This M.Sc. thesis was defended on October 16, 2019, to obtain the title of
Magíster en Investigación de Operaciones

A New Effective Mathematical Programming Model to Design CDN Topology

Author: Milton Bentos

Thesis Advisors: PhD. Franco Robledo
PhD. Claudio Risso

Jury: PhD. Pablo Sartor
MSc. Omar Viera
MSc. Alfredo Piria

Instituto de Computación
Facultad de Ingeniería

Universidad de la República October 2019

To my late father, Evio Bentos.

Acknowledgements

It has been stated that a book is a kind of synergy between an author and those who lend inspiration and support. I could not agree more with this statement, which intents to summarize my indebtedness to my tutors, Ph.D. Franco Robledo, and Ph.D. Claudio Risso. Thanks must also be offered to Ph.D. Graciela Ferreira, who helped me make the right inquiries at the right time. Although I have been supported and encouraged to reach this point by many people, I cannot fail to express my gratitude to Ph.D. Hector Cancela, Engineers Giovanna Fortez, and Leticia Grassi, A/P Fernando Yurisich, and Gustavo Gorenick. I am especially grateful to IBM Uruguay, whose IBM Academic Initiative allowed us to take advantage of the amazing features from their CPLEX. Last but not least, I must recognize that this work could not have completed without the affection and patience of my beloved wife and sons, Rita, Lucio, and Bruno.

Abstract

The Steiner Tree Problem is an umbrella of combinatorial optimization problems in graphs, most of them NP-Hard, within which, the Steiner Tree Problem in graphs (STP) is perhaps one of the most famous and widely studied. The STP consists in optimally interconnect a given set of terminal or mandatory nodes within a graph with edges of positive weights, eventually using other optional nodes. It has a wide range of applications from circuit layouts to network design, so plenty of models to find its exact solutions have been crafted. Traditionally, due to its intrinsic complexity, heuristic approaches have been used to find good quality solutions to the STP. Currently, the outstanding computing power resulting from combining developments in hardware and software capabilities makes it possible to rely upon exact formulations and generic algorithms to solve complex instances of the problem. This work introduces a flow-based mixed-integer problem formulation (MIP) for the STP using the SteinLib, a reference test-set repository. Later on, that MIP formulation is modified to solve the Quality of Service Multicast Tree problem (QoSTP). To the best of our knowledge, there is no previous MIP formulation. While existing approaches go all the way of approximation algorithms to find solutions, this MIP formulation shows promising experimental results. Optimal solutions are found for several instances, while low feasible-to-optimal gaps were obtained for most of the remaining ones.

Keywords: Flow-based model, flexible model, effective optimization, linealization, mixed-integer problem formulation, Ford-Fulkerson algorithm, Steiner Tree Problem, Quality of Service Multicast Tree Problem .

Table of contents

Chapter 1

Introduction

During the last decades, Information and Communication Technologies have been evolved and progressively merged in most of our daily tasks, up to a point where modern human and economic activities are hardly imaginable without the support of them. This process is developed at multiple levels of abstraction and functionality. It is out of the question that the Internet is the de facto global means to connect end-users with information. Furthermore, humanity is on the brink of the second generation of application changes due to the expansion of Artificial Intelligence. Within such an enormous and dynamic ecosystem, this work focuses on Content Delivery Networks (CDN). Before going into technical details, it is worth revising some of the changes telecommunications have been enduring over the last twenty years. Considering the radio or the telephone as the original milestone, modern telecommunications exist since the late 19^{th} century. Later, other communications services were added to the offer, such as television, telex (an old messaging system), cable-TV, satellite-TV, mobile telephony, etc. A subtle but crucial aspect of that arrangement of services is the fact they were independent. That is, there were technological standards, regulations, and technologies specific to each niche market. Thus, separate infrastructure was necessary to support different services. The stiffness of the whole package conspired against the diversification and evolution of those services. The massification of the Internet began in the late '90s. It was designed from scratch to be application unaware and it also supported to be deployed over legacy infrastructure. Many of us still have in mind those times when dial-up modems used the public switched telephone network as a means for accessing the Internet. The flexibility of the Internet promoted the development of new end-to-end applications, which would not need now huge investments in public infrastructure to come to life. Later, legacy services (telephone, messaging, video content) moved over this new immense infrastructure as residual traffic, at a pace so intense that in only twenty years, the infrastructure that supported legacy services is on its way to becoming obsolete.

Nowadays, massive content is sustained over Content Delivery Networks (CDNs). They are the result of the fusion of information platforms, i.g. clusters of servers that reside at data centers worldwide distributed and the vast Internet's telecommunications infrastructure. Precisely, this work aims upon the efficient design of such kind of networks. Most popular public content such as that of *Netflix*, *Facebook* and *Youtube* are conveyed over CDN owned by these corporations. Other popular content regularly accessed by many users is that of software updates. For instance, Apple Inc. and Microsoft Corporation use a third-party CDN (Akamai Technologies) to keep update packages worldwide distributed. An objective of a CDN is to keep content as close to the end-users as possible, ideally in the nearest data center, so the user can attain the best Quality of Experience (QoE). Consequently, lowering the global volume of information traversing the Internet. For such a goal, the nodes of a CDN (servers clusters within a data center) keep copies of the content. Whenever a user is trying to access some content of a CDN (e.g. a Youtube's video), the user is redirected towards the nearest node in the world. If that node already has a copy of the required information, the data is utterly sent to the requester. Whether the node does not have that copy, it solicits the content to the nearest nodes for them to feed it. The resources these copies consume while being relayed over the Internet depends on the CDN's topology, that is, of which nodes are to be used and what adjacencies are to be set among them. How to design a CDN to use the minimum expected number of bandwidth resources is the object of this work. Network design and planning are a traditional area of active research in applied mathematics. In particular, designing telecommunication networks are highly complex and frequently time-consuming tasks, then network design usually pertains to the category of NP-Hard problems. That means that no algorithm of polynomial time complexity (in the size of the input instance) is known to find solutions for them. For example purposes, we mention problems related to the design of optical networks, where the *Minimum Weight Two Connected Spanning Network* (MW2CSN, [Monma 1990], [Bienstock 1990]) and the *Steiner Two-Node-Survivable Network Problem* (STNSNP, [Baïou 1996]) are two reference cases. Those examples aim at designing resilient physical infrastructure (optical fibers and the subterranean conduits to deploy them), over which telecommunication networks are assembled. Essentially, they intend to find the minimum-cost network that connects some of its nodes with at least two physically independent paths. That guarantees that services can recover after a single failure of a node or link in the network. A CDN, on the other hand, is not a physical but logical network, deployed through connections over the Internet, so we reasonably assume that connections resiliency is provided by the Internet itself. As we see later on, in spite of the previous fact, the intrinsic complexity of designing an optimal CDN also falls into the NP-Hard category.

A fundamental premise this work relies upon one of the main features of the Internet. Indeed, as we previously mentioned, optical networks are historically designed to be resilient, which in that context means that, they count at least two physically independent paths between nodes in their backbones. The number is usually greater than two, except for a group of second priority stub nodes. The second level of abstraction within the Internet stack is the set of routers and links that compound the present Internet, which use optical networks to get connected. Any ISP around the world is aware of that characteristic, so they hire optical capacity between their nodes seeking for a certain level of physical independence, which guarantees that most connections in its network are immune to such simple failures. Besides, dynamic routing control protocols identify these malfunctions and then reconfigure the traffic routes of the clients so that it is not affected. So, communications over the Internet are highly resilient because they inherit the Internet's resiliency itself. The other type of component in a CDN is the content servers. A cluster of servers within a data center conforms to a node of the CDN. By design, the cluster functions as a whole, where active servers share the load. Whenever a server fails, it is simply removed from the cluster until its operability is restored. Regarding the services that data centers offer, every service in a mid-grade data center is duplicated. Telecommunications are highly resilient even beyond the regular Internet standard. In a data center, power supply between electricity providers and servers is bypassed with Uninterruptible Power Sources (UPS), which are complemented with Emergency/Backup Generators. Besides, high-grade data centers connect with at least two independent electricity providers. The remaining complementary services, like air conditioning, are also duplicated. The main consequence of that underlying resiliency is that we do not have to concern with it during our design. The simplest and yet realistic optimal structure for a CDN is a minimal spanning tree, i.e., a set of nodes to be connected with the minimum cost. It is a well known theoretical result that the minimal network to connect a set of nodes is a tree. In this work, we aim at a tree not only minimal but of minimum cost. Stated so, we might think that the goal is to span a preselected set of nodes, where the problem reduces to *Minimum Spanning Tree*. The MST is known to be of polynomial complexity since there are at least two famous algorithms [Kruskal 1956] and [Prim 1957] of polynomial time complexity capable of finding solutions. Later, [Minoux 1990] introduced an algorithm able to solve accurately the MST problem of a graph G given that of an $(n-1)$-node G's subgraph's solution is known. As it runs in $O(n)$ time, Minoux's algorithm is computationally more efficient than Kruskal and Prim's ones. Thus, when heuristics intended to solve SPG [1] are based on Minoux, execution times are dramatically reduced. See [Martins 2000], [Ribeiro 2002], [Kruskal 1956] and [Prim 1957].

