The Transcoders’ Placement Problem over Multicast Networks

Asaf Henig
The Transcoders’ Placement Problem over Multicast Networks

Research Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Asaf Henig

Submitted to the Senate of the Technion — Israel Institute of Technology

Shvat, 5766 Haifa Feb 2006
The Research Thesis Was Done Under The Supervision of Prof. Danny Raz in the Faculty of Computer Science

THE GENEROUS FINANCIAL HELP OF THE TECHNION IS GRATEFULLY ACKNOWLEDGED.
# Contents

Abstract ................................................................. 1  
Abbreviations and Symbols ............................................. 3  
1 Introduction .......................................................... 4  
2 Related Works ......................................................... 9  
3 The Transcoders’ Placement Over Multicast Networks (TPM) Problem ................................................... 12  
3.1 The Multi Layered Mixed Graph (MLMG) Model ................. 14  
4 Complexity Analysis .................................................. 17  
5 Approximation Algorithms For The TPM Problem ................. 24  
5.1 The Bounded Format (BF) Algorithm ................................. 25  
5.2 Compression only system ............................................ 33  
5.3 Symmetric conversion system ...................................... 37  
6 Approximation Algorithms For Extended TPM Problems ......... 41  
6.1 Multiple origin formats scenario .................................... 41  
6.2 Capacitated TPM (CTPM) ........................................... 42  
6.2.1 Link Capacitated TPM (LCTPM) ................................. 42  
6.2.2 Transcoding Capacitated TPM (TCTPM) ......................... 45  
7 The Degenerated Version Of The TPM Problem .................... 49  
7.1 An optimal cost algorithm for DTPM on trees ..................... 50  
7.1.1 Calculating the cost of cell ((X,Y),(W,Z)) ....................... 52  
7.1.2 DTT running time ................................................. 56  
7.2 Approximating DTPM by HST trees ............................... 57
8 Performance Evaluation 60
9 Conclusion And Open Problems 66

Appendices 67

A Building The DTT Matrix Cell 68
A.1 Local optimal tree . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
A.2 Cost of sending the media object downstream . . . . . . . . . . . . . 69
A.3 Adding the upstream formats . . . . . . . . . . . . . . . . . . . . . . 71

B Reducing The DTT Matrix Size 72

C Improving Step B In The DTT Algorithm 75

Bibliography 77
List of Figures

3.1 A Multi Layered Mixed Graph (MLMG) representing a TPM instance . . 15
4.1 A reduction from the SC/HS to the TPM . . . . . . . . . . . . . . . . . . 19
5.1 MLMG compression towards layer F . . . . . . . . . . . . . . . . . . . 29
5.2 A tight example for the homogenous compression TPM . . . . . . . . . 37
6.1 LCTPM - reduction via the SFBC problem . . . . . . . . . . . . . . . . . 45
6.2 TCTPM - reduction via Hamiltonian Path . . . . . . . . . . . . . . . . . 47
7.1 An example DTT matrix for $F = 2$ . . . . . . . . . . . . . . . . . . . . 52
7.2 An infeasible DTPM solution . . . . . . . . . . . . . . . . . . . . . . . . 53
8.1 A simple example for the TPM problem . . . . . . . . . . . . . . . . . . 62
8.2 The U.S.A metropolitan graph used for the second sets of tests . . . . . 63
8.3 TPM Algorithm cost as a function of the amount of open transcoders . . . 64
8.4 Comparing the BF algorithm to the currently used schemes . . . . . . . . 65
Abstract

Scalable and efficient multicasting of multimedia information is a very challenging task. This is especially true in the emerging new Internet where users use devices such as smartphones and PDAs, a considerable amount of them are connected via wireless connections, and peer to peer applications are becoming more and more popular. In this new environment, the multimedia format that should be sent to different users varies considerably and sending a media stream to a set of users often involves transcoding of formats.

This work addresses the problem of controlling multicast streaming in this new environment by defining a framework in which transcoding can be done in internal network nodes, and not necessarily at the sender’s or at the receivers’ ends. In this framework the sender retrieves all the information regarding the transcoding abilities of the various nodes and the characteristics of the links. Then, it needs to decide how to broadcast the multimedia stream, in what formats, and where to perform the needed transcoding.

In this work we focus mainly on the algorithmic aspects and the computational complexity of the problem in hand using formal theoretic tools. We first formulate the problem of multicasting a media stream in a heterogenic environment as an optimization problem and show that it is NP-hard. Then we study the difficulty of approximating this problem both by proving approximation algorithms for different variants of the problem and by proving lower bounds on the possible approximability. We show that though the problem is hard to approximate in its general case, it is possible to find a close to optimal solution for the practical case where the number of relevant formats, supported in the system, is small.

Since efficient multimedia multicasting is a very practical problem we also study the actual performance of our algorithms using simulations. We compare our general case algorithm to the currently used approaches which allows transcoding only at the sender’s or at the receivers’ ends. Our simulations’ results indicate that performing transcoding
at intermediate nodes is indeed efficient, and that our algorithm can find a much better streaming scheme than any other known paradigm.

**Keywords:** Transcoding, Approximation algorithms, Lower bounds, Steiner tree.
Abbreviations and Symbols

ADSP - Acyclic Directed Steiner tree Problem
BF - Bounded Format
DSP - Directed Steiner tree Problem
HST - Hierarchial well Separated Tree
HS - Hitting Set
MLMG - Multi Layered Mixed Graph
PCP - Probabilistic Checkable Proofs
PDA - Personal Digital Assistant
P2P - Peer to Peer
TPM - Transcoders’ Placement over Multicast networks
SC - Set Cover
SNP - Strict NP

\( \mathcal{F} \) - The set of formats recognized in our model
\( f_i \) - The compression value of format \( i \in \mathcal{F} \)
\( K \) - Maximal compression ratio in the TPM model (\( K = \frac{f_x}{f_i} \leq 1 \)).
\( S_i \) - Optimal Steiner Tree created in the MLMG level \( i \)
\( P_i \) - A shortest path connecting level 1 to level \( i \) in the MLMG
Chapter 1

Introduction

One of the important functions of the Internet is the capability to deliver live audio and video streams to users at high quality, fast speed and most important - low cost. Delivering live streams across the Internet is becoming more and more popular as computers gradually replace other media and entertainment channels such as radio, television, newspapers etc. Newly developed standards and encoding technologies, such as AVI and MPEG, have made it possible for users to receive television-quality video with various data rates and various bandwidth consumptions. In today’s heterogeneous environment, end-users use different devices such as laptops, PDAs, smart phones and others. In this environment, sending media objects to clients forces the sender to supply the data in different formats to be properly accommodated by the end users’ devices. It is reasonable to assume that the multimedia information was not initially encoded in all formats, therefore, in order to suit the clients’ devices in terms of CPU availability, screen resolution, etc., the multimedia data should somehow be transcoded.

There are two additional characteristics of the Internet that are extremely relevant to this discussion. The first is the increasing number of wireless and cellular users that are connected to the Internet. Most of the small devices (laptops, PDAs, and smart phones) are connected either through WiFi (802.11) or various 2.5G and 3G cellular technologies. The wireless connection bandwidth and other parameters such as loss and jitter have a crucial impact on the quality of a media stream delivered over this connection. Another important aspect is the move toward overlay networks and Peer to Peer (P2P) applications. More and more applications (currently mainly file sharing applications but in the future also multimedia applications) are based on the P2P paradigm where information is transferred between end users and not between users and well defined servers. When a
media stream is sent from a user to a set of other users, one can no longer assume (as in the case of a well equipped server) that the sender can transcode the data to all possible formats.

Transcoding is defined as a transformation that converts a media object from one form to another, frequently trading off object fidelity for size [36]. It can be executed at various components in the network such as servers, proxies, and clients. At each component, the cost of the format transformation is different. Placing transcoders at the end-users preserves the network architecture and transfers responsibility of the format transformation to the clients. This paradigm usually involves the option of downloading codecs. The share use of different multimedia applications by different clients has encouraged the industry to develop small software packages that can be downloaded to the users’ devices and help them convert received media to appropriate formats. However, this option can be very expensive, and in some cases even infeasible, due to limitation of computational power which may exist in many devices held by different clients. Alternatively, putting the responsibility for reformatting at the server’s hands can also be an expensive option causing the server to store all formats to each file and wait for clients’ requests. If the server chooses to transcode the files only when clients’ requests are issued, this can cause a considerable delay in delivering time. We should also remember that servers are usually one of the networks’ bottlenecks, especially when media delivering is considered, thus loading them with expensive computational tasks before serving each request only increase their instability. Currently, the internet supports only the above mentioned approaches which allows transcoding to take place only at the network edges i.e servers or clients. These approaches are relatively simple to manage and requires no complicated protocols or sophisticated management system to operate well. A third place for performing transcodings in the network, which tries to benefit from the advantages of the above mentioned approaches, are intermediate nodes (mostly proxies and even routers in the future). Performing media conversion inside the network and not at its edges benefits from both the high computational power of these network elements, compared to possible “weak” devices that the clients may hold, and the relatively balanced load on these components, compared to the obvious overload on the server. The paradigm of intermediate network media conversion was explored extensively in the past only for unicast sessions (see for example [22, 27]). The main assumptions that follows this paradigm are that intermediate nodes have the capability to transcode requested media objects according to the end-users specifications and that building a manageable system to orchestra all
components in the system is feasible.

The advantages and disadvantages for each transcoders placement option, as we have described above, become even more relevant when discussing multicast sessions. The need to send the same data to a group of clients allows us to better utilize the network resources by spreading the data via the most lightweight tree which connects the media source to all designated clients. This type of connection is called multicast. Choosing wrong network nodes to perform transcoding can considerably diminish the advantage of using this kind of session. Generally speaking, in order to decide where to perform the transcoding, and how to multicast the multimedia stream, it is wise to look at this problem as a simple matter of “cost”. The cost of sending a media object to users is composed of two components: First, the communication cost of transmitting the streams over the network with the appropriate formats. Second, the cost of the transcoding operation aiming at supplying the user with his/her compatible format. The transcoding cost is actually a function of three parameters, the two formats that the transformation is dealing with (the input and output formats), and the network node in which the transcoding is taking place. Since not all intermediate nodes have the same computational power, it is reasonable to assume that a specific transformation would use a higher percentage of CPU cycles or would take longer when conducted at a certain node compared to other nodes, meaning, it costs more. The third element that may have influence on the overall cost is the storage of the media file. This element is less important when dealing with live streams and thus, is not regarded as part of the overall cost in this research.

In order to decide how to multicast the media object and which transcoders in the network to use, one has to define methods through which network information, such as the nodes’ transcoding capabilities and link parameters (possible bandwidth, latency,...), can be gained prior to running any decision algorithm. Services such as IDMAPS [31] that provides information about network distances and links characterizations can be used in order to retrieve network information. “Discovery protocols” (like those described in [18, 42]) can be used in order to retrieve transcoding capabilities of both the receivers and the internal network nodes. Using these services, the source, who wishes to distribute the data, can discover the capabilities of the different clients and decides which media formats is most suitable to serve each client. Though the “discovery phase” is an important part in any implementation, which considers a heterogenic environment, we do not deal with this phase and start our discussion from the point in which all network information is already known to the source.
In this thesis we focus mainly on the algorithmic and computational aspects that lie in the heart of designing a multicast supporting system in a heterogenous environment. Note that the general question of optimally multicasting a media stream while delivering it in a suitable format to each client consists of many parts: First one has to find the exact network links on which the stream should be sent. Then, one has to choose the network nodes in which transcoding should take place and which media conversion to perform in each one of them. Eventually, one has to determine the exact format each client should receive out of the subgroup of formats the client supports. Though, as we can see, a full solution should answer many questions, we propose in this thesis a novel approach that solves all of them simultaneously by formulating one single optimization problem. This optimization problem is formally described using a novel graph model we named “MLMG” (multi layered mixed graph) which is developed in order to better describe the flow of data over the heterogenic network. The exact formulation of the optimization problem and the MLMG graph are described in chapter 3 of this thesis.

Following the construction of the MLMG we prove that our optimization problem is NP-hard and actually no constant factor approximation exist for most of its variants. Using basic tools from computer science theory we examine the problem’s inapproximability and prove that it has a logarithmic factor lower bound both in the cardinality of the group of clients - \( C \) and the group of formats - \( F \). We continue to examine the general case of our problem by suggesting several algorithms for gaining close to optimal solutions. We first propose an immediate approach that grantees only factor \( n \) approximation ratio, where \( n \) is the cardinality of \( C \), and continue with the construction of several factor \( F \) approximation algorithms where \( F \) is the cardinality of \( F \). The discussion on the problem’s complexity and the lower bounds proofs are presented in Chapter 4 of this thesis.

After dealing with the general case formulated in Chapter 3, we continue to examine some of the problem’s subcases. The general problem’s algorithm and the algorithms for its subcases are presented in Chapter 5. In Chapter 6 we explore extended versions of the problem in which additional constraints, such as the amount of media formats that can simulatively traverse the same network link or constraints on the amount of conversions each network element can perform, are introduced. We prove, using gap preserving reductions, that in both cases no algorithm can provide any guaranteed approximation ratio. In Chapter 7 we examine a very practical case in which the amount of media formats supported in the system is a small constant thus enables us for the first time to
reduce the linear approximation factor to a logarithmic one. The algorithm we construct uses HST trees developed recently by Y. Bartal [7].

Since proving approximation ratios for our algorithms can only indicate on their behavior in the worst case, we also examine the algorithms’ actual performance in a more empirical manner. In Chapter 8 we examine several practical scenarios for the general case algorithm using simulations. We demonstrate using both a very simple network and a network representing the US backbone that our approach, which chooses transcoding at intermediate nodes, is indeed beneficial. By comparing our novel approach to the currently used paradigms which performs transcoding only at the network edges, we show that our algorithm outperforms the trivial techniques and can find a transcoding schemes with a much lower cost.

The main contributions of this thesis is the development of the novel approach for placing transcoders at internal nodes, the development of the MLMG model, the approximation algorithms with the proofs of their guaranteed performance ratio, and the demonstration of the applicability of the general case algorithm.
Chapter 2

Related Works

The problem of efficient transcoding was addressed by many works. Papers dealing with this problem can be divided into three groups: papers dealing specifically with encoding techniques, as in the case of converting MPEG-2 to MPEG-4 [28] or the development of a low bit rate codecs [35], papers dealing with systems’ architecture as in the case of TeC (Transcoding enabled Caching) [8], and papers dealing with efficient proxy placement on the path between the server and its designated clients [15, 36]. Our work belongs mainly to the last group.

Previous research on proxy placement concentrated mainly on unicast sessions [15, 34, 36]. These papers followed a theoretic model which is different then the one we suggest, mainly because of their unicast characterizations. Moreover, these works did not deal with the unusual problem of transcoders’ placement where the use of transcoders may change the volume of the stream drastically and thus the cost of sending it over the network links. Recently, Dabran et al. [18] studied management aspects of multimedia email attachments. They studied a similar problem to the one discussed in this thesis where the sender has to decide whether to transcode the attachment file at his own mail server or alternatively to send it in its original format to be converted at receiver’s mail server. Though we adapt some of the formulation and modeling from this work, in our problem we allow transcoding to take place at intermediate nodes, thus, the network topology has an important part in our input. Moreover, in [18], the authors used a non-polynomial algorithm to search for the best possible transcoding scheme. Here, since we have many receivers and not just one, the problems complexity increases, thus, making exhaustive search infeasible and we are left with the need to develop a polynomial time approximation algorithms.
As for the use of multicast connections, initially researchers have focused on analyzing the general concept, its advantages and disadvantages [44]; however, lately, research has shifted towards the question of special components’ deployment both in its totally system based analysis [39] and in its algorithmic perspective [1, 41]. Our problem of efficient transcoders’ placement is related to this last group of papers. However, unlike most problems in that group, in this work we have assumed that all nodes in the network have a multicast capability, i.e. the capability to create a point to multipoint connection. Our goal is not to make sure this capability is maintained (as is the focus of the problems mentioned above) but to extend it with an additional value of flow conversion.

Theoretically speaking, the problem this thesis deals with is related to two well known groups of optimization problems. First, the issue of transcoders’ placement is related to a group of optimization problems called “Facility Location”. Problems in this group deal with the question of efficiently placing facilities in a plane or in a graph, in a way that will best serve potential clients of those facilities. Some of the well known problems in this group are the “K-median” [3], the “K-center” [30] and the “Facility location” [25] all three were investigated thoroughly in the past. Since our problem asks to place transcoders in a network, or more precisely, to choose the nodes in which transcoding should take place, its connection to the Facility location group of problems is clear. However, some significant properties of the problems in this group differs strongly from main features found in our problem. For example, in both the K-median and the Facility location problems, formulated on a graph, part of the objective function is to minimize the overall sum of distances between each clients and its closest facility. However, if some of the edges between clients and facilities are shared by more than a single client, the cost of using these edges will also be counted more than once. Note that when multicast connection is concerned, as dealt with in our problem, even if a stream traverses over a single edge eventually serves more than a single client, the cost of using this edge will always be counted once. Due to this, and to some other differences, we have decided to concentrate in this thesis on a slightly different model than the one shared by the group of facility location problems. This alternative model is more similar to the one used by a group of optimization problems called the “Steiner Tree”.

