	15.84	0.07	0.34	3.18	0.07	0.35	79.90
70	10	21.44	0.09	0.50	4.50	0.09	0.57	79.01
80	10	25.65	0.10	0.62	5.88	0.09	0.63	77.08
90	10	30.91	0.11	0.79	7.79	0.10	0.79	74.81
100	10	131.87	0.12	0.88	11.08	0.12	0.94	91.60
50	20	14.15	0.11	0.58	4.62	0.12	0.58	67.36
60	20	23.31	0.13	0.84	6.17	0.13	0.82	73.53
70	20	24.68	0.16	1.20	9.02	0.17	1.38	63.46
80	20	35.27	0.18	1.32	11.51	0.18	1.43	67.37
90	20	69.01	0.23	1.93	16.03	0.19	1.84	76.77
100	20	80.55	0.23	2.14	20.43	0.24	2.13	74.63
50	30	12.54	0.18	1.03	6.72	0.18	1.18	46.46
60	30	46.24	0.25	1.35	10.07	0.22	1.52	78.21
70	30	26.37	0.23	2.08	12.72	0.23	2.08	51.75
80	30	94.47	0.33	2.11	17.79	0.28	2.15	81.17
90	30	188.64	0.32	2.97	25.84	0.35	3.23	86.30
100	30	168.30	0.33	3.51	31.01	0.33	3.67	81.57
50	40	23.62	0.23	1.64	8.79	0.26	1.66	62.79
60	40	49.34	0.30	2.62	13.49	0.31	2.50	72.67
70	40	58.85	0.42	2.70	17.83	0.31	2.74	69.70
80	40	120.21	0.38	4.17	25.90	0.38	4.16	78.45
90	40	218.63	0.34	4.97	34.54	0.43	4.86	84.20
100	40	211.71	0.37	4.88	40.45	0.36	4.84	80.90

Table 3: Computational results for the modified Lagrangean relaxation and decomposition algorithm

n	m	n_L	$\widehat{t}_{sub}$	$\widehat{t}_{FeasY}$	$\widehat{t}_{FeasX}$	$\widehat{t}_L$	$igap$	gap
50	50	25	10.76	0.33	2.20	333	5.74	4.79
60	50	16	15.87	0.34	2.80	305	4.16	3.66
70	50	11	22.41	0.42	4.57	302	4.56	4.27
80	50	11	30.73	0.46	5.54	405	5.35	5.06
90	50	12	43.76	0.56	7.00	616	5.36	5.29
100	50	11	54.15	0.60	8.71	699	3.54	3.47
50	60	9	12.96	0.36	3.08	148	4.84	4.84
60	60	18	19.99	0.44	3.50	432	4.11	3.29
70	60	9	26.20	0.50	4.92	285	5.43	5.37
80	60	13	37.37	0.54	6.16	574	4.75	4.17
90	60	13	50.61	0.62	10.77	807	4.67	4.58
100	60	9	63.79	0.61	9.38	664	3.09	3.09
50	70	14	15.19	0.42	4.00	275	5.47	4.75
60	70	15	22.54	0.49	4.10	408	3.89	3.88
70	70	13	33.65	0.61	9.42	568	5.34	5.30
80	70	17	44.99	0.69	11.16	967	4.39	4.22
90	70	11	56.63	0.77	8.88	730	3.50	3.42
100	70	12	71.07	0.82	14.74	1041	4.16	4.13
50	80	13	18.25	0.51	4.40	302	4.78	4.24
60	80	12	26.24	0.50	53.32	961	4.82	4.37
70	80	13	35.14	0.68	16.85	685	3.95	3.82
80	80	12	49.37	0.84	19.90	842	5.88	4.73
90	80	9	63.70	1.17	20.07	765	4.74	4.73
100	80	17	82.44	0.94	13.87	1654	3.72	3.42
50	90	12	21.06	0.59	27.69	593	5.29	5.21
60	90	25	29.30	0.70	7.00	926	5.19	4.44
70	90	26	39.86	0.77	64.33	2729	4.91	4.83
80	90	28	55.48	0.90	22.28	2203	5.41	4.86
90	90	9	77.26	1.06	27.90	956	5.34	5.34
100	90	17	93.30	1.14	40.93	2302	3.99	3.90
50	100	9	23.00	0.57	7.29	279	3.92	3.92
60	100	12	32.18	0.74	11.83	537	6.07	5.87
70	100	14	44.36	0.91	13.09	818	4.89	4.60
80	100	26	63.56	0.95	23.51	2290	5.08	4.52
90	100	9	83.83	1.01	62.26	1325	4.75	4.75
100	100	9	106.77	1.12	274.03	3440	4.55	4.55