[1] Steiner Problem in Graphs

However, no approach can assume that all definite nodes are known for sure in advance. In addition to the mandatory nodes, the solution may include some optional nodes if they contribute to decreasing the cumulative cost. The previous problem formulation is closely related to the well-known *Steiner Tree Problem*. The Steiner Tree Problem is an umbrella of many related problems, whose summary can be found in [Hauptmann 2013]. This work regards two problems of that list. The first of them is the general Steiner Tree Problem in graphs or STP, which is stated as follows. Given an undirected graph $G = (V,E)$, with edge costs $c : E \to \mathbb{R}^+$ and a set of terminal nodes $T \subseteq V$, the goal is on finding a tree subgraph of G spanning to all terminal nodes, whose cost is minimum. Formally, we are seeking for a cost-optimal connected subgraph $G_T = (V_T,E_T)$, such that $T \subseteq V_T \subseteq V$ and $E_T \subseteq E$. The Steiner Tree Problem in graphs can be effectively solved for some families of instances. For example, the *all terminal nodes* case ($V = T$) is the Minimal Spanning Tree (MST), whose exact solution can be obtained in time $O(|E| + |V|log(|V|))$. When T equals two, the solution is the shortest path between those two terminals, which is a problem of time complexity $O(|E| + |V|log(|V|))$. Although there are a few more exceptions, as a general rule the Steiner Tree Problem in graphs is an NP-Hard problem [Karp 1972], even for grid graphs [Garey 1977]. The solution of Steiner tree problems has received considerable and strongly growing attention in the last thirty years, spanning from exact methods (see [Goemans 1993]) to heuristic ones (see [Duin 1994]). Complementarily, the problem of finding good quality lower and upper bounds for the optimal cost (i.e. relaxations) has been widely studied too. [Borradaile 2009] is a good example of approximation algorithms bounds. We strongly recommend [Polzin 2003] as a complete survey for problem variants, reductions, and efficient algorithms. Excellent surveys are given in [Winter 1987], [Maculan 1987], [Hwang 1992] and [Hwang 1992]. To solve the STP, [Aneja 1980] introduces a row generation algorithm based on an undirected formulation, [Dreyfus 1971] and [Lawler 1976] use dynamic programming techniques, [Beasley 1984] and [Beasley 1989] present a Lagrangean relaxation approach, [Wong 1984] describes a dual ascent method, [Lucena 1993] combines Lagrangean and polyhedral methods, while [Chopra 1992] develop a branch-and-cut algorithm. Also, [Thorsten 1998] presents the implementation of a branch-and-cut algorithm for solving Steiner tree problems in graphs to optimality. In particular, polyhedral methods have turned out to be quite powerful in finding optimal solutions for various Steiner tree problems thanks to an improvement in the understanding of the associated polyhedra, the availability of fast and robust LP solvers, and the experience gained about turning the theory into an algorithmic tool. Some interesting exact formulations for the SGP are presented by [Stanojevic 2006] [Diané 2006] and [Wang 2006] in his Ph.D. dissertation.

Recently, [Siebert 2018] introduced a set of integer programs (IPs) for the Steiner tree problem, when the number of terminals is fixed. Almost simultaneously, [Hua 2018] proposed a different set of integer programs (IPs) for the Steiner tree problem based on regular sparse grids. Previously, [Leggieri 2014] proposed an exact solution approach for the *Tree Problem with Delays (STPD)*. As a generalized version of the Steiner tree problem applied to multicast routing, it uses SteinLib's instances with sparse graphs to get experimental results. This work introduces an innovative mixed-integer programming (MIP) model to solve the STP. The model is based upon a flow problem formulation detailed in [Goemans 1993], over which, we will elaborate in detail later on this document. In a nutshell, if we ask flow-balance for every node in a graph G = (V, E) and inject a unit of traffic from every terminal node in T but one (R), a path must exist from all remaining terminal nodes to the given node R to drain that unit of flow injected by each one, so the result must be connected. If the set of edges used to course those flows is of minimum cost, the solution is connected and minimal, and it is, therefore, a tree. When [Goemans 1993] was published, the scale of problems solvable with standard solvers was quite limited. Nowadays, the advancement in computing power coming from hardware, combined with the improvement of more efficient algorithms, makes it possible to solve many real-world size instances. To sustain the last claim, we used IBM CPLEX to solve several instances of a standard test set instances, the SteinLib Testdata Library[2] . The SteinLib is a historical and up-to-date repository of STP instances, see [Koch 2001].

The second problem this work regards is the construction of an optimal tree for a CDN. Also known as QoSMT, Quality of Service Multicast Tree is listed in [Hauptmann 2013], where is reported as follows: *"Approximable within 1.960 for the case of two non-zero rates. Approximable within 3.802 for the case of the unbounded number of rates. For the case of three non-zero rates, the problem admits a 1.522 approximation algorithm"*. Later on, we describe the details of the QoSMT. By now, we mention that besides the terminal (T) and optional nodes ($V\backslash T$) with positive-cost edges of the STP, the QoSMT contains a predefined root node $t_0 \in T$. This root could be considered as the central node of the would-be planned CDN, where all available media to be shared reside. Each other terminal t_i node has a known *rate* r_i, which represents the importance of that terminal node as a content client. The higher the rate, the higher the number of different copies to be sent to that node from the root. Thus, the cost of the immediate link upwards in the way to the root of the multicast tree should be multiplied by r_i to be accounted for. Furthermore, subsequent links in the path to the root must be penalized with the maximum of those rates underneath them. Finally, optional nodes can be used to improve the quality of the solution.

[2]http://steinlib.zib.de/steinlib.php

QoSMT is harder than STP. We might think of STP as a particular case of QoSMT where all rates are equal to 1. Existing solutions to QoSMT instances come from approximation algorithms, whose ratios are above $3/2$. This work introduces a novel MIP formulation to the problem and modifies the SteinLib test-set to prove that standard solvers are capable of finding solutions to real-world size instances. The remaining of this document is organized as follows: Chapter-2 revises a flow-based MIP formulation for the Steiner Problem in graphs and presents experimental results for a significant test-set of SteinLib's instances; Chapter-3 introduces a novel MIP formulation for the Quality of Service Multicast Tree, which derives from the former flow one and uses a derivative test-set from the previous chapter; while Chapter-4 summarizes the central conclusions and lines of future work.

Chapter 2

An exact model to solve Steiner Tree Problem

In this chapter, we will present briefly the Steiner Tree Problem and then our proposal, a flow-based model; afterward, we will take some steps to keep model's features as much simple as possible aiming to ease solver's computing; consequently reducing execution times. Then, we will show the model's validation, which is based on instances gotten from Steinlib Testdata Library using the solver CPLEX[1] . Finally, we will draw some conclusions from our model's performance.