The use of multicast connections, which is a key element in our problem, minimizes the cost of sending the same data to a group of users. This minimization can be accomplished by sending the data via a tree with a minimum cost edges which connects the data sender to all designated clients. The question of how to build such a tree is the Steiner
Tree problem. This is actually a group of problems which has many variants, all were investigated extensively in the past [33, 46, 49]. Though our problem asks not only to create a multicast connection for streaming media, but also to choose a way to supply this media in different formats, we show in Chapter 4 that this complicated problem can be presented as different variants of the Steiner tree. In this thesis we encounter mainly two versions of the Steiner Tree problem: the undirected and the directed versions, both ask to create a minimum spanning tree connecting subgroup of nodes either in an undirected or a directed graph. Another version of the Steiner tree problem is closely related to the problem we are dealing with; this is the “Priority Steiner Tree” (PST) problem [13, 16]. The importance of the PST, compared to other Steiner Tree versions, is its focus on network heterogeneity which has a vast impact in multimedia streaming.
Chapter 3

The Transcoders’ Placement Over Multicast Networks (TPM) Problem

We deal with multicasting a media object from its source to a group of recipients over a network. We assume that each node in the network has a multicast capability (can clone an incoming packet and send the copies on all outgoing links) and some network nodes can host transcoders and thus are able to perform transcoding operations. We study the question of optimally placing transcoders in the network in a way that allows all users to receive the multimedia data in their required format, at the lowest possible cost. We name this problem the Transcoders’ Placement over Multicast networks, or in short, the TPM problem. The TPM problem is an optimization problem in which we wish to minimize the overall cost of delivering the data from the source to all clients. The cost of the data transmission includes both the communication cost and the transcoding cost as explained before. Note that since different recipients may need to receive the media in different formats, the optimal multicast tree can be different from the one chosen for a simple layer 3 multicast protocol. We assume that the media origin (the source) has the ability to learn about its clients devices’ capabilities (i.e. which media formats may suit each client), about the capabilities of the intermediate nodes to transcode the media streams, and about the network structure and the cost associated with using each link.

We deal with the TPM problem using a common model in computer science. In this model the network, in which the multicast connection should take place, is formulated using a simple undirected graph. The graph’s nodes represent both regular clients’ devices (personal computers, PDAs, etc..) and other crucial network elements (routers, switches, proxies, etc..). The weighted graph’s edges represent network links and their compatible
cost. If one uses the entire link capacity for transmitting a media stream he will pay the full link price. If one uses only part of that capacity he may pay less than the full price according to the amount of bandwidth he uses. The formal definition of the undirected edges cost is explained below. Recall that in the TPM problem network nodes may have the ability to transcode incoming multimedia data and convert its format. These nodes’ conversion ability is presented in our model using converting graphs. The converting graphs are directed weighted graphs in which the nodes represent different media formats recognized in our system. The weighted directed edges of these graphs represent transcoding abilities and their compatible cost. Such graphs appear in the right hand side of Figure 3.1 (the graph found at the left hand side of the figure will be explained later). Since each network node can have different converting capabilities and different conversion costs, each such a node is associated with one specific converting graph. The left-most converting graph in Figure 3.1, for example, is associated with network node $t_1$. According to this graph node $t_1$ has the ability to convert media format $f_1$ to formats $f_2$ and $f_3$, and to convert format $f_3$ to format $f_4$.

Following the above explanation about the different graphs comprising our model, we can now formulate more precisely the TPM problem. We would like to build a minimum cost spanning tree over the undirected graph which connects the media source to all clients. If a client requires different media formats from the one in which the media stream was initially encoded, the path to this node in the created spanning tree must traverse a network node that has transcoding capabilities compatible to the new required formats. The solution’s cost consists of the cost of the tree and the cost of the directed arcs chosen in some of the network nodes converting graphs. It is possible that in order to minimize the overall solution’s cost the solution includes several transcoding operations between a source and a designated client. Moreover, it is possible to traverse the same network link several times with different media formats. The TPM problem is thus formulated as follows:

**The TPM Problem Definition:** We are given an undirected weighted graph $G_{TPM} = (V, E)$, which represents a multicast supporting network. A set of nodes $C \subset V_{TPM}$ representing the clients, and a root node $r \in V_{TPM}$ representing the media source. We are also given a group of weighted directed graphs each representing a format conversion ability of a network node (and thus it is called the node’s converting graph). Each converting graph is formulated in the following way: $G_{convert} = (t, V, E)$ where $t \in V_{TPM}$ is the network node associated with graph $G_{convert}$ and the group of nodes $V_{convert}$ in this
graph are actually different media formats supported in our system. Each client in the network has the capability to receive the media in a subset of formats from a large group of formats $\mathcal{F}$ recognized in our world.

The weight ($c_e : E \to \mathbb{R}^+$) of each $G^{TPM}$ edge represents the basic communication cost of using that edge (link). The actual communication cost depends on the media format sent on the link. It is reasonable to look at the relationship between the basic link cost $c_e$ and this media format as a simple multiplication function: $w_e = c_e \cdot f_i$, where $f_i$ represents the compression value of format $i \in \mathcal{F}$ sent on the link. We can assume that the original media format has a compression value of 1 ($f_1 = 1$) and all other formats represent a more compressed encoding of $f_1$, thus, $\forall i f_i \leq 1$.

The TPM objective is to create a minimum weighted tree $T^{TPM}$ which connects the root node $r \in V^{TPM}$ to each client, and traverses over transcoders located in some of $V^{TPM}$ nodes. The final format reaching each client should be one of the client’s subgroup of compatible formats, and can be created by using converting graphs of the network nodes.

### 3.1 The Multi Layered Mixed Graph (MLMG) Model

A full solution for the TPM problem includes many parameters: The shape of a spanning tree over $G^{TPM}$, a group of nodes in $V^{TPM}$ that perform the transcoding, the media conversion that takes place in each chosen transcoder and eventually the exact format to be delivered to each client.

Next we introduce a new model for a given TPM problem which includes all formal TPM inputs. The shape of a single spanning tree over this model will automatically define all above mentioned parameters. We named this new presentation the “Multi Layered Mixed Graph (MLMG) model”. A mixed graph is a graph with both directed and undirected edges [19].

The MLMG model is described in Figure 3.1. Each layer in the MLMG has the same structure as the original undirected $G^{TPM}$ graph. The layers are connected to one another by directed arcs that connects copies of the same node from the original undirected graph. The arcs are the edges of the node’s converting graph. The MLMG’s layers represent the different media formats and thus the amount of layers in the graph is exactly the amount of formats in $\mathcal{F}$. Note that copies of the same undirected edge in different layers
Figure 3.1: A Multi Layered Mixed Graph (MLMG) representing a TPM instance

have different cost as a function of the exact layer in which it is found. This cost is a multiplication of the basic link cost $c_e$ and the compression value $f_i$ where $i \in \mathcal{F}$ is also the layer’s index in the MLMG and $\forall_{i=\{1,...,\mathcal{F}\}} f_i \leq 1$. It is reasonable to order the layers in the MLMG in an increasing compression form where $1 = f_1 \geq f_2 \geq ... \geq f_{\mathcal{F}}$. Following this form the graph’s top layer represents the media origin format and each layer beneath represents a compressed format.

The media source in the MLMG is the root node copy found at the top layer (also represented as $r^1$ - the black node in Figure 3.1). The clients ($C \subset V^{TPM}$) are the dummy nodes found to the left and right of the main MLMG structure. These dummy nodes are connected by zero weight directed edges to the copies of a recipient in the layers compatible to the formats accepted by this recipient. In Figure 3.1, for example, node $t'_3$ represents client $t_3$ and since $\mathcal{F}_{t_3}$ (the formats supported by client $t_3$) equals $\{f_1, f_3\}$ $t'_3$ is connected to the compatible nodes in formats $f_1$ and $f_3$. A solution to the TPM problem over the MLMG is a minimal cost tree connecting the top layer’s root node to each one of the dummy nodes representing the clients. Clearly there is a one to one mapping between a regular TPM spanning tree, which includes traversing on converting graphs, and a spanning tree created on the formulated MLMG, hence, the following claim is obvious:

**Claim:** Any optimal solution for the TPM problem over the MLMG, has the same cost as the optimal TPM solution on a regular network representing graph.
Note that a spanning tree created on a MLMG reveals all needed information for a complete and full solution of the TPM problem. If we focus only on the spanning tree’s horizontal undirected edges (those picked in the different MLMG layers) we can conclude which network links were chosen to deliver the media object to all clients and what format is sent on each link. If we focus on the vertical directed arcs of the spanning tree (those connecting the different layers) we can conclude which network nodes were chosen to perform transcodings and which format conversions actually takes place. Eventually, if we concentrate on the directed arcs connecting the dummy node we can find out which format was delivered to each recipient.
Chapter 4

Complexity Analysis

In this chapter we analyze the computational complexity of the general TPM problem. That is, we are referring to the TPM problem as it was defined in the previous chapter without weakening any of its parameters. In later chapters in this thesis we examine subcases of the TPM problem and reveal different complexity characterizations. This analysis can help understand the hardness of achieving an optimal solution or even a good approximation ratio for the minimization problem in hand, and thus it is a good starting point for this study.

First, using a simple reduction from the “Steiner Tree problem” on undirected graphs, it is easy to see that the TPM problem is NP-hard. The Steiner tree problem is NP-hard and remains so for many of its versions (a discussion on several well studied version can be found in [29]). A TPM problem with only one media format ($|F| = 1$) which is accepted by all clients and sent by the source is exactly the undirected Steiner tree problem. Since the TPM is NP-hard we cannot hope to construct an algorithm that finds an optimal solution to TPM in polynomial time. An obvious approach, therefore, is to use an approximation algorithm in order to find, in polynomial time, only a sub-optimal solution in terms of cost. The following two definitions are useful for analyzing the problem’s complexity.

**Definition 4.0.1** An algorithm $A$ is an $\alpha(n)$ − approximation algorithm for a minimization problem $P$, if for all instances of $P$ the cost of the solution produced by $A$ is no more than $\alpha(n)$ times the optimal solution. In this case $\alpha(n) > 1$ and $\alpha(n)$ is also known as the algorithm’s approximation/performance ratio.

**Definition 4.0.2** An optimization problem $P$ is said to be hard to approximate (or inap-
proximable) within a factor $\beta(n)$, if the existence of a $\beta(n)$ approximation algorithm for $P$ implies a theoretic implication that is regarded as highly unlikely (such as $P = NP$).

When discussing hardness of approximation or lower bounds for an optimization problem, one refers to the fact that there is some lower bound for approximating the problem within a guaranteed performance. Following the important work of Papadimitriou and Yannakakis [43] who first defined the class of Max-SNP problems, and the pioneering work of Arora et al. [2, 4] on the PCP theorem, it was shown that the large group of Max-SNP optimization problems have a constant $c > 1$ inapproximability bound. In other words, we cannot hope to find polynomial time approximation algorithms for problems in this group that can guarantee a performance ratio better than $c$ unless $P = NP$.

The Steiner tree problem on undirected graphs was shown to be Max-SNP-hard [37] and its approximation lower bound was proved to be at least $\frac{5601}{5600}$ [17]. Since the Steiner tree problem on general undirected graphs is a subcase of the TPM, this lower bound holds automatically to our problem.

We will now prove a tighter lower bounds for the TPM based on simple reductions and some reasonable complexity assumptions (not necessarily $P \neq NP$).

**Theorem 4.0.1** The TPM problem cannot be approximated better than $\ln F$ or $\ln n$ provided that $NP \not\subseteq DTIME(n^{\log \log n})$. Here, $n$ is the number of designated clients and $F = |F|$.

The above theorem follows from an approximation preserving reductions using the SET COVER (SC) problem [32] and the HITTING SET (HS) [32] problem. Let us first define the SC problem:

**Definition 4.0.3** SET COVER: Given a ground set $S$ of $n$ elements and $C = S_1, S_2, \ldots, S_m$ a collection of subsets of $S$, find as few as possible subsets from $C$ such that every element from $S$ is contained in at least one of the selected subsets.

The SC problem was among the first problems for which approximation algorithms were analyzed. Though many algorithms were given to the problem, none has succeeded in improving the trivial $\ln n$ performance ratio produced by the simple greedy algorithm. In 1998, following the work of U. Fiege [21], the SC has become one of the few problems for which the known approximation ratio is tight. Fiege showed that we cannot
hope to achieve a better than a $\ln n$ approximation unless problems in $NP$ have $n^{O(\log \log n)}$ deterministic algorithms (i.e. $NP \subseteq DTime(n^{\log \log n})$). The following approximation preserving reduction from the SC proves that the TPM problem holds the same lower bound.

Given an instance $I_{SC}$ of the SC problem, we build a three layered graph to formulate a TPM instance, $I_{TPM}$. The graph’s top layer is a single source, representing the media origin. The second layer is a group of nodes $M$, with a cardinality $m$ (the size of $I_{SC}$ collection of subsets). Each node in $M$ is connected to the root by a by zero weight undirected edge. The third layer in $I_{TPM}$ is a group of nodes $C$, with cardinality $n$ (the size of $I_{SC}$ ground set), representing the clients. A node $i \in C$ is connected by a zero weight undirected edge to node $j \in M$ iff element $i$ in $I_{SC}$ ground set is contained in $I_{SC}$ subset $j$.

The system’s group of media formats $\mathcal{F}$ consists of $m+1$ formats (recall that $m$ is the number of subsets in $I_{SC}$) thus, $\mathcal{F} = \{f_0, f_1, f_2, \ldots, f_m\}$. The root node $r$ sends a media stream in the format $f_0$ and each client in $I_{TPM}$ can receive it in format $f_j$ if it is connected to node $j$ in group $M$. For example, if element $i$ in $I_{SC}$ is contained in subsets 3, 4, 7 then according to our construction in $I_{TPM}$ client $i$ is connected by zero weight edges to nodes 3, 4, 7 in group $M$. Furthermore client $i$ wishes to receive the media object in formats $f_3, f_4$ and $f_7$. In $I_{TPM}$, the nodes belonging to group $M$ are the only ones with transcoding capabilities. Each node $j \in M$ can convert format $f_0$ to format $f_j$ and all conversions cost are one. The reduction we have just described appears in Figure 4.1.

Following this reduction it is easy to show that every $I_{SC}$ solution, including of course
the optimal solution, induces the same cost solution to $I_{TPM}$. A solution of cost $d$ to $I_{SC}$ means that all elements in the ground set were covered by exactly $d$ subsets. In order to produce the same cost solution for $I_{TPM}$, let us chose the compatible nodes in $M$ to perform transcodings, i.e. the nodes in $M$ which represent these exact subsets. Note that these nodes host transcoders that convert format $f_0$ to formats in $\mathcal{F}$ accepted by all clients in $I_{TPM}$. Since the overall cost of the TPM’s tree consists only of the transcodings cost (all edges in $I_{TPM}$ have zero weight), the overall format conversion has the same cost as the cardinality of the subgroup in $M$ which performs the transcoding i.e. $d$.

In the opposite direction, assume that there is an $I_{TPM}$ solution with a $d$ cost. According to our construction it means that the solution contains at most $d$ formats’ conversions from $f_0$ to a group of $f_i$s. These $d$ new formats are accepted by all clients and thus the clients have an edge connecting to the compatible nodes in group $M$. We use the chosen formats to indicate which subsets in $I_{SC}$ can be selected in order to cover all elements in that instance. Again, we chose no more than $d$ subsets to cover the entire ground set represented by the $I_{TPM}$ clients.

The constructed reduction is approximation preserving. A $d$ cost solution to any $I_{SC}$ will correspond to an equivalent cost solution to the compatible $I_{TPM}$. Suppose we had a better than $\ln n$ approximation algorithm to the TPM problem we could have used it to solve any $I_{TPM}$ instance, specifically those created by our reduction. Using this algorithm we could always create a better than $\ln n$ approximation to the SC problem indicating that $NP \subset DTIME(n^{\log\log n})$ which is considered highly unlikely.

Using almost the same construction, we can prove that the TPM problem has also a lower bound of $\ln F$, ($F = |\mathcal{F}|$). The proof follows from an approximation preserving reduction via the Hitting Set (HS) problem:

**Definition 4.0.4 Hitting Set:** Given a finite ground set $S$ of $n$ elements and $C = S_1, S_2, ..., S_m$ a collection of subsets of $S$, we would like to find a minimum cardinality subset $S' \subseteq S$ such that $S'$ contains at least one element from each subset in $C$.

The HS and the SC problems are very similar. In fact, it was proven that both problems are equivalent and that approximation algorithms and non-approximation results for the SC can carry over to the HS [5], thus, the HS problem also has a lower bound of $\ln n$ where $n$ is the ground set cardinality.