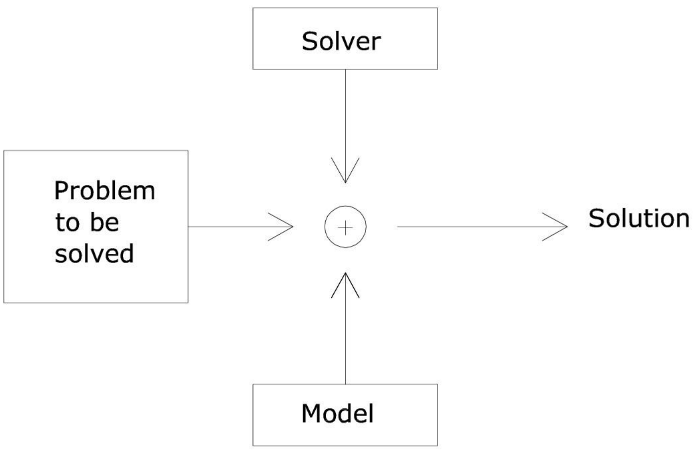

Fig. 2.1 We will use IBM's CPLEX as solver.

[1]Thanks to IBM's Academic Initiative. See https://developer.ibm.com for further information of this tremendous tool.

2.1 Steiner Tree Problem

Formally, the Steiner Tree Problem can be stated as follows: Let G be an undirected graph $G = (V, E, c)$ with a function $c : E \to \mathbb{R}^+$ representing the cost of each edge, and $T \subseteq V$ be a set of nodes, their nodes are referred to as terminals. STP asks for a *tree subgraph* of G of minimum cost $G_T = (V_T, E_T)$, such that $T \subseteq V_T \subseteq V$ and $E_T \subseteq E$ spanning from a given terminal node -called the root R- and all the terminals. In other words, G_T contains a path from each node v to the root R for all $v \in T$, but it may include some of the Steiner nodes.

2.2 Model Description

To get the minimum cost tree subgraph from a given graph we have decided to apply a flow-problem approach. It is based on a basic premise: if there is flow going through an edge, this edge will belong to the solution. As we will see soon, that course of action brings the advantage that connectivity between the terminal nodes and the root is guaranteed. Furthermore, thank for applying Ford-Fulkerson's algorithm some integrality constraints, which would have increased the computational demands, are allowed to be removed. Firstly, before moving forward, we come up with the flow's peculiarities.

Definition: A flow in G is a function $f : VxV \to R$ that satisfies these properties:

- Capacity's constraint: $f(v, w) \leq \mathit{Capacity}\ (v, w), \quad \forall v, w \in G$.
  The flow going through an edge from one vertex to another one must not exceed the edge's capacity.

- Anti symmetric's property: $f(v, w) = -f(w, v), \quad \forall v, w \in G$.
  The flow going through an edge from one vertex to another one is the opposite to the flow going in the opposite direction.

- Flow conservation: $\sum_{v \in V} f(v, w) = 0, \quad \forall u \in V - \{S, R\}$.
  The total amount of flow coming into at each vertex must be equal to the total amount of flow going out of it.

Having presented some of the flow's characteristics, we could use them to describe why connectivity is assured by our model. For the sake of clarity, let graph G be like the one that Figure 2.2 exhibits, where every node behaves according to flow conservation's property, except for the root R. In that case, the graph's root acts as a sink draining as much flow as it can receive.

Now let add a node *S* -called the source- to that graph, where each terminal node is linked up to this source using a unit-capacity edge. As Figure 2.3 displays, neither Steiner nodes nor root *R* accepts any new link, thus they are not connected directly to the source. Finally, let that *S* pushes flow into terminal nodes throughout these new edges.

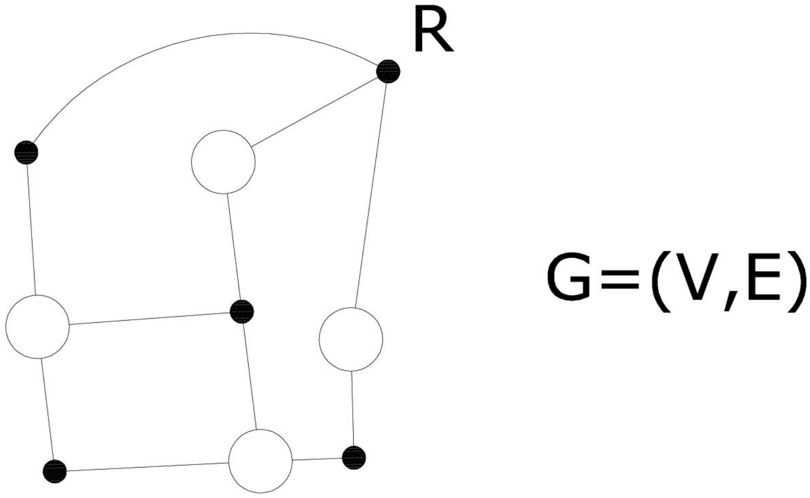

Fig. 2.2 Graph *G*: Black-solid nodes are permanent nodes while white ones are Steiner's

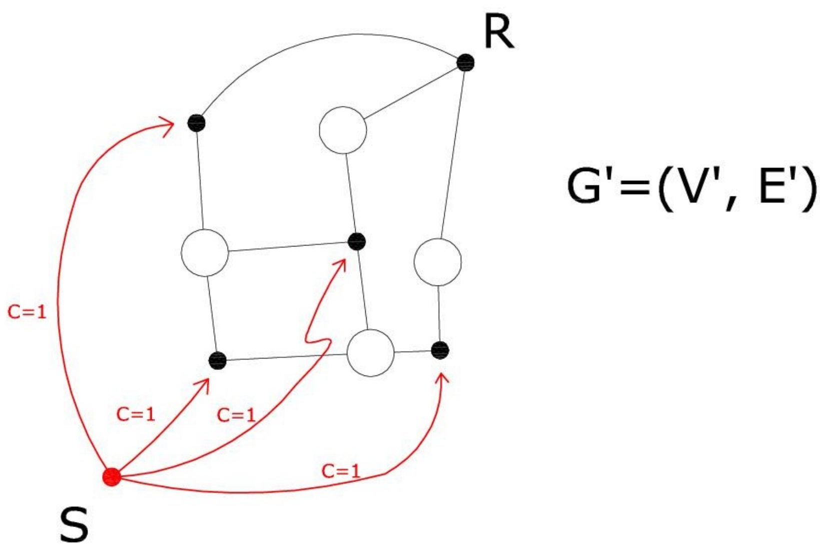

Fig. 2.3 Source *S* injecting flows in each terminal node of the original *G*, but *R*.

Resulting from the conservation's property, nodes are not allowed to hold any flow coming to them. Thus, injected flows must keep flowing from node to node until, eventually, they somehow reach out to a draining node. As a result, each inoculated flow determines a pathway going from each terminal node to the root *R*. However, as we noticed previously, those paths may differ from the optimal one as Figures 2.5 and 2.4 show. As there may be various feasible solutions, optimization is needed to get the optimal one.

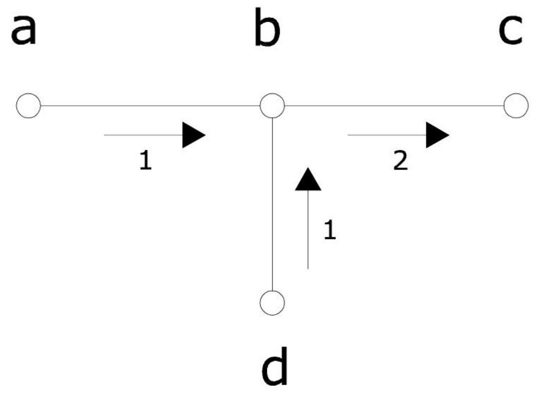

Fig. 2.4 Flow 1 going through a graph

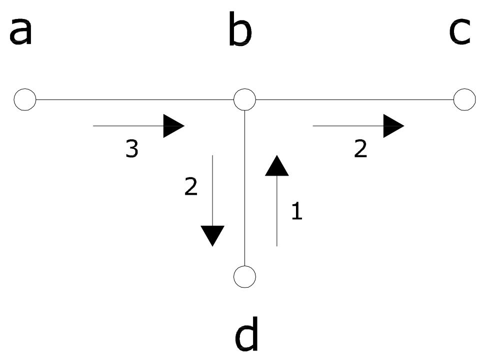

Fig. 2.5 Flow 2 going through a graph

The optimal solution must be a minimum spanning tree. This statement can be proved using *reductio ad absurdum* guessing there is a solution, which is not a minimum spanning tree. Let us suppose there is a solution, which would have at least -without loss of generality- one loop. Subsequently, removing one of these loop's edges would not change terminal nodes' connectivity with the root. Thus, this resulting graph would be also a solution, but with fewer edges than the optimal solution. In such a circumstance, we would obtain a solution, whose cost is lower than the optimal one. That constitutes an absurd. Because of that, minimizing the cost of all the flows injected into the terminal nodes will return a tree spanning from each one of them to the root R.

Model's variables

This subsection is aiming to find a set of variables, which models the cost of flows going through a graph. First of all, let equation 2.1 compute the cost of a solution F.

$$\Phi_{STP}(F) = \sum_{ij \in E_F} c_{ij} \tag{2.1}$$

While c_{ij} was previously defined, E_F is the solution's set of edges. Second, let y_{ij} be a Boolean variable that says whether an edge from the set E belongs to the feasible solution. Then, y_{ij} converts equation 2.1 into the expression 2.2:

$$\Phi_{STP}(F) = \sum_{ij \in E} c_{ij} * y_{ij},\ where\ y_{ij} \in \{0,1\} \tag{2.2}$$

Having already defined y_{ij} as a Boolean variable, there is a couple of issues to be engaged. The first one is related to the variable's integrality. As is known, any mixed-integer programming's model can be resolved by using a *branch-and-bound* algorithm, which may lead to exponential time complexities. Thus, the more integer variables there is the more time it takes to solve the model. Since adding any other integer variable would transform the model into a sluggish one, we resort to another algorithm to overcome that, Ford-Fulkerson. It computes the maximum flow in a network using unitary capacity augmenting paths. Hence, it is straightforward that a graph with integer capacities will have an integer value for the maximum flow going through it. Therefore, emanating from the construction of the edges connected to the source S, see Figure 2.3, it can be said that the maximum flow in our model will be an integer. Taking it for granted, we are allowed to define x_{ij} as a real variable. This variable is expected to toggle y_{ij}: if there is flow passing throughout the edge ij, y_{ij} goes up. Otherwise, it goes down. As a result of lifting integrality restriction for x_{ij}, the search for the optimal solution will be computationally far less challenging.

As we said, the second issue comes from the handling of undirected graphs. This kind of graphs seems to need a couple of variables to determine both the flow's direction and magnitude, as 2.6 shows. Nevertheless, as not keeping the number of variables restrained

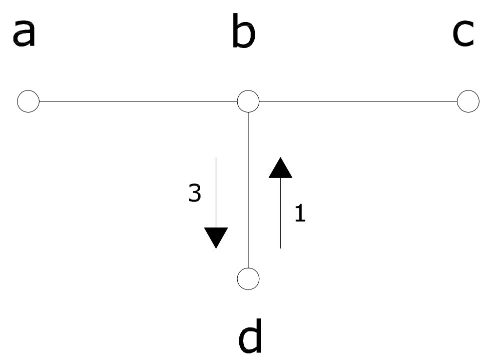

Fig. 2.6 Two flows going through the same edge, but different direction.

may degrade performance, it is desirable if the model could cope with both kinds of graphs, directed or undirected ones, adding just one variable. This issue will be checked by converting any undirected graph into a directed one. Figure 2.7 shows how this transformation takes place, as the graph G from Figure 2.2 becomes in $G' = (V, E')$. It is accomplished by duplicating every link of the graph, but those linked to R and S. In other words, no edge enters the source -which only pushes flow into the terminal nodes- nor leaves the root -which just receives the various flows-. Therefore, in the case of undirected graphs, there is no need to define any other variable to handle flow than x_{ij}. Because of the abovementioned, any mathematical sentence will refer to E' from now on.

In conclusion, our model requires just a couple of non-negative variables per edge:

- A Boolean variable y_{ij}, which registers whether the edge ij belongs to the feasible solution.

- A non-negative real variable x_{ij}, which indicates the maximum flow going through the edge ij.

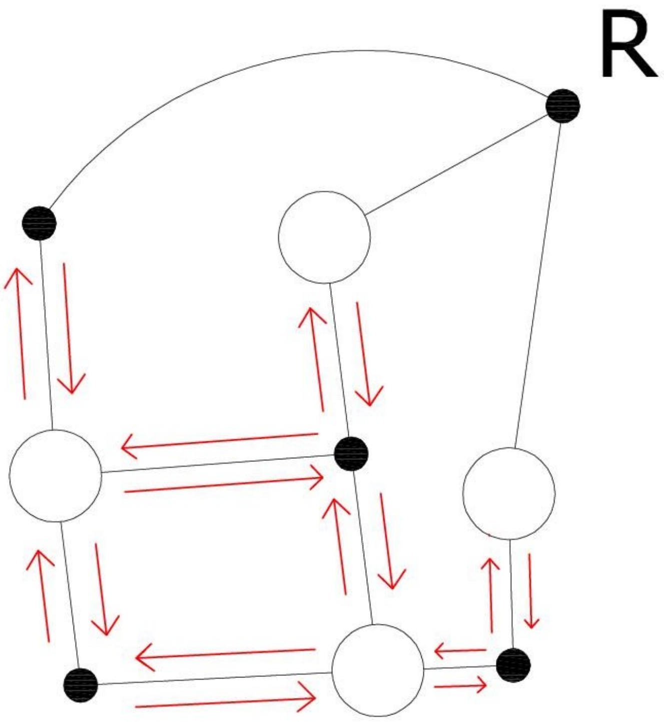

Fig. 2.7 Bidirectional graph G'

Building the model

Having defined the variables, we are ready to see the equations that will build the model.

- First of all, we already have discussed in Subsection 2.2 how to computing the feasible solution's cost, but Equation 2.3 includes the transformed graph E'.

$$\Phi_{STP}(F) = \sum_{ij \in E'} c_{ij} * y_{ij} \tag{2.3}$$

- Second, keeping in mind the construction of the links connected to the source S described in Section 2.2, it is unswerving that the flow through any edge is bounded by the total flow infused by S. The fact that no edge can receive more flow than the total one injected is expressed in Equation 2.4. The total flow could be calculated thanks to the cut showed in Figure 2.8. Note that if there is flow going through an edge, x_{ij} is positive and y_{ij} must go up. In other words, the flow defines whether the edge ij belongs to the feasible solution F.

$$x_{ij} \leq (\mid T \mid -1) * y_{ij}, \quad \forall (ij) \in E' \tag{2.4}$$

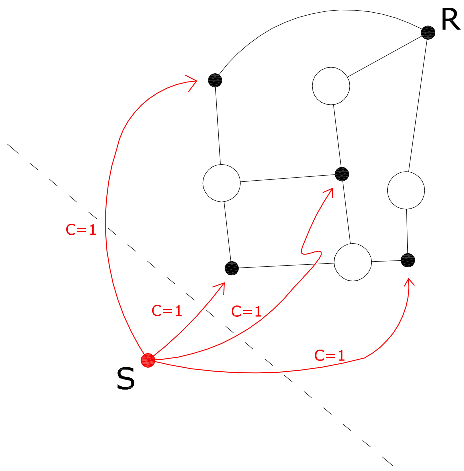

Fig. 2.8 Computing the total flow

- Third, the flow conservation property will help us to get the last two equations required to build our model. One of them says that no Steiner node receives flow from the source S. Another one is derived from the fact that the source S pushes flow into the terminal nodes throughout unitary capacity edges, so the balance constraint must be unitary for all of them, but the root R.

$$\sum_{(ij)\in I+(i)} x_{ij} - \sum_{(ki)\in I-(i)} x_{ki} = 0, \quad \forall i \in (V \backslash T) \tag{2.5}$$

$$\sum_{(ij)\in I+(i)} x_{ij} - \sum_{(ki)\in I-(i)} x_{ki} = 1, \quad \forall i \in T \backslash \{R\} \tag{2.6}$$

Having described the model's expressions, we ask the solver to find $G_T = (V_T, E_T)$ as the solution of the following problem.