To prove the $\ln F$ inapproximability result we use the following reduction: Given
an instance - $I_{HS}$ of the HS problem, we construct the same graph for a TPM instance $I_{TPM}$ as we did previously. This time, the second layer nodes, which represent possible transcoders placements, have the size of the $I_{HS}$ ground set (in other words, here $n = F$). The third layer nodes, representing the designated clients, now have cardinality compatible to $S$ subsets in $I_{HS}$. All edges in this reduction follows the same characteristics as the previous reduction, i.e., all edges have zero weight, source connected to all layer two nodes and all layer two nodes are connected to compatible layer three nodes. The media conversion cost is 1 and each node from group $M$ (layer 2) can convert format $f_0$ to the compatible format $f_i$.

This reduction’s proof is similar to the proof of the previous reduction. The small differences between the two reductions are two-folds: First, in the reduction via the HS problem, the ground set $S$ represents the group of media formats $F$ and not the group of designated clients. Second, the elements we are choosing in the later reduction are taken from the ground set itself and not from the group of subsets, as it was done in the SC. Following the above reduction from the HS problem, and its obvious proof, we can claim that the TPM problem cannot be approximated within $(1 - \varepsilon)\ln F$ for any $\varepsilon > 0$. This claim concludes the proof for Theorem 4.0.1.

Note, that the reductions we have used indicate that one of the characteristics of the TPM problem, that makes it inherently difficult, is that each client recognizes a subgroup of formats in the system. Following this observation, in order to minimize the solution’s overall cost one must find a minimal cardinality subgroup of formats which are recognizable by all clients. We now present a different reduction that proves a tighter lower bound for the TPM problem than the lower bound proved above. Our interest in this reduction is not the new lower bound but rather the presentation of a different TPM characteristics that makes the problem difficult to approximate.

**Theorem 4.0.2** The TPM problem cannot be approximated better than $\ln^2 F$ provided $NP \not\subseteq ZTIME(n^{polylog n})$, where $ZTIME(t)$ denotes the class of languages that have a probabilistic algorithm (with zero error) that runs in expected time $t$.

We prove Theorem 4.0.2 using the MLMG construction and a simple reduction from the Directed Steiner tree Problem (DSP). The DSP is a version of the Steiner tree problem for directed graphs. In this version one needs to connect a root $r \in V$ to all terminals via a minimum cost directed tree. Halperin et. al. [26] proved that the DSP has a $\ln^2 V$
Given an instance - $I_{DSP}$ of the Directed Steiner tree Problem, with a root $r$ and a group of terminals $C \subset V$, we regard this instance as the transcoding graph of a root node in a compatible TPM problem instance - $I_{TPM}$. $I_{TPM}$ network topology is a simple star shape graph in which the number of nodes is at least as the amount of terminals in $I_{DSP}$. The root in $I_{TPM}$ has the only transcoding capability which is represented by $I_{DSP}$ itself. All edges in $I_{TPM}$ network topology, which connects the clients to the root, have zero cost. Let each client in $I_{TPM}$ demands a different media format represented by the different terminals of $I_{DSP}$.

It is obvious that the above simple reduction is an approximation preserving. A $d$ cost directed Steiner tree which connects the root to all terminals in $I_{DSP}$ induces a $d$ cost TPM solution by following the same tree on the $I_{TPM}$ root converting graph. This tree helps us create all needed formats for $I_{TPM}$ clients with exactly the same cost $d$. This proves theorem 4.0.2.

It is important to note the following three observations about the difficulties of the TPM problem revealed from the above reduction via the DSP:

**A)** The network topology has no significance in proving the $\ln^2 F$ lower bound. In the above reduction we used a star shape graph to represent the network, however, the same threshold result could be proven using any other network topology provided the number of network nodes is at least as the number of DSP designated terminals.

**B)** In the above reduction it was sufficient that each $I_{TPM}$ client would recognize a single specific format $f \in F$ and not a subgroup of formats.

**C)** The reduction does not change even if the initial directed Steiner tree is acyclic and even if all arcs cost the same. The acyclic directed Steiner tree problem (ADSP) is a subcase of the DSP for acyclic directed graphs. Using a simple reduction from the Set Cover problem it is easy to see that the ADSP has a $\ln n$ lower bound where $n$ is the amount of designated terminals (for the ADSP problem a tighter lower bound such as $\ln^2 n$ is not known currently). A scenario in which the system holds a single format conversion graph which is also acyclic describes a system with compression capabilities only. We elaborate on such a system in the next chapter. However, this proves that our TPM problem, even on a compression only system, maintains its $\ln F$ lower bound.

The reduction from the DSP and the above observations reveal that the difficulty of
approximating the TPM problem lies both in the fact that each client may recognize a subgroup of formats and also in the use of directed converting graphs. We use this characteristic of the problem in order to prove additional results in other chapters of this thesis.
Chapter 5

Approximation Algorithms For The TPM Problem

In the previous chapter we showed hardness of approximation for the general TPM problem described in Chapter 3. Following that discussion, in this chapter we present polynomial time approximation algorithms both for the general problem and for several of its simple subcases. Our algorithms run on the Multi layered mixed graph (MLMG) which is created from a given TPM instance. The algorithms produce an $O(F)$ approximation ratio where $F = |F|$ is also the amount of MLMG layers. Note that there is no contradiction between this $O(F)$ approximation guarantee and the $\ln n$ lower bound we presented in Chapter 4. Following the obvious mathematical fact that if $\ln n < O(F)$ then $n < e^{O(F)}$ we observe that there is an infinite group of instances in which the amount of clients is smaller than exponential factor of $F$ and where our lower bound is valid.

Before presenting our algorithms, we note that the TPM problem, formulated on a MLMG, has also an obvious approximation algorithm. If we transform a given MLMG instance to an entirely directed graph (replacing each undirected edge by two opposite directions edges) we can consider the TPM problem, represented by that graph, as a regular directed Steiner tree problem (DSP), and use known algorithms in order to solve it. There are several approximation algorithms for creating a directed Steiner tree [12, 49] in polynomial time, however, as we stated in Chapter 3, creating such a tree with a guaranteed approximation ratio is considered to be a very difficult problem. Currently, the best approximation algorithm (due to Charikar et. al. [12]) achieves a $i(i-1)n^{1/i}$ performance ratio in time $O(v^n n^{2i})$ for any fixed $i > 1$ ($n$ is the amount of terminals and $v$ the amount of overall graph nodes). Thus, only an $O(n^8)$ ratio can be achieved in poly-
nomial time, for any \( \varepsilon > 0 \). Practically speaking, in today’s World Wide Web, networks may support a considerable amount of clients, making the approximation ratio achieved by the DSP almost irrelevant. However the number of used media formats is relatively small. Though we do not know of a specific academic study that provides a list of all used media formats, such an attempt has been made by Wikipedia, the free encyclopedia (see http://en.wikipedia.org/wiki/List_of_file_formats). This web site reveals that currently there are only 21 different video file types and 47 audio file types that are in use by different software vendors. It is therefore reasonable to assume that in any operating computer network, \( F << n \) and thus it is preferable to solve the TPM problem using an algorithm with an approximation ratio in the cardinality of \( F \) and not in the cardinality of \( C \) - the group of clients. Such an algorithm is presented in the next section.

5.1 The Bounded Format (BF) Algorithm

In this section we present an approximation algorithm for the general TPM problem without weakening any of its characteristics. We name this algorithm the BF for “Bounded Formats”, indicating its advantage over the guaranteed performance achieved by any known directed Steiner tree algorithms when the amount of formats in the system is small. Like the Steiner tree algorithm, BF creates a spanning tree connecting the media source (the root node’s copy in the top MLMG layer) to all clients (the MLMG’s dummy nodes). However, our algorithm benefits from the special properties of the MLMG which enables it to achieve a better approximation ratio. Namely, the fact that a large portion of the graphs’ edges are undirected and the fact that all layers have the same structure.

Algorithm BF works in three phases (see pseudo-code below). In the first phase the algorithm chooses a single format for each client out of the group of formats the client supports. The data will be delivered to the client eventually using this format. Suppose a specific client supports a group of formats \( Q \subset F \). Our algorithm chooses the format \( q \in Q \) if and only if the path from the root to this format is the shortest one among all formats in \( Q \). After choosing a single format for each client we can erase the dummy node representing the client and all directed edges connecting it to the main MLMG structure. The copy of the client node in the chosen format represents the client in the following phases. We can now consider the problem as if each client supports a single media format.

The algorithm’s second phase runs over all MLMG’s layers which contain nodes rep-
resenting clients. For each such layer it creates an undirected Steiner tree connecting all designated terminals. Since, as mentioned many times before, the undirected Steiner tree is a NP-hard problem, it is up to the implementer of the BF to choose any known algorithm to create (a suboptimal) Steiner tree. A very simple algorithm (see [29]), which creates a spanning tree over a new graph containing only the designated terminals, yields a $2 - \text{approximation}$ performance guarantee. A much more complex algorithm, which is due to Robins et. al. [46] and achieves the best performance ratio known so far, can produce a $1 + \frac{\ln 3}{2} \approx 1.55$ approximation factor. Though in order to achieve the best performance for our BF algorithm it is better to use the latter heuristic, there is of course a performance-time-complexity trade-off. The simple spanning tree algorithm, with the $2 - \text{approximation}$ guarantee, has a running time of $O(m + n \log n)$. Robins et. al. algorithm, which achieves the best known performance, has a very high running time, $O(n f(\theta))$ for some parameter $\theta$, and its optimal performance is achieved for $\theta \to \infty$. Since the difference in performance between the known algorithms is not big, in our algorithmic simulations, presented in chapter 8, we have used a simple heuristic (due to Goemans and Williamson [24]) which is relatively close to the $2 - \text{approximation}$ algorithm. From now on, as suggested in [13], we mark the time complexity for the undirected Steiner tree creation as $\text{Steiner}(n)$ and its performance guarantee by $m$.

\begin{algorithm}
\textbf{Algorithm 1} (BF): \(\frac{m(2F-1)}{K}\) - approximation TPM Algorithm
\begin{algorithmic}
  \STATE $i = 1$; \{The top layer\}
  \STATE $S = \emptyset$; \{A structure for holding the various trees\}
  \STATE $\text{cost}_{BF} = 0$;
  \STATE $\text{Tree} = \emptyset$;
  \STATE \textbf{PHASE 1:}
  \FORALL{$c \in C$ (TPM Clients)}
    \STATE $P \leftarrow$ The Shortest path from $r^1$ to client $c$.\footnote{1}
    \STATE delete the dummy node $c$ and all directed edges connecting to it.
    \STATE $c_i \leftarrow$ The new node to represent client $c$ found at layer $i$.\footnote{2}
  \ENDFOR
  \STATE \textbf{PHASE 2:}
  \WHILE{$i \leq F$}
    \STATE $S_i \leftarrow$ A spanning tree connecting all terminals in $C_i$.\footnote{3}
    \STATE $\text{Tree} \leftarrow$ edges of $S_i$
    \STATE $i+ = 1$;
  \ENDWHILE
  \STATE \textbf{PHASE 3:}
  \STATE $p = \emptyset$ \{A structure holding F-1 paths\}
  \STATE $i = 1$
  \WHILE{$i \leq (F - 1)$}
    \STATE $p_i \leftarrow$ the shortest path connecting $S_i$ to $S_{i+1}$
    \STATE $\text{Tree} \leftarrow p_i$ (Avoid edges added in the previous step)
    \STATE $i+ = 1$;
  \ENDWHILE
  \STATE $\text{Cost}_{BF} = \text{Cost}(\text{Tree})$;
  \STATE Output: $\text{Cost}_{BF}, \text{Tree}$;
\end{algorithmic}
\end{algorithm}
In the BF’s third phase we connect the MLMG’s top layer Steiner tree, formulated at the second phase, to the Steiner trees at all other layers. We do it using the shortest available path. Note that following the BF three phases, it is possible that some MLMG’s edges will be chosen more than once. For example, it is possible that some edge was chosen in the second phase to be part of an undirected Steiner tree at a certain layer, and then it was chosen again to be part of a shortest path connecting two undirected Steiner trees. Our algorithm counts such an edge only once. After finishing all three phases the algorithm outputs the tree we have just created and it’s overall weight, i.e. the solution’s cost. Recall that since we are facing an NP-hard problem, this algorithm generates only an approximated solution.

**Theorem 5.1.1** Algorithm BF is an \( \frac{m(2F-1)}{K} \) - approximation algorithm for the TPM problem (\( m \) is the performance ratio of the chosen undirected Steiner tree heuristic, \( F = |\mathcal{F}| \), and \( K \) is the maximal compression ratio \( K = \frac{f_r}{f_1} \leq 1 \)).

**Proof:** We start the proof by focusing on the second phase of the algorithm and thus we can assume that each client has a single format. We show that the above theorem holds in such a case (i.e. clients support at most one format). We later prove that the theorem holds also for the original TPM instance.

Denote an optimal Steiner tree built at level \( i \) by \( S_i \) and a shortest path connecting the spanning tree \( S_1 \) to \( S_i \) by \( P_i \) (\( P_1 = 0 \)). Since our produced solution consists of \( F \) spanning trees and \((F - 1)\) shortest paths, the overall weight of our algorithm’s output tree (denoted by \( \text{Cost}(BF) \)) is bounded by the following formula.

\[
\text{Cost}(BF) \leq \sum_{i=1}^{F} \text{cost}(P_i) + m \sum_{i=1}^{F} \text{cost}(S_i)
\]

This can be re-written as:

\[
\sum_{i=1}^{F} \text{cost}(S_i) \geq \frac{\text{Cost}(BF) - \sum_{i=1}^{F} \text{cost}(P_i)}{m} \quad (5.2)
\]

\(^1\)The shortest path can be calculated using Dijkstra’s Algorithm [47] for example.  
\(^2\)The last node in \( P \) before the dummy node of client \( c \).  
\(^3\)Consider the original root node (\( r^1 \)) as one of \( C^1 \) terminals.
We mark by $Opt$ the minimal cost feasible solution to a given TPM problem which is formulated on a MLMG and prove the following two equations:

$$\forall i \ 2 \leq i \leq F \quad Opt \geq \text{cost}(P_i) \tag{5.3}$$

$$\forall i \ 1 \leq i \leq F \quad Opt \geq K \cdot \text{cost}(S_i) \tag{5.4}$$

**Proving equation 5.3:** We consider all clients found at layer $i$ in the MLMG. Note that any TPM solution must connect the root ($r^1$) to all clients via some spanning tree. Let us mark by $Y^c_i$ the length (cost) of a brunch in $Opt$ connecting the root to a client $c$ found in layer $i$. It is obvious that $\forall c, i \ Opt \geq Y^c_i \geq P^c_i$ where $P^c_i$ is the length of the shortest path from the root to that client. Note that the root is a part of any Steiner tree connecting all clients of layer 1, thus, $P_i$, which is the shortest path connecting the top Steiner tree to the steiner tree at layer $i$, is not longer than any shortest path connecting the root to any layer $i$ client, which are part of the Steiner tree of layer $i$. In other words: $\forall c, i \ P^c_i \geq P_i \square$.

**Proving equation 5.4:** This equation is proven using the MLMG’s second property: The identical topology of all the layers. Given any TPM solution, formulated on a MLMG, and in particular the optimal solution, let us erase all the solution’s vertical edges connecting different layers. Then, we compress all layers down to the bottom layer. This latter action can be viewed as copying the chosen horizontal undirected edges, found at layers $1 - (F - 1)$, to the bottom layer $F$. While copying these edges we change their cost according to the ratio between the compression factor of their original layer to the compression factor of the bottom layer ($f_F$). The compression action is best presented in Figure 5.1. Note that the result of this action is the creation of one undirected graph at the bottom layer $F$. This graph represents all horizontal moves chosen by the given TPM solution. For simplicity we use the term compression for this action from now on. It is easy to prove that the new undirected graph, created at the bottom layer, is connected. If it is not connected it means that the original solution was also not connected which contradicts its feasibility.

We mark by $K$ the maximal compression ratio in our system which is described by the following formula: $K = \frac{f_F}{f_1} \leq 1$. If indeed $f_1 = 1$ then $K = f_F$, however, $f_1$ may be different from 1. One can describe a system in which the source node itself doesn’t hold the raw media object but rather a more compressed format. In this case $f_1 < 1$ and $K = \frac{f_F}{f_1} > f_F$. 


Observe that for any horizontal edge, $e_i$, chosen by any TPM solution at layer $i$ the following inequality holds: $\text{Cost}(e_f) \geq K \cdot \text{Cost}(e_i)$ where $e_f$ marks the copy of edge $e_i$ found at layer $F$. Note that edge $e_f$ must exist since all MLMG layers have the same structure. Recall that $Opt$ is the cost of an optimal solution, $Opt_F$ is the compressed optimal solution towards layer $F$ ($Opt_F$ contains only copies of horizontal edges from $Opt$) and $S_F$ the minimal Steiner tree connecting all terminals in $Opt_F$. $Opt_F$ is an undirected graph connecting all copies of clients. Note that any Steiner tree, connecting only a subset of the clients, created at $Opt_F$ must cost less than $S_F$, which is the minimal tree which spans all clients. Using the above observations we have the following inequality which proves Equation 5.4:

$$\forall i \ 1 \leq i \leq F \quad Opt \geq Opt_F \geq S_F \geq K \cdot S_i$$

It is left to show that our performance analysis holds even when the first phase of the algorithm is taken into consideration, i.e. when each client can accept more than a single media format. First note that the analysis of Equation 5.4 is valid for any subset $S_i$ of the clients. This is true since regardless of the format used by the optimal algorithm for a client $c$ (and it has to chose one format), when compressing all the layers towards the bottom we still receive a connected graph which spans all clients. The overall weigh of this graph is not larger than the weigh of the optimal TPM solution but it is bigger than a compressed Steiner tree connecting only subgroup of clients.