Minimize $_{x,y}$ $\Phi_{STP}(F) = \sum_{i,j \in E'} c_{ij} * y_{ij}$
subject to:

- $y_{ij} \in \{0, 1\}, \ \forall (ij) \in E'$
- $x_{ij} \geq 0, \ \forall (ij) \in E'$
- $x_{ij} \leq (\mid T \mid -1) * y_{ij}, \ \forall (ij) \in E'$
- $\sum_{(ij) \in I+(i)} x_{ij} - \sum_{(ki) \in I-(i)} x_{ki} = 0, \ \forall i \in (V \backslash T)$
- $\sum_{(ij) \in I+(i)} x_{ij} - \sum_{(ki) \in I-(i)} x_{ki} = 1, \ \forall i \in T \backslash \{R\}$

2.3 Testing the Model's Performance

Throughout this work, we relied upon *IBM ILOG CPLEX(R) Interactive Optimizer 12.6.3* as the optimization solver. The server was an *HP ProLiant DL385 G7, with 24 AMD Opteron(tm) Processor 6172 with 64GB of RAM*. The results shown in tables 2.1 to 2.7 come from testing instances from Steinlib[2] Testdata Library's Class B and Class I080. In each one of them the first column contains the names of the instance and the entries from left to right are:

- the number of nodes in the graph $|V|$,
- the number of terminal $|T|$,
- the number of undirected edges in the graph $|E|$,
- the optimal value for the instance according to *Steinlib Opt* $(Steinlib)$,
- the gap of the solution with regard to the Steinlib value Gap,
- the number of variables associated with the solution $\#Vars$,
- the number of constraints involved in the solution $\#Constraints$,
- the time in seconds elapsed until the solver finds the solution T_F, and
- the time in seconds elapsed until the solver confirms the solution was found T_C.

Note: Although the solver uses a gap of 0.01% to stop the minimum search, we set 6 hours as a limit on the maximum amount of time dedicated to calculating the output.

[2] http://steinlib.zib.de/steinlib.php

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt (Steinlib)	Gap	# Vars	# Constraints	T_F	T_C
b01	50	9	63	82	0.00%	424	472	0.51s	0.51s
b02	50	13	63	83	0.00%	418	464	0.60s	0.60s
b03	50	25	63	138	0.00%	421	468	0.49s	0.49s
b04	50	9	100	59	0.00%	640	686	7.36s	38.81s
b05	50	13	100	61	0.00%	634	678	0.67s	0.67s
b06	50	25	100	122	0.00%	634	678	20.99s	29.10s
b07	75	13	94	111	0.00%	635	708	0.63s	0.63s
b08	75	19	94	104	0.00%	632	704	0.64s	0.64s
b09	75	38	94	220	0.00%	623	692	0.95s	0.95s
b10	75	13	150	86	0.00%	959	1028	15.67s	43.88s
b11	75	19	150	88	0.00%	965	1036	108.84s	2464.84s
b12	75	38	150	174	0.00%	962	1032	16.74s	72.33s
b13	100	17	125	165	0.00%	840	936	45.49s	69.92s
b14	100	25	125	235	0.00%	843	940	13.57s	63.98s
b15	100	50	125	318	0.00%	837	932	0.99s	0.99s
b16	100	17	200	127	0.00%	792	495	3.25s	3.25s
b17	100	25	200	131	0.00%	792	495	2.47s	2.47s
b18	100	50	200	218	0.01%	1281	1374	76.85s	76.85s

Table 2.1 Steinlib's Testset B from $b01$ to $b18$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt (Steinlib)	Gap	# Vars	# Constraints	T_F	T_C
i080-001	80	6	120	1787	0.00%	778	850	10.56s	66.61s
i080-002	80	6	120	1607	0.00%	787	862	6.82s	6.82s
i080-003	80	6	120	1713	0.00%	775	846	0.61s	6.71s
i080-004	80	6	120	1866	0.00%	781	854	8.04s	22.57s
i080-005	80	6	120	1790	0.00%	790	866	28.40s	48.58s
i080-011	80	6	350	1479	0.00%	1368	763	7.48s	7.48s
i080-012	80	6	350	1484	0.00%	1378	768	10.19s	10.19s
i080-013	80	6	350	1381	0.00%	1386	772	2.63s	2.63s

Table 2.2 Steinlib's Testset I080 from $i080-001$ to $i080-013$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt (Steinlib)	Gap	# Vars	# Constraints	T_F	T_C
i080-014	80	6	350	1397	0.00%	1376	767	2.41s	2.41s
i080-015	80	6	350	1495	0.00%	1386	772	6.72s	6.72s
i080-021	80	6	3160	1175	0.00%	12482	6320	26.22s	26.22s
i080-022	80	6	3160	1178	0.00%	12482	6320	29.63s	29.63s
i080-023	80	6	3160	1174	0.00%	12482	6320	28.87s	28.87s
i080-024	80	6	3160	1161	0.00%	12482	6320	14.91s	14.91s
i080-025	80	6	3160	1162	0.00%	12482	6320	18.07s	18.07s
i080-031	80	6	160	1570	0.00%	620	389	1.18s	1.18s
i080-032	80	6	160	2088	0.00%	630	394	3.97s	3.97s
i080-033	80	6	160	1794	0.00%	620	389	2.13s	2.13s
i080-034	80	6	160	1688	0.00%	626	392	6.58s	6.58s
i080-035	80	6	160	1862	0.00%	632	395	1.74s	1.74s
i080-041	80	6	632	1276	0.00%	2486	1322	7.27s	7.27s
i080-042	80	6	632	1287	0.00%	2478	1318	6.22s	6.22s
i080-043	80	6	632	1295	0.00%	2490	1324	5.45s	5.45s
i080-044	80	6	632	1366	0.00%	2498	1328	7.65s	7.65s
i080-045	80	6	632	1310	0.00%	2494	1326	8.11s	8.11s

Table 2.3 Steinlib's Testset I080 from $i080-001$ to $i080-i045$

i080-101	80	8	120	2608	0.00%	466	312	1.09s	1.09s
i080-102	80	8	120	2403	0.00%	470	314	1.75s	1.75s
i080-103	80	8	120	2603	0.00%	468	313	1.81s	1.81s
i080-104	80	8	120	2486	0.00%	472	315	2.01s	2.01s
i080-105	80	8	120	2203	0.00%	474	316	0.98s	0.98s
i080-111	80	8	350	2051	0.15%	2146	2214	20858.73s	Time Out
i080-112	80	8	350	1885	0.00%	2122	2182	2580.28s	21582.07s
i080-113	80	8	350	1884	0.00%	2125	2186	35.33s	21561.64s
i080-114	80	8	350	1895	0.00%	2146	2214	30.84s	16712.61s
i080-115	80	8	350	1868	0.00%	2149	2218	35.46s	21041.72 s
i080-121	80	8	3160	1561	0.00%	12482	6320	34.83s	34.83s
i080-122	80	8	3160	1561	0.00%	12482	6320	23.17s	27.51s
i080-123	80	8	3160	1569	0.00%	12482	6320	32.19s	51.84s
i080-124	80	8	3160	1555	0.00%	12482	6320	19.85s	24.23s
i080-125	80	8	3160	1572	0.00%	12482	6320	27.48s	57.90s
i080-131	80	8	160	2284	0.00%	620	389	3.08s	3.08s
i080-132	80	8	160	2180	0.00%	622	390	4.20s	4.20s
i080-133	80	8	160	2261	0.00%	626	392	2.84s	2.84s
i080-134	80	8	160	2070	0.00%	628	393	4.82s	4.82s
i080-135	80	8	160	2102	0.00%	622	390	2.30s	2.30s
i080-141	80	8	632	1788	0.00%	2488	1323	11.18s	11.18s
i080-142	80	8	632	1708	0.00%	2488	1323	6.73s	6.73s
i080-143	80	8	632	1767	0.00%	2476	1317	12.48s	18.49s
i080-144	80	8	632	1772	0.00%	2504	1331	8.55s	8.55s
i080-145	80	8	632	1762	0.00%	2494	1326	7.55s	7.55s

Table 2.4 Steinlib's Testset I080 from $i080-101$ to $i080-145$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Consts	T_F	T_C
i080-201	80	16	120	4760	0.00%	470	314	1.69s	1.69s
i080-202	80	16	120	4650	0.00%	464	311	2.73s	2.73s
i080-203	80	16	120	4599	0.00%	466	312	2.44s	2.44s
i080-204	80	16	120	4492	0.00%	470	314	3.26s	3.26s
i080-205	80	16	120	4564	0.00%	470	314	1.88s	1.88s
i080-211	80	16	350	3631	0.00%	1378	768	9.64s	9.64s
i080-212	80	16	350	3677	0.00%	1380	769	17.21s	17.21s
i080-213	80	16	350	3678	0.00%	1366	762	29.30s	29.30s
i080-214	80	16	350	3734	0.00%	1378	768	22.74s	34.87s
i080-215	80	16	350	3681	0.00%	1382	770	19.11s	25.45s
i080-221	80	16	3160	3158	0.00%	12482	6320	52.48s	447.24s
i080-222	80	16	3160	3141	0.00%	12482	6320	179.72s	179.72s
i080-223	80	16	3160	3156	0.00%	12482	6320	610.39s	665.89s
i080-224	80	16	3160	3159	0.00%	12482	6320	356.81s	356.81s
i080-225	80	16	3160	3150	0.00%	12482	6320	60.45s	298.49s
i080-231	80	16	160	4354	0.00%	634	396	5.22s	5.22s
i080-232	80	16	160	4199	0.00%	628	393	2.64s	2.64s
i080-233	80	16	160	4118	0.00%	616	387	4.15s	4.15s
i080-234	80	16	160	4274	0.00%	634	396	3.79s	3.79s
i080-235	80	16	160	4487	0.00%	626	392	5.25s	5.25s
i080-241	80	16	632	3538	0.00%	2490	1324	71.57s	231.33s
i080-242	80	16	632	3458	0.00%	2500	1329	33.85s	36.21s
i080-243	80	16	632	3474	0.00%	2486	1322	35.93s	48.34s
i080-244	80	16	632	3466	0.00%	2498	1328	57.28s	57.28s
i080-245	80	16	632	3467	0.00%	2498	1328	35.96s	39.19s

Table 2.5 Steinlib's Testset I080 from $i080-201$ to $i080-i245$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Consts	T_F	T_C
i080-301	80	20	120	5519	0.00%	466	312	1.39s	1.39s
i080-302	80	20	120	5944	0.00%	470	314	3.29s	3.29s
i080-303	80	20	120	5777	0.00%	466	312	2.18s	2.18s
i080-304	80	20	120	5586	0.00%	466	312	1.74s	1.74s
i080-305	80	20	120	5932	0.00%	472	315	1.86s	1.86s
i080-311	80	20	350	4554	0.00%	1372	765	17.26s	17.26s
i080-312	80	20	350	4534	0.00%	1368	763	15.37s	15.37s
i080-313	80	20	350	4509	0.00%	1374	766	9.41s	12.83s
i080-314	80	20	350	4515	0.00%	1382	770	9.34s	18.75s
i080-315	80	20	350	4459	0.00%	1374	766	7.02s	7.02s
i080-321	80	20	3160	3932	0.00%	12482	6320	139.27s	370.82s
i080-322	80	20	3160	3937	0.00%	12482	6320	247.88s	247.88s
i080-323	80	20	3160	3946	0.00%	12482	6320	247.49s	464.73s
i080-324	80	20	3160	3932	0.00%	12482	6320	66.94s	276.77s
i080-325	80	20	3160	3924	0.00%	12482	6320	110.79s	283.94s

Table 2.6 Steinlib's Testset I080 from $i080-201$ to $i080-325$

Clase	Instancia	$\|V\|$	$\|T\|$	$\|E\|$	Archivo	Óptimo			
Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Consts	T_F	T_C
i080-331	80	20	160	5226	0.00%	620	389	5.22s	5.22s
i080-332	80	20	160	5362	0.00%	626	392	4.86s	4.86s
i080-333	80	20	160	5381	0.00%	616	387	7.84s	7.84s
i080-334	80	20	160	5264	0.00%	628	393	4.43s	4.43s
i080-335	80	20	160	4953	0.00%	632	395	4.51s	4.51s
i080-341	80	20	632	4236	0.00%	2486	1322	13.85s	20.62s
i080-342	80	20	632	4337	0.00%	2486	1322	56.84s	56.84s
i080-343	80	20	632	4246	0.00%	2472	1315	19.79s	48.71s
i080-344	80	20	632	4310	0.00%	2488	1323	22.69s	26.13s
i080-345	80	20	632	4341	0.00%	2482	1320	186.43s	186.43s

Table 2.7 Steinlib's Testset I080 from $i080-301$ to $i080-345$

2.4 Conclusion

Our STP model has been tested with instances from Steinlib's classes B and I080. The solver achieved the accurate value in each of the 120 instances, but one. Even in that case, the gap was just 0.15% due to running out of time. Note that the optimum was founded within a minute in almost 9 of 10 instances, while that value was confirmed within 10 minutes in more than 90% of the cases.

To conclude with the chapter, Table 2.8 shows a summary of computational results. In each one of them the first column contains the names of the instance and the entries from left to right are:

- The number of instances NI
- the range of the selected instances in terms of number of nodes $Nodes$,
- the maximum size of the selected instances in terms of number of edges $Edges$,
- the number of instances where the optimum was not obtained before reaching the threshold time of 6 hours $NOPT$,
- the percentage of instances, whose optimum was obtained within 1 minute by the solver P_F,
- the percentage of instances, whose optimum was verified within 10 minutes by the solver P_F,
- and the average gap *Average*.

Testset	NI	$Nodes$	$Edges$	$NOPT$	P_F	P_C	$Average$
Steinlib's B	18	50-100	Up to 200	0	88.89%	94.44%	0.0006%
Steinlib's I080	100	80	Up to 3160	1	87.00%	94.00%	0.0015%

Table 2.8 Results with STP's model

Chapter 3

A new model for Quality of Service Multicast Tree Problem

In this chapter, we propose a model able to solve accurately the Quality of Service Multicast Tree problem[1] introduced by [Karpinski 2005]. While that work proffered an approximation algorithm, it did not submit any test set. Thus, unlike the Steiner tree problem, there is no available database yet -as far as we know- to perform benchmarking of the QoSMT problem attacked in this work. To get around that limiting factor and draw some conclusions from our model, we decided to produce instances derived from the Steinlib's Library. Following a similar structure to the previous chapter, we will present the QoSMT problem and describe in detail our proposal to cope with it.

3.1 Quality of Service Multicast Tree Problem

Formally, the Quality of Service Multicast Problem can be stated as follows.
Let $G = (V, E, l, r)$ be an undirected graph with two functions, $l : E \to \mathbb{R}^+$ representing the length of each edge, and $r_i : V \to \mathbb{R}^+$ the rate of each node. The QoSMT problem[2] asks for a minimum cost subtree $F = (V_F, E_F)$ of G spanning from a given root R and a set of terminal nodes. Additionally, zero-rate nodes are called Steiner's, and likewise STP, they are not required to be connected to the root. The cost of an edge e in F is: $cost(e) = l(e) * \overline{r_e}$. The second term in the equation, called *rate of edge*, is the maximum rate r_i in the component of $F - \{e\}$ that does not contain R.

[1] It is also known as QoSMT.
[2] See [Karpinski 2005] and [Charikar 2004]

3.2 Model's Description

Given that QoSMT problem is a generalization of the STP, it should not be unexpected that the model developed for the Steiner Tree Problem is the start point of this analysis. A slightly modified version of the flow-based model developed in Chapter 2 will be enough to cope with the more demanding requirements of QoSMT. Fortunately, that course of action brings the same advantages than STP's model enjoys, i.e. guaranteed connectivity between the terminal nodes from scratch, a reduced set of integer variables and the ability to handle with directed and undirected graphs. Due to the reasons mentioned above, Section 2.2 will be followed from here on. It includes that any mathematical sentence will refer to the transformed graph $G = (V', E')$ showed in Figure 3.1.