The proof of equation 5.3 under the consideration of $BF's$ algorithm first phase, is
based on the fact that we have chosen for a client \( c \) a format that minimizes the distance from the root to the dummy node representing that client. Consider a layer \( i \) and a client \( c \) that Algorithm \( BF \) puts in layer \( i \). Since the optimal solution must also connect client \( c \) to the root, \( Opt \) must be greater or equal to the distance from the root to node \( c \), and thus \( Opt \geq cost(P_i) \).

To complete the proof of Theorem 5.1.1 we need some basic algebra. Using Equation 5.3 (sum over all the \( i \)'s):

\[
(F - 1)Opt \geq \sum_{2}^{F} cost(P_i)
\]

From Equation 5.4 and Equation 5.1(again sum all the \( i \)'s) we get:

\[
F \cdot Opt \geq K \cdot \sum_{i=1}^{F} cost(S_i) \geq K \cdot \frac{Cost(BF) - \sum_{2}^{F} cost(P_i)}{m}
\]

As discussed in Chapter 4, the undirected Steiner tree problem is MAX-SNP, thus always \( m > 1 \). Moreover, since \( K = \frac{f_r}{f_1} \leq 1 \) we get that \( \frac{m}{K} > 1 \) and thus we get:

\[
\frac{m(F - 1)}{K} Opt \geq \sum_{2}^{F} cost(P_i)
\]

\[
\frac{mF}{K} \cdot Opt + \sum_{2}^{F} cost(P_i) \geq Cost(BF)
\]

\[
\frac{mF}{K} \cdot Opt + \sum_{2}^{F} cost(P_i) \leq \frac{mF}{K} \cdot Opt + \frac{m(F - 1)}{K} Opt
\]

\[
\frac{m(2F - 1)}{K} Opt \geq Cost(BF) \quad \square
\]

Let us now analyze the time complexity of Algorithm \( BF \). The algorithm’s three phases run serially, thus, the algorithms overall running time is simply an order of the maximal running time among all three phases. \( BF \) first phase calculates shortest path for each client in the graph, thus, its time complexity is no more than a \( O((Fn)^2) \) \(^4\). The second phase creates a Steiner tree on each layer. Following our previous analysis we mark the Steiner tree time complexity as \( Steiner(n) \) and we get that the second phase takes \( O(F \cdot Steiner(n)) \). In the last phase we calculate \( F - 1 \) shortest paths connecting the top layer’s tree to the trees beneath. This phase can be calculated using a simple

\(^4BF \) runs on the MLMG which has \( nF \) nodes
“All-pairs shortest path” algorithm like the Floyd-Warshall [47] or by running \(n\) times Dijkstra’s Algorithm which takes no more than \(O(F^2n^3)\). Note that for the algorithm’s analysis we could settle with the paths calculated at the first phase connecting the root to each layer with no need to calculate new paths at phase 3. Summing all three phases we get a simple polynomial time complexity \(\max(O(F^2n^3), O(F \cdot \text{Steiner}(n)))\).

Note that in Chapter 4 we showed the existence of a \(\log F\) lower bound for our TPM problem. In this chapter we have presented an \(O(F)\) approximation algorithm for the problem. Can the \(O(\log F)\) \(O(F)\) gap be closed?

Remember that in chapter 4 we have also presented the strong connection between the TPM and the Directed Steiner tree problem (DSP). If we assume that an \(O(\log(F))\) approximation algorithm does exists for the TPM, surly, such an algorithm can be used for a DSP instance with \(F\) nodes and \(n = O(F)\) terminals. Unfortunately, as we mentioned before, such an algorithm doesn’t exist. The best algorithm can achieve an \(O(n^\varepsilon)\) performance for every \(\varepsilon > 0\) in polynomial time, but, a \(\text{polylog}(n)\) performance can be obtained only in a quasi polynomial running time. This observation is not an evident that the mentioned gap can never be closed, however, it does indicate that achieving such a result, might be hard.

**DEALING WITH THE COMPRESSION RATIO \(K\).** Note that theoretically speaking, though we claimed to have an \(O(F)\) approximation algorithm for the general TPM case, the algorithm’s performance is also a function of the system’s compression ratio \(K\). This compression ratio is calculated from the set of media formats which are input variables for the problem, thus, \(K\) can sometimes be big, making the algorithm’s approximation performance very large. However, practical tests that were done to compare different media formats revealed that the compression ratio has some threshold under which the media cannot be properly viewed and thus systems shouldn’t be design to support such formats. For example, the following web site: [www.dom9.org](http://www.dom9.org), which mainly deals with dvd technologies, has published an extensive test comparing the performance of seven available media formats which are examined on 3 different movies (see the full article in [http://www.doom9.org/index.html?/codecs-103-1.htm](http://www.doom9.org/index.html?/codecs-103-1.htm)). The largest compression ratio in their test was about 23\% \((K = \frac{L_F}{L_1} = 0.77)\). The test reveals that even a 21\% compressed format can sometimes return clips with serious quality degradation. Other tests done, such as the one published in

\[5\text{We only need shortest path from the top layer that has } n \text{ nodes, thus we get } n \cdot (F n)^2\]
“http://people.csail.mit.edu/tbuehler/video/codecs/index.html”, claim different practical results, pointing to the difficulty in establishing a unified standard for claiming what is a good enough image quality. The above mentioned web sites deal with media encoding technologies, thus, the discussions in these web sites focus on comparing different media codecs (MPEG vs. AVI for example). It is also possible to focus on adjusting the same media format to suit different screen sizes and screen resolutions. In these cases we can encounter very large compression ratios that do not necessarily cause quality problems in the eyes of the viewers. However, in these cases it is important to ask whether the media compression is reversible, i.e. is it possible to restore the media file to its original size and quality. This question is imperative to this discussion, however, its answer lies in the field of information theorem in general and data entropy in particular which are out of the scope of this thesis.

For a brief explanation of the answer, let us mention only two basic definitions from information theorem:

A “Lossless Compression” is a compression of data in which all the information is preserved, thus, a compatible decompression can restore the information back to its original size and content. The well known zip compression, for example, belongs to this group.

A “Lossy Compression” is a compression of data in which some of the information is lost, making it impossible to restore the data back to its original state. Most encoding techniques comprise in media files (such as MPEG) belong to this group.

In order to avoid formal proofs we simply state that when factor $K$ is too large, the compression it describes becomes lossy, making it infeasible to guarantee a possible data decompression that restores the media to its original format. We can model this observation into our MLMG graph in the following way: If there is some large compression ability in our system, represented by a directed path pointing down between two MLMG layers with a large compression gap among them, practically speaking, there couldn’t be a directed path going back to the top layer since the compression is lossy and the original format cannot be restored. In the next section we prove that when the MLMG represents a compression ability only system (all vertical arcs in the graph are pointing down) factor $K$ can be dropped from the approximation ratio allowing us to preserves our $O(F)$ analysis of the algorithm. In a MLMG graph, which represents a practical scenario, there could be directed edges pointing up and not just down, however, following the analysis of Sec-
tion 5.2 we can bound the compression ratio relevant to the approximation performance to a factor which is less than the maximum compression in the system.

In Chapter 7 we present a different angle to deal with the compression ratio $K$. This angle takes a more theoretical approach. We also discuss this issue in the conclusions and open problems’ chapter of this thesis.

## 5.2 Compression only system

In this section we deal with a group of subcases of our general TPM problem. All instances in this group have the following two characteristics: A) Each client in these instances supports a single format. B) They describe a compression only system - a network in which all possible transcodings are compressions only. In the MLMG model it is easy to describe such a system. Remember that in our construction the MLMG layers should be ordered according to their compression ratio from the least compressed format (the media original encoding, for example) to the most compressed one, thus, a compression only system is represented by a MLMG in which all vertical directed arcs are pointing down. In the previous chapter (Theorem 4.0.2) we already proved that the compression only TPM instances have a $\ln F$ lower bound, here we show how to use the special characteristics of these instances to improve the approximation ratio of the general case.

The importance of the compression only TPM does not lie in the algorithm we offer to solve it but in its approximation analysis. In fact, algorithm $BF$ without its first step $^6$ is enough for our purpose. Remember that in this algorithm we first formulate undirected Steiner trees at each layer and then we connect all Steiner trees by shortest paths going from the top Steiner tree to the Steiner trees at the layers beneath. We will now prove the following theorem:

**Theorem 5.2.1** Algorithm $BF$ produces an $m \cdot (2F - 1)$ approximation ratio for the compression only TPM problem with single format per client.

Note that the difference between the approximation ratio we prove for this case and the one proved before for the general case is the lack in the compression ratio $K$. This

---

$^6$The first step is not needed since each client already supports a single format.
difference is the result of changing equation 5.4 of the former proof with the following equation.

\[ \forall i \in \{1, \ldots, F\} \quad Opt \geq \text{cost}(S_i) \quad (5.5) \]

We need to prove that Equation 5.5 is true in our case. Recall that \( Opt \) is the cost of an optimal solution and \( S_i \) is an optimal undirected Steiner tree connecting all clients of layer \( i \). First note that in order to connect all designated clients of Layer 1 (the top layer) to the root node, any solution for the compressed TPM version must create a tree at layer 1 connecting these exact clients. This is true since we assumed that all MLMG vertical edges are pointing downwards and that each client wishes to receive exactly one specific format. Since \( S_1 \) is the cheapest possible tree connecting all terminals of layer 1 our above equation holds for that layer.

Unlike the top layer, it is not true that any solution must create trees at layers 2 to \( F \). It might be, for example, that an optimal solution will choose to traverse over two different directed edges going down from layer 1 to layer 2 thus creating a forest at layer 2 and not a tree. In order to prove Equation 5.5 for layers 2 to \( F \) let us look at a TPM instance where all vertical edges pointing down. We focus on a certain layer \( i > 1 \). As we did for the general case, let’s compress all the above layers in the direction of layer \( i \in F \) in order to get one undirected graph representing all horizontal moves in the layers up to layer \( i \) (including it). At a compression only system, as opposed to the general case, it is easy to see that it is not needed to compress the entire MLMG in order to guarantee the creation an horizontal undirected graph. In this system, when compressing the layers of the given solution down towards layer \( i \) we are constantly maintaining a connected graph form. This can easily be proven by induction on the layers \( j : 1 \leq j \leq i \). We leave this simple induction to the readers.

For the rest of this proof we claim that in the compressed graph, described above, the cost of any TPM solution is smaller than the cost of the same solution on the original graph since this compressed graph does not include the vertical arcs. Furthermore, it presents a one layer picture of the solution in which all terminals of layer \( i \) are connected over an undirected graph. Since the shortest way to connect a group of terminals to a root is via an optimal Steiner tree this compressed graph is at least as expensive as any optimal Steiner tree \( S_i \). That proves Equation 5.5.

We can now prove Theorem 5.2.1 using equations 5.1, 5.2 and 5.3 of the previous
section which are still valid in this case. □.

Let us now consider an even more degenerate subcase of the TPM. In this case we assume also an homogenous conversion cost, i.e all possible transcodings in the network cost the same. We mark this cost by $X$. Moreover, we also assume that the root, the network node which holds the media object and wishes to multicast it, can convert the media to all formats in $\mathcal{F}$. We name this TPM subcase the homogenous compression TPM problem.

The approximation algorithm we offer for this case resembles the BF algorithm but it also benefits from the assumption about the roots conversion capabilities. The algorithm runs on the problem’s compatible MLMG. It iteratively go over all $\mathcal{F}$ layers and at each layer $i \in \mathcal{F}$ connects the local copy of the root node $r^i$ to the group of terminals at that layer using any undirected Steiner tree algorithm. The connection of the upper layers to the $i^{th}$ Steiner tree is done using an arc going down from an upper copy of the root node to the $i^{th}$ copy of that node. According to our assumption regarding the root’s converting ability, such an arc must exist. If the algorithm does not find an arc going from $r^{i-1}$ to $r^i$ than there must be some other arc going from $r^i$ to $r^j$ in some upper layer ($j < (i - 1)$). It is simple to see that our above algorithm runs in time $O(F \cdot \text{Steiner}(n))$.

**Theorem 5.2.2** The approximation algorithm for the homogenous compression TPM problem produces an $mF$ – approximation ratio, where $m$ is an approximation ratio for the undirected Steiner tree problem and $F = |\mathcal{F}|$.

**Proof:** Following the proof of Theorem 5.2.2, it is easy to show that the approximation ratio we claimed to this TPM instance is correct. First note the slight differences between the equations describing the cost of the tree outputted by the algorithms for the former case and for this case. Here, the tree’s cost ($Cost_T$) is composed of $F$ times an undirected Steiner tree, created at each layer, and the cost of the arcs connecting these layers through copies of the root node ($r^i$), such connections don’t require any additional horizontal edges. The algorithm needs only $F - 1$ arcs to connect all layers and all arcs cost the same ($X$).

$$Cost_T = (F - 1)X + m \sum_{i=1}^{F} \text{cost}(S_i)$$  \hspace{1cm} (5.6)

In our proof for Theorem 5.2.2 we have already discussed the reason according to which the compression factor $K$ was dropped from the algebraic equations and eventually
from the performance factor. The question that remains open for this TPM case is why did another factor $F$ was dropped from the approximation ratio? The answer to this question is that since we are dealing with a subcase in which all vertical arcs in the MLMG have an equal cost, we need not separate between equation 5.3 and 5.5 but to combine them. In other words, we can claim that for all layers $i \in \{2..F\}$ $Opt \geq cost(S_i) + X$ where $S_i$ describes the undirected Steiner tree formulated at layer $i$ and $X$ the cost of connecting layer $i$ to some layer above. This equation is true since there is no way, even for an optimal solution, to reach a layer $i > 1$ without traversing at least one vertical arc, and all vertical arcs cost $X$. For the top layer an additional cost $X$ is not needed, thus, for layer 1 we claim $Opt \geq cost(S_1)$. Using these equations, combined with the equation describing the algorithm’s output, the algebraic result is exactly the above performance ratio $\square$.

In order to show that for this TPM problem case our approximation analysis is tight we present an infinite family of instances on which our algorithm indeed produces an $mF – approximation$. Consider a graph $G^{TPM}$ in which all clients are connected in a star shape to a group of nodes $L$ as depicted in Figure 5.2. The communication cost between nodes in $L$ and the client are 0. Only the root node and the nodes in $L$ have transcoding capability and with costs 0. The compression ratio to all media formats from the original format is very small $\varepsilon \approx 1$ (there is hardly any compression). In Figure 5.2 the clients are marked with colored nodes (nodes with the same colors can receive the same media format) and the root node is the one painted black. Only 3 nodes in the graph can perform transcoding operations: The root and the two nodes (the $L$ nodes) marked with an inner circle. Note that such a graph maintains all properties of the homogenous compression TPM case.

It is obvious that an optimal solution will send the media in its original format through an optimal Steiner tree, and perform all required transcoding at the $L$ nodes. An optimal solution, therefor, costs no more than one undirected Steiner tree - $Opt = S_1$. Our algorithm, contrarily, will chose to transform the original media format to all required formats at the root and will have to pay $F$ times $m \cdot S_1$ to each media format. Since the clients are connected with a star shape to the same nodes, the Steiner trees chosen to each format will be the same. Furthermore, since the compression ratios are tiny all Steiner trees will ultimately cost the same, about $S_1$. Our algorithm will thus pay $mF \cdot S_1$ leaving the approximation ratio $mF$.  

36
Figure 5.2: A tight example for the homogenous compression TPM

5.3 Symmetric conversion system

So far we have dealt with TPM problem cases in which transcoders had asymmetric conversion ability. The fact that a given transcoder could convert format $f_i$ to format $f_j$ does not indicate that the same transcoder has the ability to convert $f_j$ to $f_i$ at the same cost. The asymmetric transcoding capabilities were represented using directed edges. In this section, we focus on the case of symmetric conversion ability. If a network entity can transcode format $f_i$ to $f_j$ it can also transcode the formats in the opposite direction and at the same price. Here transcoding capabilities are represented using only undirected conversion graphs. We name this TPM version the SYMMETRIC TPM problem or the STPM problem for short.

Using our former presented results on the general TPM problem one can immediately conclude the following facts:

A) The $\ln F, \ln n$ lower bounds of the general TPM remains even for the symmetric case. Following our SET COVER and HITTING SET reductions, it is obvious that even if every $f_i$ format could be converted back to format $f_0$ our reduction still holds.

B) One can still use our BF algorithms for the general case in order to receive an $O(F)$ approximation ratio for the STPM problem. An $O(n^F)$ approximation can also be gained by using a directed Steiner tree approximation algorithm and addressing the entire MLMG as a directed graph.

Let us start the discussion on the symmetric TPM (STPM) by showing that it is a special case of the group Steiner Tree problem.
Definition 5.3.1 Group Steiner tree problem: Given an undirected weighted graph $G = (V, E)$ and a collection $Q$ of subsets (groups) of nodes $q_i \subset V \ (i \in (1..K))$, connect all groups of vertices using the most lightweight tree.

Note the following two observations: One, w.l.o.g we can assume that the groups are disjointed. If there is a node belonging to $x$ groups connect it to $x$ dummy nodes using zero cost edges and replace the original node in each of the $x$ groups by a different dummy node. Two, it is suffices that the group Steiner tree will touch no more than one node in each group. The two observations reveals the connection between the STPM and the group Steiner tree. In the STPM, formulated on the MLMG, each client is represented by a separate group containing nodes from all the formats the client supports. The STPM objective is to create some spanning tree connecting at least one vertex from each group. Note, that the very restricted STPM case, where each client supports only a single format, is easily solvable using a regular Steiner tree heuristic.

The group Steiner tree problem was first discussed by Reich and Widmayer in 1989 [45] which gave it the first approximation algorithm with a guaranteed performance of $O(K)$ ($K$ is the amount of groups). A major breakthrough in approximating the problem came only in 1998 following the work of Bartal’s on probabilistic approximation of metric spaces [7]. Since we are using Bartal’s result in a more direct manner in Chapter 7, we chose not to elaborate on it here but only to explain its general concept in the following paragraph.

Many optimization problems are defined in terms of a metric space. Several problems are known to be difficult to solve on general graphs but are considered easy on tree graphs. Good examples are the DOMINATING SET (DS) problem [32] and the INDEPENDENT SET (IS) [32] problem, both can be solved optimally on trees but considered very difficult, in terms of approximation ratio, on general graphs [9]. The idea behind Bartal’s work is the possibility that metric spaces can be well approximated by other metric spaces. In other words, in order to solve NP-hard problems on a “difficult” metric space $M$ we should find a “nice” metric space $N$ that can “approximate” $M$ relatively well. Then, an algorithm for a problem $P$ on metric $N$ induces an algorithm for $P$ on metric $M$ with only a small penalty in performance. Bartal, in his 1998 work [7], has succeeded in finding such metric spaces that can probabilistically approximate general graphs by a small factor. He named these “nice” metric spaces HSTs, short for “Hierarchically Well-Separated Trees”. A more in-depth explanation about the general concept and in
particular Bartal’s pioneering work can be found in Chapter 7.

The idea of using simple metric spaces has driven researchers to focus on producing algorithms for difficult problems on trees. Using Bartal’s technique and its result on HSTs, they automatically gained an algorithm for general graphs with an approximation ratio of $O(\log(n))$ times the approximation they achieved for the tree metrics ($n$ is the number of graph nodes). The first to implement this technique for the group Steiner tree problem were Garg, Konjevod and Ravi [23]. The three used a linear programming and randomized rounding to establish a $\log N \cdot \log K$ approximation algorithm for the problem on trees where $N$ is the cardinality of largest group and $K$ is the number of groups. Their result, combined with Bartal’s work, automatically produced an $O(\log N \cdot \log K \cdot \log n)$ approximation factor for the problem on general undirected graphs. The same approximation result was later achieved by Charikar et. al [10]. Charikar succeeded to derandomize the tree metric establishment and achieve the first deterministic approximation algorithm with a logarithmic approximation factor.

Using any of the above mentioned algorithms, which have produced a logarithmic factor approximation, we can immediately gain the following performance ratio for our STPM problem: $O(\log N \log K \log n) \implies O(\log F \log^2 n)$. This ratio is achieved following these three observations. First, the maximum size of any group in our graph construction is $N = |F|$. Second, the amount of groups in the STPM problem is as the amount of clients ($K = |C| = O(n)$) and third, though our construction of the MLMG increases the amount of nodes by a factor of $F$, $\log Fn = \log F + \log n$ and our above analysis holds.

**How much can we improve the $O(\log F \log^2 n)$ approximation ratio for the STPM problem?** Note, that this result was achieved without taking into consideration the MLMG characteristics, mainly, the fact that all layers have the same topology. However, it is easy to show that the approximation factor we achieved cannot be improved much. Consider an instance $I_{GS}$ of a general group Steiner tree over an undirected graph with $n$ nodes. We will build an instance - $I_{STPM}$ for the STPM problem where $I_{GS}$ is its single transcoding graph. As we did for analyzing the TPM lower bounds, consider a simple network with a star topology with at least $K + 1$ nodes, where $K$ is the amount of groups in $I_{GS}$. The star’s central node will represent the media origin and $K$ of its surrounding vertexes will be the media clients. Let the central node, the root, be the only node with transcoding capability where $I_{GS}$ is its transcoding graph. Note that following this reduction $I_{GS}$ nodes represent different media formats in $I_{STPM}$. The clients in this instance are able to receive formats from the group they represent in $I_{GS}$. Following our

39
construction it is clear that an optimal solution over $I_{GS}$ induces an optimal solution with the same cost on $I_{STPM}$. As we claimed above, the best known algorithm for solving the group Steiner tree problem achieves an $O(\log^3 n)$ approximation ratio. Applying this algorithm on our instance we can guarantee the same approximation result with only a slight change. Since $I_{GS}$ nodes represents STPM’s formats ($F = n$), we can conclude that the best algorithm can achieve an $O(\log^3 F)$ approximation ratio on $I_{STPM}$. More formally, if we could develop a better than $O(\log^3 F)$ approximation algorithm for the STPM problem in general we could apply it to the $I_{STPM}$ instance we just constructed thus improving the best known $O(\log^3 n)$ approximation ratio for the group Steiner Tree problem. We have not showed here a clear lower bound for the STPM problem but rather an hardness of achieving a better than $O(\log^3 F)$ approximation factor.

In this chapter we presented algorithms dealing with the general TPM problem and some subcases. The main contribution of these algorithms is their ability to guarantee an approximation ratio of $F$, the number of formats in the system, and not $n$ the number of clients. Note that despite these algorithms the gap between the $O(F)$ approximation guarantee and the $\ln F$ lower bound still exists, thus, a large ground is left for future research. In the following chapters we continue to study more interesting instances on the TPM problem.
Chapter 6

Approximation Algorithms For Extended TPM Problems

In this chapter we focus on several practical extensions to the TPM problem. The basic TPM problem, as defined in Chapter 3, is only a special subcase of these extensions. For example, we examine here a scenario in which the source node holds the media object in several formats in advance before sending it on the net. We also examine a different scenario which considers the bandwidth limitations of the network links and thus prevents using the same link unbounded amount of times. Note that since we are dealing in this chapter with extensions to the TPM problem, the problem’s lower bounds as described in Chapter 4 also hold for these extensions.

6.1 Multiple origin formats scenario

Consider a case in which the TPM root, wishing to distribute the media stream, holds that media stream in more than a single format. That is, the sender has apriori access to several formats of the media without the need to perform transcoding. This case is rather practical considering most web servers, with media downloading capability, hold the media objects in several formats corresponding to different client’s capabilities. Suppose the source node has the original media stream encoded in all formats of the set $Q \subseteq \mathcal{F}$. It is simple to represent such a case in our MLMG model simply by choosing one copy of the source node from the set $Q$ layers and connect it to the rest of $Q$ layers using zero weight edges. The chosen copy is the equivalent to the root node in the original TPM problem defined in Chapter 3. Since moving between the different root’s copies
in $Q$ layers cost nothing, this presentation is equivalent to the multiple origin format scenario. Using this presentation we can now run our original BF algorithm to solve this TPM version. Of course our initial analysis of the performance ratio produced by the algorithm is still valid, leaving us with the same approximation ratio $m \cdot (2F - 1)$.

### 6.2 Capacitated TPM (CTPM)

The definition of the TPM problem in Chapter 3 did not take into consideration the finite capacity of network links and the finite transcoding ability of real network elements. In this section we present extended models that capture the above mentioned properties. It turns out that once these properties are added to the TPM problem it becomes inapproximable, that is, neither the link capacitated version nor the transcoding capacitated version can be approximated at all. In order to prove these inapproximation results we use reductions from NP-complete problems to unique versions of our capacitated TPM (CTPM) problems. In such a unique version it is easy to distinguish between two type of CTPM instances. The first type are CTPM instances $I$ with $Opt(I) \leq X$ which are mapped to YES instances of the NP-complete problem and the second type are instances $I$ with $Opt(I) \geq \rho \cdot X$ which are mapped to NO instances of this NP-complete problem. Clearly finding such a reduction would imply that if we had a $\rho - approximatio$ algorithm to the CTPM optimization problem we could have used it to decide the NP-complete problem in polynomial time, thus, such an algorithm could not exist. This technique, often used to prove hardness of approximation results, is called a gap-preserving reduction and we use it to prove the inapproximability results for the two CTPM problems below.

#### 6.2.1 Link Capacitated TPM (LCTPM)

A possible solution to a given TPM instance may traverse the same network link more than once. This can easily be seen when presenting the solution on a compatible MLMG. For example, consider a media stream ready to be sent from a source to two different clients over a 'Y' shaped network. The two clients, located at one side of the network, demand the media in two different formats, and the source node, at the other side, has the only ability in the network to transcode this media into the appropriate formats. In order

\[1\text{Note that in this TPM version we need only } F - |Q| \text{ vertical paths to connect MLMG layers not in } Q, \text{ thus, the approximation ratio can actually be improved to } m \cdot \left(2F - |Q|\right) \]

---

42
to deliver the stream the source must locally convert it into the two formats and send it on the same link to the other side of the network. For the media to properly reach the clients, the links from source to the internal cross point must have enough bandwidth to be able to simultaneously traverse the two formats. Practically speaking, it is possible that network links have some capacity which limits the amount of bits they can transfer at any given moment. Such a constraint should prevent us from creating a TPM solutions which uses the same link more than a bounded amount of times simultaneously. The question of how to create a TPM solution without violating such a constraint is the LCTPM problem. Following is a more formal definition of this problem.

**The LCTPM Problem Definition:** we are given an undirected capacitated weighted graph $G^{LCTPM} = (V, E)$, with a nonnegative capacity $C_e$ associated with each edge $e \in E$. The $G^{LCTPM}$ graph represents the multicast supporting network and replaces the $G^{TPM}$ graph of the regular TPM problem. We are also given the same inputs as the regular uncapacitated TPM version: A set of nodes representing the clients, a single node representing the media source, a group of weighted directed graphs representing format conversion ability of network nodes and eventually a subset of formats associated with each client to represents the media formats the client can accept.

Similar to the regular TPM problem, the LCTPM problem objective described over a compatible MLMG is to to create a spanning tree which connects the root node (the media source) to the dummy nodes which represent the clients. The resulted spanning tree must withstand the capacity constraints of the network links, meaning, the sum of compression ratios of the media formats which traverse each link must not exceed the capacity $C_e$ of that link. More formally, if $X_{(e,i)}$ is an indicator variable representing the traversing of media format $i \in F$ on network link $e \in E$ ($X_{(e,i)} = 1$ if format $i$ traverses on link $e$ and 0 otherwise), and $f_i$ is the compression ratio of format $i \in F$ then the following formula represents LCTPM problem capacity constraints: $\forall e \in E \sum_{i=1}^{F} f_i \cdot X_{(e,i)} \leq C_e$.

Using the technique of gap preserving reductions, as it was explained above, we can prove the following theorem:

**Theorem 6.2.1** The LCTPM problem cannot be approximated within any given performance ratio $\rho$ unless $P = NP$.

The theorem is proven using reduction from the Steiner Forest with Bandwidth Constraints (SFBC) problem to a unique version of the LCTPM problem.
**Definition 6.2.1** Steiner Forest with Bandwidth Constraints (SFBC): Given a graph \( G = (V,E) \) a set \( D \subset V \) of terminal where \(|D| = K\), we want to find \( K \) different Steiner trees to connect the same set of terminals \( D \) such that any two Steiner trees do not share any common edge.

It was shown in [38] that the SFBC problem is NP-complete. In order to prove Theorem 6.2.1 we assume that there is a polynomial time approxim ation algorithm for the LCTPM problem that guarantees a performance ratio \( \rho \geq 1 \). Given an instance \( I_{SFBC} \) of the SFBC problem with an undirected graph \( G = (V,E) \) and a set of terminals \( D \), as depicted in Figure 6.1A, we construct the following LCTPM instance - \( I_{LCTPM} \): We extend graph \( G \) of \( I_{SFBC} \) to a complete graph \( G' \) representing \( I_{LCTPM} \) network. This is done by adding edges with weight \( \rho \cdot \binom{n}{2} \), where \( n \) is the number of vertices in \( V \), and assigning weight 1 to the original edges of \( E \). Moreover, we chose one of the vertices in \( D \) to represent \( I_{LCTPM} \) root (the root is marked by \( r \) in Figure 6.1B). To each of the remaining \(|D| - 1\) terminals we connect \(|D|\) vertices in a star shape manner using a zero weight edges. These new \(|D|(|D| - 1)\) terminals represent \( I_{LCTPM} \) clients (these are the non solid colored nodes in Figure 6.1B). \( I_{LCTPM} \) has \(|\mathcal{F}| = |D|\) formats and \( \forall i \in \mathcal{F} \ f_i = 1 \). Moreover in \( I_{LCTPM} \) \( \forall e \in E \ C_e = 1 \), in other words, each link in the constructed network has only enough bandwidth to carry a single media format at any given moment. The root \( r \) holds the media object in format \( f_1 \) and at each star, containing \(|D|\) vertices, each node requires a different format. Only the root node \( r \) has transcoding capabilities and can convert \( f_1 \) to any other format at zero cost. This finishes the construction of instance \( I_{LCTPM} \) which can be depicts fully in Figure 6.1.

Following our reduction, note that if there is a solution for the \( I_{SFBC} \) which constructs \(|D|\) different spanning trees to connect the \(|D|\) terminals then our constructed \( I_{LCTPM} \) has a solution with a cost at most \( \binom{n}{2} \) (the maximum amount of edges in \( E \)). In such a solution, the root \( r \) converts format \( f_1 \) to every other format in \( \mathcal{F} \) and sends the media object over the \(|D|\) different spanning trees, exactly those chosen for \( I_{SFBC} \). Note that in this solution each network link is not used more than once and the only links that are in use are those from the original graph \( G \) with a cost of 1. If no solution for \( I_{SFBC} \) can be constructed on graph \( G \) than in order to create any feasible solution for \( I_{LCTPM} \) we need to use at least one new edge (the thinner edges in Figure 6.1B), making the overall cost of the solution at least \( \rho \cdot \binom{n}{2} \). Following our explanation on the gap preserving reduction technique, if we had a \( \rho \)-approximation algorithm for the LCTPM problem, we could use it as a polynomial time algorithm to solve the SFBC problem. This contradicts the NP
completeness of SFBC, and thus, the existence of any $\rho$-approximation algorithm for the LCTPM problem.

6.2.2 Transcoding Capacitated TPM (TCTPM)

In this section we bound the computational ability of the transcoders placed at the network nodes. We assume that network elements cannot convert more than a bounded amount of media objects at any given moment since it may exhaust their CPU cycles and prevent them from performing other crucial functions. A good example is a computationally weak computer which is the only network node with transcoding capabilities. It is practically inconceivable to create a truly multicast session using only the conversion abilities of this node in order to simultaneously supply the media stream in many different formats. In our MLMG model, such a constraint is described by bounding the overall cost of edges of the same converting graph that can be part of the created spanning tree solution. Following is a more formal definition of the TCTPM problem.

The TCTPM Problem Definition: We are given the same input as the regular TPM problem to represent the multicast supporting network: An undirected weighted graph to represent the network structure, a set of nodes in the graph to represent the clients and a single graph node to represent the media source. In the TCTPM problem, each weighted directed graph $i$, which represent a network node transcoding capability, is defined with

Figure 6.1: LCTPM - reduction via the SFBC problem

A) SFBC Instance  B) LCTPM Instance
an additional nonnegative numerical value $U_i$. This value, is the constraint on the overall allowed usage of the conversion graph edges, meaning, the total cost of using conversion graph $i$ in a TCTPM solution should not exceed $U_i$. More formally, if $X_{(e,i)}$ is an indicator variable representing the usage of edge $e$ of conversion graph $i$ in a TCTPM solution, and $C_e$ is the cost (weight) associated with this edge, than the following formula represents TCTPM problem usage constraint: $\forall$ conversion_graph $i \sum C_e \cdot X_{(e,i)} \leq U_i$. Note that when all edges in a converting graph have the same cost and this cost is also equal to the $U$ value associated with the graph, than the media stream sent on the network can use this converting graph only once in a TCTPM solution.

Similar to the regular TPM problem, the TCTPM problem objective, described on a compatible MLMG, is to create a spanning tree connecting the source node to the dummy nodes which represent the clients. This TCTPM spanning tree should not violate the usage constraint of any conversion graph in it.