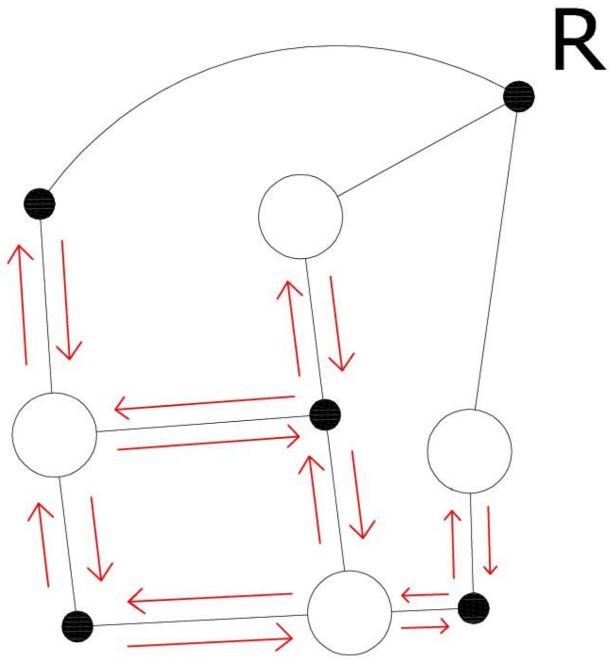

Fig. 3.1 Bidirectional graph G'

Concerning the connectivity, we resort over to the remarkable peculiarity of any flow-based model: if there is flow going into an edge, this edge will belong to the solution. To illustrate the approach already used in the previous chapter, let S be an outward node to the graph G that infuses flows to the terminal nodes.So, each flow continues flowing from the initial terminal node to their neighbors' nodes until, anyhow, it reaches out to the draining node, the root R. Furthermore, based on the deductive reasoning described in Subsection 2.2, the optimal solution for the QoSMT problem will be a tree.

Model's variables

As the key difference between the Steiner Problem and QoSMT is the node's rate feature, we will consider a couple of items to establish the variables to be used. The first one is the survey of the variables used in Chapter 2 to be recovered for QoSMT's model. The second one implies the analysis of how many others are wanted. As we will see, our proposal requires just one extra variable. To begin with, given the definition of the cost of an edge presented in section 3.1, the solution's cost will be:

$$\Phi_{QoSMT}(F) = \sum_{e \in E_F} l_e * \overline{r_e} \tag{3.1}$$

As any flow-based model entails two variables per edge, let x_{uv} and y_{uv} be them. The first one, x_{uv}, is associated with the crossing of flow into an edge, while y_{uv} is a Boolean variable indicating whether an edge from E' belongs to the solution. Finally, a variable z_v per node is compelled to deal with the rate of edge. There is no compulsion to force x_{uv} nor z_v to be an integer. Consequently, they change the cost expression into Equation 3.2.

$$\Phi_{QoSMT}(F) = \sum_{uv \in E'} y_{uv} * l_{uv} * z_v \tag{3.2}$$

Note that nonlinearity is inherent to the $QoSMT$. Therefore, an auxiliary variable ηuv to linearize $\Phi_{QoSMT}(F)$ is wanted. In conclusion, our model requires:

- An integer variable y_e, which indicates whether the edge e belongs to the solution.
- A non-negative real variable x_e, which registers the maximum flow going through the edge e.
- A non-negative real variable z_v, which indicates the *rate of edge* of the node v.
- A non-negative real variable η_e, which combines the variables y_e and z_v.

Building the model

Given the defined variables in Subsection 3.2, we are placing those expressions to establish up the model. As minimization is one of the two paramount features of this work, expressions are represented as a system of inequalities.

- First of all, the expression to be minimized is the solution's cost given in Equation 3.2.
- Second, expression 3.3 states that no edge receives more flow than the total injected in each terminal node. Moreover, it shows how x_{uv} commands y_{uv}. Note that if there is flow going through an edge, x_{uv} non-negative and y_{uv} must go up.

$$x_{uv} \leq (\mid T \mid -1) * y_{uv}, \quad \forall (uv) \in E' \tag{3.3}$$

- Third, likewise Steiner Tree problem, two equations are inferred from the flow's conservation property. The first one indicates that no Steiner node receives flow from the source S. Accordingly, the balance constraint is naught for all of them. The other equation asserts that the source pushes flow into the terminal nodes throughout unitary capacity edges. Then, the balance constraint is unitary for each of them, except the root. See Equations 3.4 and 3.5, respectively.

$$\sum_{(ut)\in I+(u)} x_{ut} - \sum_{(wu)\in I-(u)} x_{wu} = 0, \quad \forall u \in (V \backslash T) \tag{3.4}$$

$$\sum_{(ut)\in I+(u)} x_{ut} - \sum_{(wu)\in I-(u)} x_{wu} = 1, \qquad \forall u \in T \backslash \{R\} \tag{3.5}$$

- Fourth, it is time to tackle the *rate of edge*, which is the trademark of this problem. In the opening, each node assigns its r_v to z_v, which controls the node rate. According to the definition given in Section 3.1, each solution's non-leaf node v receives the maximum downstream rate. If that value is higher than its original value, z_v takes it. Figures 3.2 shows the downstream's nodes affected in the assignment of z_v.

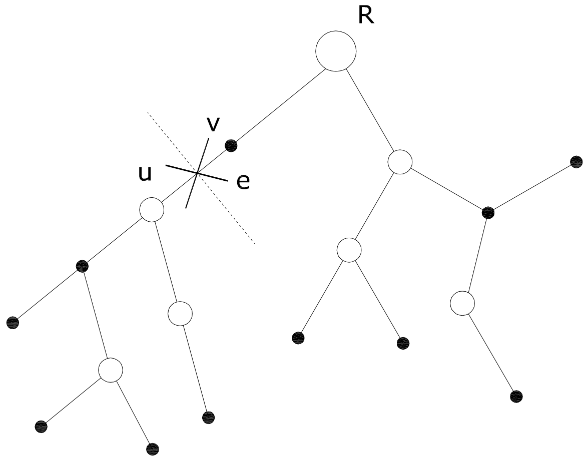

Fig. 3.2 Definition of $\overline{r_e}$.

The quest for which edge belongs to the solution remains. Therefore, we resort to y_{uv} to discern them as expression 3.7 shows, where $C = max \ \ \{r_v : v \in T \backslash \{R\}\}$.

$$z_v \geq \ \ r_v, \quad \forall v \in V \backslash \{R\} \tag{3.6}$$

$$z_v \geq z_u + C * (y_{uv} - 1), \quad \forall v \in V \backslash \{R\}, \quad (uv) \in E' \tag{3.7}$$

To analyze it, let v be a non-leaf node, while u_0, u_1, and u_2 are all its neighbors. As Figure 3.3 shows, all of them belong to the solution, but u_1. As we will show hereafter, u_0 and u_2 may affect the value of z_v, while u_1 might not.

– On one hand, let expression 3.7 be evaluated at those nodes that belong to the solution. When it is assessed at u_0 ($y_0 = 0$) and u_2 ($y_2 = 0$), it becomes the expressions 3.8 and 3.9, respectively.

$$z_v \geq z_{u_0} \tag{3.8}$$

$$z_v \geq z_{u_2} \tag{3.9}$$

Consequently, expression 3.7 drives z_v to choose the higher value from downstream's node belonging to the solution.

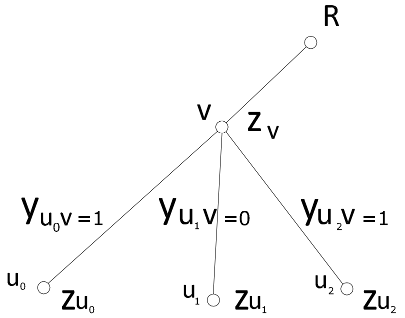

Fig. 3.3 Node rate's assignment analysis.

– On the other hand, if the expression 3.7 is evaluated at a node that does not belong to the solution, as u_1, it transforms firstly into $z_v \geq z_u + C * \ (-1)$. As C is higher than any value of z_u, the right side of that inequality is non-positive. Given the variables' definition seen in Subsection 3.2, eventually,the evaluation becomes into expression 3.10. Hence, it is noteworthy that inequality 3.7 secures that no outside node contributes to z_v.

$$z_v \geq 0, \quad \forall v \in V \backslash \{R\} \tag{3.10}$$

In summation, expression 3.6 imposes a fundamental requirement for z_v. Also, the inequality 3.7 gets the maximum of downstream's nodes connected to the node v, while ignores the contribution of other edges that do not belong to the solution.