We would like to prove the following theorem:

**Theorem 6.2.2** *The TCTPM problem cannot be approximated within any given performance ratio $\rho$ unless $P = NP$.*

We prove Theorem 6.2.2 using a gap preserving reduction from a version of the Hamiltonian path problem. The objective of the Hamiltonian path problem is to find a simple path in a graph that passes through every graph node exactly once. In the following reduction we use a version of this problem in which the path must starts from a specific node. It is well known that the Hamiltonian path problem is NP-complete [47] thus our used version of the Hamiltonian path is NP-complete also. Suppose now that there is a polynomial time approximation algorithm for the TCTPM problem that guarantees a performance ratio $\rho \geq 1$. Given an instance of our desired version of the Hamiltonian path problem $I_{ham_path}$, which consists of an undirected graph $G = (V, E)$ and a node (lets say node 1) which starts the path, we construct the following TCTPM instance - $I_{TCTPM}$. $I_{TCTPM}$ network is a simple line with $n + 1$ nodes, where $n$ equals the amount of graph nodes in $I_{ham_path}$ ($n = |V|$). All edges in this line have zero cost. The amount of formats in $I_{TCTPM}$ is also $n$ ($|\mathcal{F}| = n$). Node 1 in $I_{TCTPM}$ is the source node which holds the media object and nodes 2 to $n$ are the clients each requires the media in a format compatible to it’s index (node 2 demands format 2, node 3 demands format 3 etc...). Node $n + 1$ in $I_{TCTPM}$ is a simple network node.
For each network node in \( i \in \{1..n\} \) (the first \( n \) nodes in \( I_{TCTPM} \) network) we build a conversion graph which is represented by the connectivity of node \( i \) in \( I_{ham\_path} \). For example, if node 3 in graph \( I_{ham\_path} \) is connected to nodes 1, 4, 5 than node 3 in \( I_{TCTPM} \) is capable of the following conversions: \( f_3 \rightarrow f_1, f_3 \rightarrow f_4, f_3 \rightarrow f_5 \). Moreover, since the \( I_{ham\_path} \) graph is undirected, node 1, for example, is of course capable of converting \( f_1 \rightarrow f_3 \). All conversions of the first \( n \) nodes in \( I_{TCTPM} \) cost 1. Node \( n + 1 \) has a complete conversion ability (from each format to every format) but each of its conversions cost \( \rho \cdot (n - 1) \) (note that a hamiltonian path in \( I_{ham\_path} \) would cost \( n - 1 \)). The usage constraint of each of the first \( n \) conversion graphs equals 1 and the usage constraint of the \( n + 1 \) conversion graph equals \( \infty \), in other words, in our network the first \( n \) conversion graphs can be used only once and the last conversion graph can be used unbounded amount of times. Our above reduction is best described in Figure 6.2. The left graph in the figure is an \( I_{ham\_path} \) example and the right graph is the constructed \( I_{TCTPM} \). The non solid nodes in \( I_{TCTPM} \) figure are the source nodes and the clients.

Following the reduction we just described, it is obvious that if \( I_{ham\_path} \) has a Hamiltonian Path than we can use this path to decide how to reach all clients in \( I_{TCTPM} \) with the compatible format and without using any of the first \( n \) conversion graphs more than once. The Hamiltonian Path in \( I_{ham\_path} \) uses exactly \( n - 1 \) edges. These edges would indicate which single media conversion should take place in each of \( I_{TCTPM} \) first \( n \) conversion graphs. Note that since the conversion graphs in the MLMG are directed, in order to
move between formats in $I_{TCTPM}$ we must use a specific node, exactly the node through which the Hamiltonian path has traversed in $I_{ham\_path}$. Of-course for a complete $I_{TCTPM}$ solution, in this case, we need no more than $n - 1$ media conversions which gives a cost of $n - 1$ to the problem in hand. If $I_{ham\_path}$ has no Hamiltonian Path than in $I_{TCTPM}$ we must use at least one edge from the conversion graph of node $n + 1$ in order to create required formats and to maintain the problem’s constraint. An optimal $I_{TCTPM}$ solution in this case would cost more than $\rho \cdot (n - 1)$. Again, if we apply the $\rho$ - approximation algorithm, we presumably have, to solve $I_{TCTPM}$, and we check whether the cost of the obtained solution is less than $\rho \cdot (n - 1)$, we gain a polynomial time algorithm for the Hamiltonian path problem. This of-course contradicts the existence of a $\rho$ - approximation algorithm for our TCTPM problem. □
Chapter 7

The Degenerated Version Of The TPM Problem

In previous chapters we described approximation algorithms for various versions of the TPM problem. In all these versions we regarded the group of media formats \( \mathcal{F} \), and mainly the group’s size \(|\mathcal{F}| = F\), as an exogenous variable for our problem, and as such, we could not guarantee that \( F \) would have a small value. However, as the amount of media formats currently in use is relatively small, and as developers are straggling to converge the group of formats used today, we could look at a more degenerate version of the TPM problem. In such a version, one can consider the amount of formats \( F \) to be a very small constant which is an inherent factor of the problem. We name this TPM version the degenerated TPM (DTPM) problem. Since \( F \) is a small constant in the DTPM problem, in this chapter we consider algorithms with an exponential running time in \( F \) as a mean of improving our performance guarantee.

Note that the lower bounds presented for the TPM problem in Chapter 4 are not valid for the DTPM version, however, the problem is still NP-hard. For \( F = 1 \) the TPM is identical to the undirected Steiner tree problem, thus, we cannot hope to achieve an optimal solution even for cases in which \( F \) is very small and we are left, again, with trying to approximate it. An obvious approximation algorithm for the DTPM problem is our former BF algorithm. Since the BF algorithm guarantees an \( \frac{1}{K} \cdot (2F - 1) \)-approximation ratio, it can produce a very good result in cases where \( F \) is very small, exactly the cases dealt by the DTPM problem. However, embedded in the BF approximation ratio is the maximum compression factor \( K = \frac{f}{f_1} \), which can be significant and increase the actual performance guarantee. In Chapter 5 we presented an explanation indicating that this fac-
tor can not be too big in practical scenarios. However, in order to tackle the compression ratio issue from a theoretical point of view, we have offered to address the entire MLMG graph as a directed graph. Then, by running a known algorithm for directed Steiner tree problem (DSP) we get an $O(n^k)$ approximation ratio that does not include factor $K$. In this chapter we present a second theoretical approach to overcome the factor $K$ problem. This approach does not address the entire MLMG as directed graph but benefits from the small number of system’s formats $F$ in the DTPM case. For this very practical TPM version our approximation algorithm produces an $O(\log n)$ performance ratio that does not include the factor $K$. In order to achieve this ratio we use HST trees first introduced by Bartal [6] as briefly explained in Section 5.3. For that purpose, we first present a polynomial time algorithm which produces an optimal solution for the DTPM problem on trees. Then, we show how to use this algorithm for a general undirected graph and prove the $O(\log n)$ performance ratio.

7.1 An optimal cost algorithm for DTPM on trees

In this section we present a dynamic programming algorithm which produces an optimal solution for the degenerate TPM problem on a weighted tree graph. The algorithm finds a minimal cost solution for sending a media object from a source, located at the tree’s root, to the clients located at the tree’s leaves. The algorithm runs in polynomial time with regards to the amount of network nodes $n$, but the multiplication factor is exponential with respect to the number of formats, $F$. Recall that in Chapter 4, following a reduction from Directed Steiner Tree (DST) problem, we proved that the original TPM problem is NP-hard even on a tree graph.

Algorithm DTT (for DTPM on Trees), described below, receives as an input the tree’s root and the vector of formats recognized by the system. W.l.o.g, we assume here that the source node, which holds the media object in the original format $f_1$, is located at the tree’s root and that all designated clients are located at the tree’s leaves. Note that if a client, or the media source, are located at an internal tree node, one can connect such a node to a zero weight edge and create a new leave or root as needed. We also assume that each node in the graph holds a data structures containing the following data: The nodes’ connectively to their children (descendent nodes) and parent. The nodes’ converting graph (if exists). The nodes’ desired formats (incase the node is a client)

\[\text{\footnotesize Since the graph is a tree, a node can have several children but only a single parent.}\]
and most importantly an internal data structure, mainly a matrix, holding the cost of a suboptimal solution as we elaborate below.

Using recursion the DTT algorithm starts its calculation with the tree’s leaves and traverses back towards the root. At each of the tree’s nodes the algorithm finds the cost of a suboptimal solution in which the current node represents the root. In order to calculate this suboptimal solution, each node holds a matrix which contains the costs of all possible ways for our node to send and receive different media formats. The calculation of each matrix cell is done using the compatible cells of the matrices found at the node’s children and the node’s own transcoding capabilities. At the end of the algorithm run, all matrices will be filled and the problem’s solution will appear at the matrix of the root node.

We now describe the exact shape of the matrix found at each node. A cell in the matrix represents a foursome built of two pairs: \(((X,Y),(W,Z))\). The first pair is the group of formats the node receives from both directions, \(X\) marks the root’s direction and \(Y\) the leaves’ direction. The second pair is a bit more complicated. \(W\) marks the entire group of formats available to the node after receiving \(X \cup Y\) and from which the node can chose the formats to send to each of its children. \(Z\) marks the formats that should be sent upstream, from the node to its parent. Consider the following example: The cell \((012,34),(0124,3)\), marks the optimal cost of a solution which uses some formats from \((0,1,2,4) = (W)\) to send the data downstream, to the leaves, and format \(3 = (Z)\) upstream towards the parent. This cost is calculated under the assumption that the node just received formats \((0,1,2) = (X)\) from its parent and \((3,4) = (Y)\) from its children. In general, each column in the matrix contains all cells with the same pair \((X,Y)\), and each row in the matrix contains all cells with the same pair \((W,Z)\). Note the difference between calculating the minimal cost of sending the formats from \(W\) downstream, and sending the formats from \(Z\) upstream. Since we are dealing with a tree graph a node may have several children but only a single parent, thus, our algorithm must calculate which formats from \(W\) should be sent to each of the node’s children, it is possible that some formats are not used at all. Contrarily, we do not engage in the same process for sending formats \(Z\) to the node’s parents. Since the parent matrix is not yet calculated, as we use a dynamic programming algorithm, we have no base for choosing a subgroup of formats from \(Z\), thus, our calculation consider all formats from \(Z\) as part of the media object sent upstream. As mentioned earlier, the matrix holds all possible foursomes, thus, our cells contains all possible \(Z\) groups. Moreover, as there are \(F\) formats in the system there are \(2^{4F}\) possible \(((X,Y),(W,Z))\) foursomes and \(2^{4F}\) cells. Figure 7.1 depicts a typical matrix

51
for $F = 2$. In Appendix B we describe a way to reduce the size of the nodes’ matrices, thus, decreasing the running time of our algorithm. Generally speaking, the way to reduce matrix size is by eliminating matrix cells that cannot be a part of an optimal solution.

### 7.1.1 Calculating the cost of cell $((X,Y),(W,Z))$

The cost of the matrix cell $((X,Y),(W,Z))$ is the sum of three elements:

A) The cost of creating new formats using the nodes transcoding capabilities. More precisely, the cost of a forest in the node’s converting graph connecting the received formats $(X \cup Y)$ to the new formats $(W \setminus (X \cup Y))$. If a node cannot create all the new formats in $W$ then the cell’s cost is $\infty$.

B) The cost of sending formats from $W$ to the nodes’ children. Note that in order to calculate this cost we first need to chose the children that actually send to the node the formats in $Y$, only then we can decide which formats to send to the rest of the children who did not participate in that delivery. Again, if the node’s children cannot deliver all formats in $Y$ then the cell’s cost is $\infty$.

C) The cost of sending the formats in $Z$ to the node’s parent. Unlike part B), this phase is relatively simpler since the calculation is straightforward. No decision making should take place since the node has a single father.
The overall cost of a matrix cell is eventually the minimal cost sum of the above mentioned parts. However, though it appears that by finding the cheapest way to achieve each of the sub-costs separately we can gain an optimal solution to a given foursome (cell), this is not the case. An optimal DTPM solution on trees should not be only the cheapest but also describe a feasible flow of stream from the media source to all clients. Unfortunately it is highly possible that if we choose parts \( A \) and \( B \) separately we will construct a solution that does not guarantee the existence of a path from the root to each client. Figure 7.2 depicts a MLMG tree graph for a part of a tree network with 4 formats. The figure is an example of a scenario in which combination of optimal solutions to parts \( A \) and \( B \) does not describe a feasible flow of stream. The dotted arcs in the figure represent a possible solution for the matrix cell \( ((1,34)(1234, -)) \) which is calculated for the large node in the network (remember that all 4 large nodes in the MLMG actually represent a single network element). In step \( A \), we are choosing the cheapest way to create format 2 using the node’s transcoding capabilities. In this example the chosen optimal solution was a simple conversion from format 4 to format 2. In step \( B \), we need to choose which formats to send to each of the node’s children. For that purpose we first look for the best way to receive formats 3, 4 from the node’s children. In the example the optimal result was to receive format 3 from one child and format 4 from the other. Then we chose to send the children all of \( W \) formats, as depicted in the figure. Note, that by combining the results of steps \( A \) and \( B \) we have created an infeasible DTPM solution. As depicted in Figure 7.2, the tree’s bottom nodes receive formats that are not created in a logical
manner. The large node, for which we calculated the matrix cell, sends to three of its 
grandchildren formats 2, 3, 4 without connecting them to the format received from its 
parent - format 1.

In order to generate a feasible solution for a matrix cell, the DTT algorithm, found be-
low, must check that the proposed solution creates a path connecting the formats received 
from the node’s parent to each of the designated clients. Our algorithm simply calculates 
all possible combinations of steps A and B mentioned above, and chooses the minimum 
cost combination that also represent a feasible stream of the media object. It is simple 
to see that step C cannot jeopardize the feasibility of a solution, thus, after guaranteeing 
a feasible and optimal scenario for steps A and B the algorithm adds the minimal cost 
result of step C.

Algorithm 2 (DTT): Dynamic Programming Algorithm for DTPM on Trees

\[
\text{Input} \quad \text{treeNode } N, \text{ Vector } F
\]

\[
\text{for } A = 0 \text{ to } A = \#N.\text{sons do}
\]

\[
\text{DTT}(N.\text{sons}[i], F) \{ \text{Recursive call} \}
\]

\[
\text{end for}
\]

\[
N.V \leftarrow \text{createVector}(F) ^2
\]

\[
\text{for } i = 0 \text{ to } i = N.V.\text{length do}
\]

\[
\text{if } N \text{ is a root and } V[i].X \neq f_1 \text{ then}
\]

\[
N.V[i].\text{cost} = \infty ^3
\]

\[
\text{end if}
\]

\[
\text{if } N \text{ is a leaf and } N.V[i].Y \neq \emptyset \text{ then}
\]

\[
N.V[i].\text{cost} = \infty ^4
\]

\[
\text{end if}
\]

\[
C \leftarrow \infty
\]

\[
G \leftarrow \emptyset , S \leftarrow \emptyset \{ \text{Priority Queues} \}
\]

\[
S \leftarrow \text{SteinerForests}(N, N.V[i].X \cup N.V[i].Y, N.V[i].W \setminus (N.V[i].X \cup N.V[i].Y)) \{ \text{See Appendix A Section A.1} \}
\]

\[
G \leftarrow \text{SendingToChildren}(N, N.V[i].Y, N.V[i].W) \{ \text{See Appendix A Section A.2 and Appendix C} \}
\]

\[
C.\text{Cost} = \infty \{ \text{Holds the cost of an optimal solution} \}
\]

\[
C.\text{Group} = \emptyset \{ \text{Holds the scheme according to which formats should be created and sent to the children} \}
\]

\[
\text{while } S \neq \emptyset \text{ do}
\]

\[
\text{Forest } T = S.\text{POP}() \{ T \text{ is a Steiner Forest representing local transcoding using the node’s converting graph} \}
\]

\[
\text{for } i = 0 \text{ to } i = G.\text{length} - 1 \text{ do}
\]

\[
\text{if CheckFeasibility}(T, G[i]) == \text{ Feasible then}
\]

\[
\text{tmpCost} = \text{Cost}(T) + \text{Cost}(G[i])
\]

\[
\text{if tmpCost} < C.\text{Cost then}
\]

\[
C.\text{Cost} = \text{tmpCost}
\]

\[
C.\text{Group} \leftarrow G[i], T
\]

\[
\text{end if}
\]

\[
\text{end if}
\]

\[
\text{end for}
\]

\[
\text{end while}
\]

\[
C.\text{Cost} += \text{Cost of sendingZtotheparent} \{ \text{See Appendix A Section A.3} \}
\]

\[
N.V[i].\text{Cost} = C.\text{Cost}
\]

\[
N.V[i].\text{Group} = C.\text{Group}
\]

\[
\text{end for}
\]

\[
^2\text{Creating vector which holds all possible foursomes } (X, Y, W, Z).
\]

\[
^3\text{A root has a dummy provider of format } f_1.
\]

\[
^4\text{A leaf has no children, thus, all columns with } Y \neq \emptyset \text{ are irrelevant.}
\]
As stated before, algorithm DTT starts from the tree’s leaves and continues up to the root. On each node the algorithm fills a data structure representing all possible foursomes \((X, Y, W, Z)\). This data structure can be a complex matrix, as was explained above, or it can be a vector, as presented in the algorithm (marked by \(N.V\) in algorithm DTT where \(N\) is the current tree node). In order to find an optimal cost for a matrix cell, we are storing all possible partial costs in two priority queues. In the algorithm first Step we calculate all possible ways to create new formats according to the current foursome and by using the node’s converting graph. More formally, we create spanning trees on the converting graph to connect formats from \(X \cup Y\) to formats in \(W \setminus (X \cup Y)\). These spanning trees are stored in the first priority queue (for further explanation see Appendix A, Section A.1).

In the second phase we calculate all possible ways to send the node’s children formats from \(W\) while simultaneously receiving from some children the formats in \(Y\). This phase is relatively difficult since we have no previous knowledge on the number of children who sent the \(Y\) formats nor do we know which formats these children received in the first place. In Appendix A Section A.2 we present a simple technique to overcome these difficulties, in Appendix C we further improve this technique. The different options to send the formats in \(W\) while receiving formats \(Y\) are stored in the second priority queue (marked by \(G\) in Algorithm DTT). In the algorithm’s third step we are using a while loop to construct all combinations between the first two steps and choosing the minimal option which is also feasible. To the chosen option we are adding, at the last step, the cost of sending the \(Z\) formats to the node’s parent. All steps described here are embedded in a for loop which covers all cells in the node’s matrix.

At the end of the algorithm run, all nodes’ matrices are full, however, this does not finish our work on finding the optimal solution. In order to eventually find the optimal solution we need to backtrack the path indicated by the cell with the minimum cost at the root’s matrix. This backtracking concept can be found in many algorithms (such as Dijkstra’s algorithm [47]), however, we decided to skip its formal description in Algorithm DTT. Generally speaking, in order to backtrack the optimal solution, each matrix cell must hold, in addition to the cell’s cost, a data structure which points to the formats we decided to send to each of the node’s children. In algorithm DTT this data structure is \(N.V[i].Group\), where \(N.V[i]\) marks the \(i^{th}\) cell in node’s \(N\) matrix. Since the root receives a single format \(f_1 \in \mathcal{F}\), the problem’s optimal solution is found at the root’s matrix cells with \(X = f_1\) and with a minimal cost. Using the “Group” data structure each cell holds, we can reveal the exact formats the root sent to each children. For these children
we follow the same procedure. We choose a matrix cell with a minimal cost which has a compatible group \( X \) according to the formats sent to that child by the root. We continue with the same technique until we eventually reach the leaves and reveal which format each client received.

An extensive explanations on the three parts which comprise the cost of a matrix cell can be found in the thesis appendices. In the next section we analyze algorithm’s DTT running time using some calculations mentioned in these appendices.

7.1.2 DTT running time

In order to bound the running time of Algorithm DTT be observe that it uses an outer For loop and a recursion to go over all cells of all graph nodes’ matrices. In Appendix B we prove that a node’s matrix should not have more than \( O(3^2F) \) cells. Inside the outer loop the algorithm calculates the cell’s cost. In the first part of the cell cost calculation the algorithm computes all possible Steiner forests over the node’s converting graph. For that purpose we use the known Astar exhaustive search heuristic. This heuristic runs in time \( O(2^{F^2}) \) (See Appendix A, Section A.1). In the second part of the calculation, which runs sequentially to the first part, the algorithm creates all possible schemes to send and receive the media object from the node’s children. In Appendix A Section A.2 we prove that there are \( O(n^F \cdot F \cdot 2^{2F^2}) \) such schemes, and since each scheme requires checking all children matrices, the overall running time of this part is: \( O(n^F \cdot F \cdot 2^{2F^2}) \cdot O(3^2F \cdot n) = O(n^{F+1} \cdot 2^{2F^2}) \). In the cell cost calculation third part the algorithm uses a While loop which validates the solution’s feasibility. This inner loop runs sequentially to the two former parts and creates combinations of the results calculated previously. The inner loop running time is thus: \( O(2^{F^2}) \cdot O(n^{F+1} \cdot 2^{2F^2}) = O(n^{F+1} \cdot 2^{3F^2}) \). It is obvious that this part has the largest running time among all three parts of the cell cost calculation.

Recall that following algorithm’s DTT recursion, the outer loop repeats it self for each node in the network, hence, the overall running time of DTT, marked by \( \xi \) is:

\[
\xi = n \cdot O(3^2F) \cdot O(n^{F+1} \cdot 2^{3F^2})
\]

Since \( F \) is a small constant in the DTPM problem

\[
\xi = O(n^{F+2})
\]
7.2 Approximating DTPM by HST trees

In Section 5.3 of Chapter 5 we have briefly explained the idea of approximating metric spaces by a tree. This idea can be used to solve difficult optimization problems and produce a relatively “good” approximation ratios for these problems. In this section we elaborate on this topic mainly to show its applicability for our own DTPM problem. We prove that using our previously produced DTPM optimal solution for a tree graph we can generate an $O(\log n)$ performance ratio for the same problem on general graphs. In order to do that we need the following basic definition of a metric space approximation.

**Definition 7.2.1** A metric space $N = (V, d_N) \alpha - approximates a metric space $M = (V, d_M)$ if: For any $u, v \in V, d_M(u, v) \leq d_N(u, v) \leq \alpha \cdot d_M(u, v)$ where $V$ is the set of metric points and $d_N$ is the distance function.

Extending the above definition, in his 1996 paper [6], Y. Bartal has defined the concept of probabilistically-approximation of metric spaces. This concept was a key element in making the metric approximation field more applicable for improving approximation results for NP-hard problems.

**Definition 7.2.2** A set $S$ of metric spaces over $V \alpha - probabilistically approximates $M$ (over $V$) if there exists a probability distribution over metric spaces $N \in S$ such that:

$\forall N \in S, \forall u, v \in V, d_N(u, v) \geq d_M(u, v)$ (N dominates M).

$\forall u, v \in V, E(d_N(u, v)) \leq \alpha \cdot d_M(u, v)$

The following theorem is the key motivation to find “good” metric spaces which can $\alpha - probabilistically approximate other “difficult” metric spaces.

**Theorem 7.2.1** if the set of metric spaces $S$ can $\alpha - probabilistically approximates $M$ and $\forall N \in S$ there exists an algorithm that has a performance ratio $\beta$ for problem $P$, then there exists an algorithm for $P$ over $M$ that has a performance ratio $\alpha \cdot \beta$.

Motivated by the above theorem, which we later prove specifically for the DTPM problem, Bartal has also developed “good” metric spaces which he named **Hierarchically Well-Separated Trees** or HSTs for short. He proved that any metric space can be probabilistically approximated by set of HSTs which can produce a relatively small $\alpha$ (distortion) for the above theorem. We are using the following definition of HSTs.
**Definition 7.2.3** A k-hierarchically well-separated tree (k-HST) is a rooted tree with the following properties:

- The edge weight from any node to each of its children is the same.
- The edge weights along the path from the root to a leaf are decreasing by a factor of at least $K$.

Bartal’s main result was that every weighted connected graph $G$ can be $\alpha - probabilisticly - approximated$ by a set of $k - HSTs$ where $\alpha = O(k \log(n) \log \log(n))$. Bartal’s result was latter improved by Fakcharoenphol et.al. [20], who succeeded to reduce the stretch of any edge in the dominating tree metrics to an $O(\log(n))$. Bartal’s and Fakcharoenphol primarily result were used to the creation of randomized algorithms for general metric spaces since their tree building technique involves coins tosses. Fakcharoenphol et.al. work [20], and a later work by Charikar et.al [11], have showed ways to derandomize algorithms which use probabilistic metric space approximation, therefor, in the proof found below we neglected the probability issues and described our steps in a deterministic fashion.

Using the results we have presented so far in this section, we would like to create an $O(\log(n))$ approximation algorithm for the DTPM on general graph. Given the general undirected graph $G = (V, E)$ of DTPM we will first create set of $HSTs$ $T$ (using either Bartal’s or one of his successors techniques). For simplicity, from now on we address the set of trees as a single tree $T$. The original nodes of $G$ are $T's$ leaves and that includes the original root $r$. For clarity sake, in order to put $r$ at the tree’s root just rotate the tree without changing its general structure. Note that in $T$ only the new root $r$ and the set of leaves may have transcoding capabilities. We run the optimal dynamic programming algorithm on $T$ and mark by $A_T$ the DTT algorithm run on this created tree. We mark by $A_G$ the simulation of $A_T$ on the general graph $G$. Due to the directed arcs our MLMG is not a full metric space, however its horizontal undirected layers are. Moreover there is no problem simulating our algorithm run $A_T$ on the general graph MLMG. Each conversion that took place in $T$ would take place on the $G's$ MLMG in the compatible nodes. The only changes are the routing paths between any two nodes and not the formats conversions that took place.

Since our algorithm produces an optimal result on $T$ and since the tree metric dominates the original general graph metric, the following formula holds:
\[ Cost(A_G) \leq Cost(A_T) = Cost(Opt_T) \]

We now mark by \( Opt_G \) the optimal solution of the DTPM problem on graph \( G \) and we mark by \( B_T \) the optimal solution simulation on the tree \( T \). Again, there is no problem to simulate \( Opt_G \) on \( T \). Nodes the are connected in the optimal steiner tree which was formulated on \( G \)'s MLMG optimal solution would be connected in the spanning tree created on \( T \). Moreover, the same conversions that took place on \( G \) would take place on \( T \). According to the above equation and the fact that \( T \alpha \) probabilisticly approximate \( G \), we get:

\[ Cost(A_G) \leq Cost(A_T) = Cost(Opt_T) \leq Cost(B_T) \leq \alpha \cdot Cost(Opt_G) \]

Since we used HST trees construction we know that \( \alpha = O(\log n) \), thus:

\[ Cost(A_G) \leq O(\log(n)) \cdot Cost(Opt_G) \]

The \( A_G \) simulation has the same running time as the original algorithm \( A_T \). In other words, it can achieve an \( O(\log(n)) \) approximation in polynomial time in the amount of nodes \( n \) and exponential time in the amount of formats \( F \). The best approximation we have achieve so far, without the use of Bartal's technic, is due to the directed Steiner tree algorithm which deliverers only a \( O(\log^2(n)) \) approximation ratio in quasi polynomial time in \( n \) (not \( F \)). Since the DTPM considers \( n >> F \) the solution presented in this chapter may have an important practical impact.
Chapter 8

Performance Evaluation

In the previous chapters we described several algorithms for the TPM problem and its related subcases and generalizations. Our main considerations were the theoretical aspects of the problem and thus we mainly dealt with the algorithms’ approximation ratios. Note that approximation ratios guarantees the behavior of any instance of the problem, more specifically, it gives evident for the upper bound cost of the worst case scenario. Approximation ratio analysis does not deal with the average cost of the algorithms nor does it deal with common case scenario of the problem. In order to examine the actual performance of the suggested algorithms, we conducted a set of simulations’ runs. The simulations’ results not only show the advantage of our general technique, which allows transcoding at intermediate nodes, but also demonstrate the applicability of our algorithms and give evidence to the TPM problem practical importance. The simulation code was written in Java and run over a pentium 4 computer with a 400MHz Intel processor.

In the first simulation we considered a simple example which shows the advantages of transcoding at intermediate nodes in the network. Figure 8.1a depicts a simple star shape graph which contains 8 nodes in which node 1 represents the media origin (the root) and nodes 2 to 7 represent the clients. The media file was originally encoded in format $f_1$ but the clients ($v_2, ..., v_7$) would like to receive it in another format $f_2$. We assume in this example that all vertexes in the graph, except the source, are capable of hosting a transcoder to convert format $f_1$ to $f_2$. The conversion cost is the same at all nodes, furthermore, there is no compression, i.e. all formats has the same bits compression as the original encoded format $f_1$. Let us consider now two options of delivering the correct

\[1\]For convenience, instead of ignoring the source as a place for transcoding, one can assume that conversion at the source is very expensive and thus it does not pay to transcode at the root.
format ($f_2$) to all clients: Option 1, send the media file in the original format, $f_1$, from the source to all clients and convert the format to $f_2$ at the clients themselves. Option 2, send the media, in its original format, from the source to the intermediate node 8 in which the format conversion to $f_2$ would take place. Then, continue to deliver the media in the new format to all the clients. Figure 8.1b depicts a comparison between these two strategies. Note that option 1 uses 6 connection links (marked by solid edges) and 6 conversion operations to deliver the media, whereas option 2 uses 7 different connection links but only one converting operation. In general, if sending the media file from the source to the clients costs more than the formats’ conversion then it is preferred to use less connection links but more transcoding (as was offered by option 1). This claim should of course be reversed if transcoding costs more then the media delivery. In that case, option 2, which converts only once, should probably be chosen. Figure 8.1b depicts the cost of the two options as a function of the ratio between connections’ and conversions’ costs. One can see that as the ratio increases, meaning transcoding costs more than traversing over the communication links, it is cheaper to transcode at some intermediate node than to transcode at all clients. Note that in our cost function we combine the transcoding cost with the communication cost. This combination may be very difficult and ambiguous since each cost deals with a completely different set of network resources. The communication cost represents the utilization of the network links while transcoding cost represents the utilization of the network nodes, proxies routers or regular end point computers. As a result, the combination of the two costs, or more specifically the ratio between the basic communication cost and the basic transcoding cost, is an indicator to the importance of each one of the cost component in the overall evaluation. The tree found by the algorithm highly depends on the value of this ratio.

The second set of simulations, shown in Figure 8.2, depicts a practical implementation of the TPM problem and our $BF$ algorithm presented in Chapter 5. Consider the network demonstrated by the map which spans the 50 largest metropolitan areas in the United States [14]. The basic links’ costs in the graph are equal to the physical distance between the nodes they connect, links spanning greater distance cost more. We set the source node in this map to be Washington DC and all nodes (Washington DC included) are clients. We consider a format conversion scheme which deals with a realistic compression conversions that can take place with a MPEG-4 media stream. This conversion scheme is mentioned in [18] and contains five different profiles (formats) of MPEG-4. The compression ratios of the five formats are $1, 1, 0.5, 0.16, 0.032$ and the formats con-
Figure 8.1: A simple example for the TPM problem

a) A star shape graph with intermediate node transcoding

b) TPM solutions as a function of the ratio between conversions' and links' costs.
versions of the different network elements represent a single compression capability only. Each format can be converted only to the closest format according to their compression ratio \((f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow f_4 \rightarrow f_5)\). We assume in this simulation that equal amount of nodes (metropolitan areas) receive the media stream in a different format (Each format in the system has 10 nodes representing clients demanding that specific format). The transcoding costs from one format to another are chosen randomly for every graph node.

In this set of simulations we want to demonstrate the benefit of allowing an increasing amount of nodes to host transcoders. In each run of the simulation we added additional 5 transcoders and then checked the cost of a solution produced by running our BF algorithm. In the map presented, the triangular nodes are the ones capable of transcoding at a certain round of the simulation.

The result of this simulation is presented in the table of Figure 8.3. It is clear from the table that when more transcoders are available the final cost decreases. This analysis, that ignores issues like the ratio between conversions’ cost and connections’ cost, is obvious since our algorithm always look for the shortest path connecting the source to each layer in the MLMG. As more transcoders are opened the more options the algorithm has for paths finding between different formats. There are, however, situations in which despite the opening of more transcoding the TPM’s solution’s cost doesn’t decrease. This effect
A TPM solution with regards to the amount of transcoders opening

of diminishing return indicates the point where enough transcoders were already placed.

Next we compare the performance of the BF algorithm and the two other simple heuristics for solving the TPM problem. The first heuristic is format conversion at the clients. Meaning, the media origin sends the media stream in the original format $f_1$, and each client transcode it to suit its needed format. The second heuristic is transcoding at the source. The source creates copies of the media stream in all formats and multicast each media format to all compatible clients. We compare these two heuristics to the BF algorithm which can choose transformation at intermediate nodes in the network (our algorithm examines also the other two options). The comparison of the three methods is demonstrated by the ratio between each of the two simple heuristics solutions’ costs and our BF algorithm’s cost. The simulation was done on a graph representing an ISP’s (Internet Service Provider) network. We have created such a graph using the BRITE tool [40] over the Waxman’s model [48]. Our graph included 250 nodes and exactly 1250 edges.

Figure 8.4a shows the performance of the algorithm as a function of the ratio between the average communication cost and the average conversion cost. The formats’ conver-
Figure 8.4: Comparing the BF algorithm to the currently used schemes

The graphs in Figure 8.4 reflect the advantage of our BF algorithm. When increasing the average cost of conversions, as presented in 8.4a, our BF algorithm preserves its relative cost, whereas the other two trivial solutions suffer from an increase in cost. Thus, the ratio between the simple heuristics’ costs and our algorithm’s costs increases. On the average, transcoding at the clients costs 15 times more than our BF heuristic, and transcoding at the source costs more than twice our algorithm. Graph 8.4b demonstrates our algorithm’s advantage when dealing with large compressions. Although the other two solutions showed no decrease in cost, as compression increases, our algorithm does. As a consequence, the ratio between the two heuristics and our algorithm increases. On the average, transcoding at the clients costs 12 times more than our algorithm, and transcoding at the source costs more than twice the cost of our algorithm.
Chapter 9

Conclusion And Open Problems

In this thesis we have addressed the question of choosing where to perform transcodings when multicasting media streams in heterogenic networks. The question in hand is crucial especially when considering peer to peer applications where no end-users have enough computational strength to guarantee formats conversion at real time. We have showed that this problem is NP-hard and very difficult to approximate. Using a new model which we named the MLMG (Multi Layered Mixed Graph) model we developed new approximation algorithms which provide near optimal solutions for various special cases of the problem. We showed, using simulations, that our developed scheme, which enables intermediate transcoding, has a better performance in terms of cost than the commonly used solutions for the problem.

The results presented in this paper are only a first step for many interesting problems in both the theoretical and practical aspects alike. For the general TPM problem we presented an algorithm which guarantees an $O(F)$ approximation ratio but it also depends on the maximal compression ratio in the system $K$. Though we gave practical explanations for the reason this ratio cannot affect the algorithm performance we did not succeed to avoid it in the pure theoretical analysis. This is a question which remained open. We also examined, for example, the degenerated TPM (DTPM) problem in which the amount of formats the system recognizes is a small constant. We established an $O(\log n)$ performance for this problem, however, it remains to implement this algorithm and to test it in a simulated environment. Moreover, since we have no tighter lower bound for this problem than an obvious constant we do not know how good is the $O(\log n)$ performance ratio.

Viewing the giant leap, made in recent years, towards the establishment of a viable
heterogenic network, it seems that a reliable multicast of multimedia streams is not a faraway vision. Though currently it is mainly used in the TV cable industry we can expect that such a system will be part of the IP world in the near future. In order to really develop this pure IP based system, many more questions must be raised and answered: For example one should deal with questions related to scalable and robust protocols, questions related to quality of service guarantee, and many more questions related to specific deployment of vital network components.
Appendix A

Building The DTT Matrix Cell

This appendix includes analysis of the three parts comprising the cost of a matrix cell in algorithm DTT which is described in Chapter 7.

A.1 Local optimal tree

Recall that $W$ represents the group of formats we can use to send the media stream to the node’s children. Moreover the node already received the stream in formats $X \cup Y$, thus, the node can send that stream immediately using only these formats or it can choose to use its own transcoding capabilities to generate new formats from $W \setminus (X \cup Y)$. The cheapest way to create the new formats is to follow a Steiner forest on the node’s conversion graph. Unlike a Steiner tree on a directed graph (dgraph), which connects a single source to all clients using one single tree, a Steiner forest can connect the clients to multiple sources using several trees as the amount of different sources. In our case the sources of the Steiner forest are the $X \cup Y$ formats and the clients are the $W \setminus (X \cup Y)$ formats. Note that the relevant sets appear in the input of function SteinerForests found in algorithm DTT. It is easy to see that on a dgraph (As the node’s converting graph is) creating a Steiner forest is identical to creating a classical Steiner tree (simply create a dummy node and connect it using directed arcs with a zero weight to all sources of the Steiner forest). Finding a Steiner tree on dgraph is an NP-hard problem, as we discussed in previous chapters. There are, however, many algorithms which can produce an optimal solution but run in exponential time in the amount of graph nodes. One of the algorithms is the Astar exhaustive search that is mentioned in [18]. The Astar algorithm creates all possible trees and stores them in a data structure according to their overall cost. Though
in order to find the cheapest tree we need only to pop the top tree in the data structure, we are leaving the rest of the trees in their original order in case some trees produce an infeasible DTPM solution. The running time of such an exhaustive search is bounded by \( O(2^m) \), where \( m \) is the number of graph edges. In our case \( m \leq F^2 \), and since we can endure an exponential running time in \( F \), this heuristics is a reasonable approach.

### A.2 Cost of sending the media object downstream

The DTT algorithm’s second part seeks to find the minimal cost of sending the media object to all clients. For matrix cells which represent a scenario in which the current node also receives the media stream from its children (cells with \( Y \neq \emptyset \)) we also need to chose the children who participated in sending the stream back. Finding the optimal way to simultaneously send the media object downstream and receive it from the upstream is not a trivial task. Note that we know neither the amount of children participating in the upstream delivery, nor which formats the children received in the first place from the downstream. Our solution is to simply check all possible options and to chose the cheapest one which doesn’t jeopardize the feasibility of the overall solution. Though it appears that following such a scheme may take an exponential time in the node’s degree (the number of children) and thus making the trace infeasible, we show below that this is not the case.

Let us first construct a data structure representing all possible options to receive the formats in \( Y \). This data structure holds groups containing pairs of \((R,S)\) formats, where \( R \) is the set of formats a node’s child might receive and \( S \) is the set of formats the same child have sent back to the current node. Following is an example for the data this data structure needs to hold:

We examine a cell in some node’s matrix with the following foursome \((1,23,123,−)\) compatible to \((X,Y,W,Z)\) respectively. In this example we assume that the system recognizes only 3 formats \((1,2,3)\). The possible options for the return of formats \( Y = 23 \) are the following 6 groups:

\[
\begin{align*}
A & = (1,23) \\
B & = ((1,2)(1,3)) \\
C & = ((1,2)(2,3)) \\
D & = ((1,3)(3,2)) \\
E & = ((1,2)(12,3)) \\
F & = ((1,3)(13,2))
\end{align*}
\]

Group \( A \) represents a single child which received format \( f_1 \) and sent back both formats \( f_2 \) and \( f_3 \). Group \( B \) represent two children both received format \( f_1 \) and each returned a different format. Group \( C \) represents two children the first received format \( f_1 \) and returned
and the other used $f_2$ and sent back $f_3$. The rest of the groups are easy to follow.

Note the following two observations: A. The maximum amount of different $(R,S)$ pairs is no more than $2^{2F}$. B. In order to return the formats in $Y$, there is no need for more than $|Y|$ children. The amount of groups we need to construct is a function of the amount of pairs calculated in A and the size of $Y$ which cannot exceed $|F| = F$. We mark by $\gamma$ the amount of groups:

$$\gamma = \binom{2^F}{1} + \binom{2^F}{2} + \binom{2^F}{3} + ... + \binom{2^F}{Y} < (2^{2F}) + (2^{2F})^2 + ... + (2^{2F})|Y| < F \cdot 2^{2F^2}$$

As we pointed out, no group can have more than $F$ pairs, thus, the amount of overall pairs we need to check is less than $F^2 \cdot 2^{2F^2}$. In Appendix C we present a way to decrease this amount of pairs and thus improving algorithm DTT running time.

Given a group in the data structure, which represents a way for the current node to receive the $Y$ formats, we calculate all different options to construct it. For each $(R,S)$ pair in the given group we calculate the minimal cost for each child to fulfill it (to receive $R$ and to send $S$). A child should have a cell in their matrix in which $R \subseteq X$ and $S \subseteq Z$. Since for each child we are looking for the minimal cost cell compatible to the $(R,S)$ pair, it is obvious that in such a cell $R = X$ and $S = Z$. Note that to the cost of the child’s chosen cell we need to add the cost of sending the formats in $R$ from the node to that child. This additional cost is not part of the already calculated cost at the child cell. Following the above procedure we can find the minimal cost for constructing a group. We simply chose for each pair in the group the child who can carry it out with the minimal cost.

After calculating a group in the data structure and choosing the node’s children who carry it out, we need to send the rest of the children formats from the cell’s $W$ set. This process is almost similar to the previous one except here we need only to create subgroups $R$ where $R \subseteq W$ and look in the children cells with $R = X$ but $Z = \emptyset$. For each compatible cell we examine in the children matrices we need to add the cost of sending the formats from the current node to the examined child.

In this process the following problem that may occur. At first we chose the children who sent the formats in $Y$ at the minimal cost according to the given constructed group. Then, we chose the optimal way for the rest of the children to receive the formats from $W$. Since we chose the optimal solution for each part separately, it is possible that a different combination of children would produce a solution with a cheaper cost. In order to solve this problem we are storing for a given group all possible ways to construct it while sending the formats in $W$ to the rest of the children. As an example for this solution we
focus on the following group: \( G = ((1,2)(1,3)) \). Here we describe a scenario in which the current node receives from two of its children formats 2 and 3 while sending them format 1. Assume the current node has 5 children. We construct all possible solutions in which two children carry group \( G \) and the other three children are delivered with their desired formats without sending any format back. Since any group cannot have more than \( F \) pairs not more than \( F \) children can participate in any group, thus, the overall amount of solutions does not exceed \( \left( \frac{n}{F} \right) < n^F \) where \( n \) is the nodes degree (the amount of node’s children). Note that we did not exceed the polynomial time guarantee of \( n \). Following this calculation we can now summarize this section: Data structure \( G \) found at algorithm DTT holds all the groups which represent a way to return the \( Y \) formats, and for each such group all possible combinations of children who can participate in the group. Like data structure \( S \), \( G \) should also be an ordered data structure (a priority Q for example). The overall amount of cells in \( G \) would therefore be less than \( n^F \cdot \gamma = n^F \cdot F \cdot 2^{2F^2} \).

### A.3 Adding the upstream formats

After adding all previous calculations to the total cost of a specific matrix cell, we are left with the formats needed to be sent to the node’s ancestor (parent). This phase requires only adding the cost of sending each format from \( Z \) on the uplink connecting the current node to its parent. The following sum represent this cost:

\[
\text{linkCost}(N,N.parent) \cdot \sum_{i=1}^{\lfloor \frac{|z|}{F} \rfloor} f_z[i]
\]
Appendix B

Reducing The DTT Matrix Size

As explained in Chapter 7, we use Bartal’s technique to solve the DTPM problem. For that purpose we first find an optimal solution for the problem on trees. In Section 7.1 we presented a dynamic programming algorithm that finds such an optimal solution and showed that it is based on the construction of matrices at each node. These matrices represent a suboptimal solution for the DTPM problem in which the current node is the root. The suboptimal solution is found following a process in which all matrix cells are filled with a solution cost compatible to the media formats the node received from both its children and its parents.

It is obvious that decreasing the matrix size without jeopardizing the ability to find an optimal solution improves the running time of our algorithm. We mentioned in Section 7.1 that the basic size of the nodes’ matrices is the amount of possible \((X, Y, W, Z)\) foursomes and since there are \(F\) formats in the system there are \(2^{4F}\) foursomes, thus, \(2^{4F}\) cells. However, though there are many foursomes, it is easy to see that some of them are practically irrelevant since they describe no logical flow of streams and can never be part of the algorithm’s final solution. The following set of three rules point to an actual relation between the four sets of formats in each foursome. These rules can be used to decrease, to a large extent, the size of the matrices thus helping us skip irrelevant calculations and improve the algorithm’s running time.

A) \(Y\) can contain any subset of \(F\) that is disjointed to its \(X\) pair (\(\forall (X, Y)\) pair : \(X \cap Y = \emptyset\)).

EXPLANATION: No optimal solution will pay twice for receiving a format from both the root and the leaves. Using the network multicast capability the node needs only a single instance of any format to send it to more than one direction.
B) \( W \) must contain any subset of \( \mathcal{F} \) that consist of \( X \cup Y \). \( (X \cup Y) \subseteq W \).

**Explanation:** Network nodes that receive \( X \) and \( Y \) formats can send the media object in a subgroup of \( X \cup Y \) format and it can also use \( X \cup Y \) to create new formats using the node’s converting graph. In any case, \( X \cup Y \) formats are always available.

C) \( Z \) must contain any subset of \( \mathcal{F} \) that is in \( W \) but not in \( X \). \( Z \subseteq (W \setminus X) \).

**Explanation:** Since we have received formats \( X \) from the node’s parent it is not reasonable to send these formats back. Moreover, in order to send formats back to a parent a node has to either receive these formats himself or to create it using its own transcoding graph.

Following the above three rules it is easy to show that the number of actual relevant cells in the matrix is less than \( 3^{2|F|} \) (which is smaller than \( 2^{4|F|} \) for every \( F \in \mathbb{N} \)). We start by calculating the amount of columns in the new matrix. Remember that the number of columns equals to the number of relevant \((X,Y)\) pairs. Rule A states that \( X \cap Y = \emptyset \):

When \( X \) consists of only a single format there are \( 2^{F-1} \) compatible \( Y \) subsets. Where \( X \) consists of two formats there are \( 2^{F-2} \) compatible \( Y \) subsets and so on. By defining \( \mathcal{N} \) to be the number of columns we have the following:

\[
\mathcal{N} = (\binom{F}{1})2^{(F-1)} + (\binom{F}{2})2^{(F-2)} + (\binom{F}{3})2^{(F-3)} + \ldots + (\binom{F}{F})2^{(0)}.
\]

The above equation is Newton’s Binomial Formula without its first element, thus:

\[
\mathcal{N} = (2 + 1)^F - (\binom{F}{0})2^F 1^0 = 3^F - 2^F.
\]

In order to calculate the amount of rows in the matrix we use rules \( B \) and \( C \). However, it appears that the number of rows is not constant and depends on the respective column. In other words, the number of column cells is a function of the relevant \((X,Y)\) pair. Rule B suggests that the number of possible subsets in \( W \) is \( 2^{(F - |XY|)} \). Rule C suggests that the number of possible \( Z \)s is \( 2^{(|W| - |X|)} \). Therefore, as \( X \) and \( Y \) contain less formats, the amount of different \((W,Z)\) pairs increases. When \( X \) contains a single format and \( Y \) is an empty set, both \( W \) and \( Z \) can receive the maximum amount of formats’ subsets (there is no meaning for \( X \) and \( Y \) to be empty). The following algebraic formula describes the maximum amount of \((W,Z)\) pairs that, as explained above, are compatible to the \((X,-)\) column in the matrix where \(|X| = 1\). The maximum amount of pairs equals to the maximum amount of rows a matrix column can have. We denote this number by \( \mathcal{M} \):
\[ \mathcal{M} = (F^{-1})2^0 + (F^{-1})2^1 + (F^{-1})2^2 + (F^{-1})2^3 + \ldots + (F^{-1})2^{F-1}. \]

Note that the above formula is again Newton’s Binomial Theorem:

\[ \mathcal{M} = (1 + 2)^{F-1} = 3^{F-1} \]

Now, by multiplying the number of matrix columns with the maximum amount of rows we get the size of new matrix.

**Corollary B.0.1** The DTPM matrices size, needed for the dynamic programming algorithm on trees is: \( N \cdot M = (3^F - 2^F) \cdot 3^{(F-1)} \ll 3^{2F}. \)
Appendix C

Improving Step B In The DTT Algorithm

In algorithm DTT, which creates an optimal solution for the DTPM problem on trees, we construct many ordered data structures. One of these data structures, which should help us choose how to deliver data downstream, holds all possible options to both send and receive the media objects to and from the node’s children. In Appendix A, Section A.2 we explained how to construct the cells of this data structure. We showed that we need to create \((R,S)\) pairs and to build groups of pairs representing the different schemes of Receiving and Sending media formats from/to the node’s children. We counted the amount of groups that should be created and saw that as a result the size of the data structure is big. In this appendix we identify a way to reduce this size. Generally speaking, though the number of \((R,S)\) pairs and the number of constructed groups are both big, many of them are irrelevant since they do not reflect a reasonable way the formats in \(Y\) are returned to the network node.

Next we present several rules that characterize relevant pairs and relevant groups that needed to be considered. Using these rules, we can avoid the irrelevant construction of cells in the data structure and reduce our implementation cost. Some rules are direct result of the improved matrices characterization which we described in Appendix B.

\((R,S)\) pair rules:

- \(R,S \neq \emptyset\).
  Explanation: If \(Y \neq \emptyset\) some children have returned the media stream and must have originally get it otherwise we describe an infeasible solution.
• \( S \subseteq Y \).
Explanation: \( S \) is the group of formats returned to the current node by one of the children. Since \( Y \) describe the entire group of formats the node received from its children it is obvious that the formats in \( S \) must be in it.
• \( R \subseteq W \).
Explanation: The children which returned the formats in \( Y \) must have accepted formats from \( W \) in the first place since \( W \) represents the entire group of formats the node can send both upstream and downstream.
• \( R \cap S = \emptyset \).
Explanation: If a media format was sent to a child (making it in the group \( R \)) there is no need for the same child to return it since the current node already has it.

**Group rules:**

We mark by \( Q \) a given group and by \( |Q| \) the amount of \((R,S)\) pairs in the group.
• \( |Q| \leq |Y| \).
Explanation: In order to return \( |Y| \) different formats to the current node we do not need more than \( |Y| \) children. Since each \((R,S)\) pair represents the behavior of a single child, there shouldn’t be more than \( |Y| \) such pairs in \( Q \).
• \( \biguplus_{i=1}^{Q} S_i = Y \).
Explanation: The union of formats in \( S \) groups found in the different pairs of \( Q \) should be exactly the formats in \( Y \). This is true since on one hand the node’s children must return all formats in \( Y \) and on the other hand we are looking for a minimal cost solution, thus, there is no need to return a format that is not in \( Y \).
• \( \forall i,j \in Q \ S_i \cap S_j = \emptyset \).
Explanation: For every two pairs \( i,j \in Q \) there is no media format that can be found in both there \( S \) groups respectively. This is true since we are looking for a minimal cost solution. The current node should not need more than a single instance of a media format in order to distribute it to other nodes, thus, the node’s children needs only to send back a single copy of each format.

Many other group rules are direct result of the pair rules thus we have decided to avoid mentioning them.


Bibliography