For illustrative purposes, let the tree shown in Figure 3.4 be a solution, where solid black dots represent terminal nodes with its associated rate. Note that each non-leaf terminal node receives the maximum from the solution's downstream.

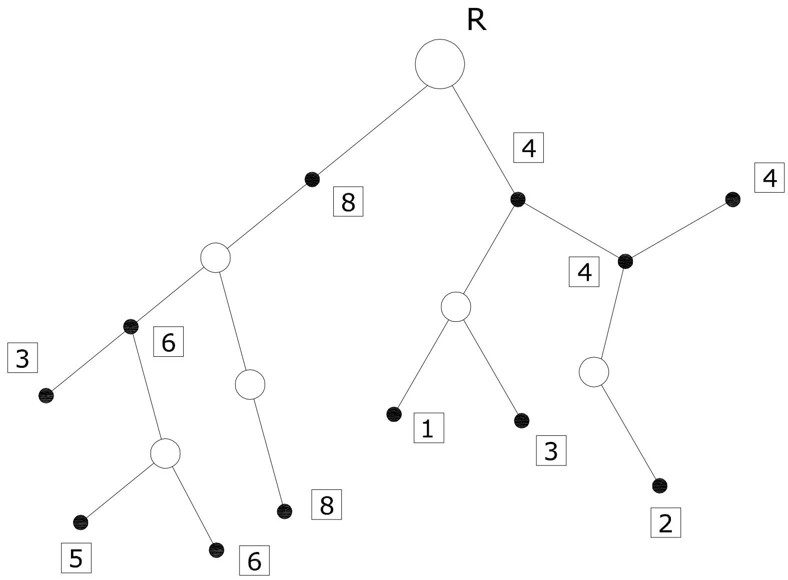

Fig. 3.4 Rate's assignment.

Linealization of the model

Given that the solution cost obtained in Equation 3.2 is non-linear, an assisting variable is wanted to linearize the equation $\Phi_{QoSMT}(F) = \sum_{uv \in E'} y_{uv} * l_{uv} * z_v$. Thanks to variable η, that equation is transformed into the following linear expression.

$$\Phi_{QoSMT}(F) = \sum_{uv \in E'} l_{uv} * \eta_{uv} \tag{3.11}$$

Note that several of the previous equations and inequalities may be required to keep the consistency of the problem.

Having described the model's system of inequalities, we ask the solver to find $G_T = (V_T, E_T)$ as solution of the following problem:

$Minimize_{\ x,y,z,\eta}\ \Phi_{QoSMT}(F) = \sum_{uv \in E'} l_{uv} * \eta_{uv}$
subject to:

- $x_{uv} \geq 0,\ \ \forall (uv) \in E'$
- $y_{uv} \in \{0,1\},\ \ \forall (uv) \in E'$
- $z_v \geq\ \ 0,\ \ \forall v \in V \backslash \{R\}$
- $z_v \geq\ \ r_v,\ \ \forall v \in V \backslash \{R\}$
- $\eta_{uv} \geq\ \ 0,\ \ \forall (uv) \in E'$
- $(\mid T \mid -1) * y_{uv} \geq x_{uv},\ \ \forall (uv) \in E'$
- $\sum_{(ut) \in I+(u)} x_{ut} - \sum_{(wu) \in I-(u)} x_{wu} = 1,\ \ \forall u \in T \backslash \{R\}$
- $\sum_{(ut) \in I+(u)} x_{ut} - \sum_{(wu) \in I-(u)} x_{wu} = 0,\ \ \forall u \in (V \backslash T)$
- $z_v \geq\ \ z_u + C * (y_{uv} - 1),\ \ \forall v \in V \backslash \{R\},\ \ (uv) \in E'$
- $\eta_{uv} \geq\ \ z_u + C * (y_{uv} - 1),\ \ \forall u \in V \backslash \{R\},\ \ (uv) \in E'$

3.3 Testing the Model's Performance

As we pointed out previously, throughout this work, we relied upon *IBM ILOG CPLEX(R) Interactive Optimizer 12.6.3* as the optimization solver. The server was an *HP ProLiant DL385 G7, with 24 AMD Opteron(tm) Processor 6172 with 64GB of RAM*. Regarding the instances, to the best of our knowledge, there is no test data available to validate this model. The primary cause is that there have not been studies taking on Quality of Service Multicast Tree Problem recently, but [Karpinski 2005]. Even they did not generate any database at all. Trying to face this stricture and draw some conclusions, we built a database. The test data was constructed by taking the instances from Steinlib Testdata Library's Class B and Class I080 and using random values between 1 and 100 for the nodes' rate. The results are shown in tables 3.1 to 3.5 come from testing the abovementioned modified instances [3]. In each one of them, the first column contains the names of the instance and the entries from left to right are:

- the number of nodes in the graph $|V|$,
- the number of terminal $|T|$,
- the number of undirected edges in the graph $|E|$,
- the optimal value for the instance according to *Steinlib Opt* $(Steinlib)$,
- the gap of the solution with regard to the Steinlib value *Gap*,
- the number of variables associated with the solution #*Vars*,
- the number of constraints involved in the solution #*Constraints*,
- the time in seconds elapsed until the solver finds the solution T_F, and
- the time in seconds elapsed until the solver confirms the solution was found T_C.

[3] The reader can access them in the URL $www.fing.edu.uy/\sim frobledo/QoSMT\_Modified\_Instances$.

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Constraints	T_F	T_C
modified b01	50	9	63	6080	0.01%	424	472	0.59s	0.59s
modified b02	50	13	63	5770	0.01%	418	464	0.50s	0.50s
modified b03	50	25	63	7925	0.01%	421	468	0.71s	0.71s
modified b04	50	9	100	3883	0.01%	640	686	13.39s	145.86s
modified b05	50	13	100	3146	0.01%	634	678	10.06s	41.84s
modified b06	50	25	100	7820	5.96%	634	678	9476.51s	Time Out
modified b07	75	13	94	7616	0.01%	635	708	10.11s	10.11s
modified b08	75	19	94	7364	0.01%	632	704	11.12s	29.35s
modified b09	75	38	94	15877	0.01%	623	692	18.08s	18.08s
modified b10	75	13	150	5096	0.01%	959	1028	15.67s	3667.31s
modified b11	75	19	150	6718	11.78%	965	1036	4600.61s	Time Out
modified b12	75	38	150	10716	0.01%	962	1032	1988.57s	3276.68s
modified b13	100	17	125	12076	0.01%	840	936	32.19s	797.12s
modified b14	100	25	125	15159	3.61%	843	940	18.54s	Time Out
modified b15	100	50	125	20599	0.01%	837	932	33.56s	118.26s
modified b16	100	17	200	5288	11.12%	1287	1382	56.34s	Time Out
modified b17	100	25	200	6807	15.67%	1287	1382	19499.50s	Time Out
modified b18	100	50	200	9884	0.01%	1296	1394	19513.19s	21588.79s

Table 3.1 Modified Steinlib's Testset B from $b01$ to $b18$

Note: Although the solver uses a gap of 0.01% to stop the minimum search, we set 6 hours as a limit on the maximum amount of time dedicated to calculating the output.

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Constraints	T_F	T_C
modified i080-001	80	6	120	114943	0.01%	778	850	17.67s	24.72s
modified i080-002	80	6	120	103965	0.01%	787	862	6.83s	15.24s
modified i080-003	80	6	120	89109	0.01%	775	846	8.00s	8.00s
modified i080-004	80	6	120	68624	0.01%	781	854	69.44s	69.44s
modified i080-005	80	6	120	105221	0.01%	790	866	39.47s	599.39ss
modified i080-011	80	6	350	99490	6.49%	2131	2194	19757.94s	Time Out
modified i080-012	80	6	350	103884	25.35%	2146	2214	7.35s	Time Out
modified i080-013	80	6	350	84759	25.73%	2158	2230	13662.17s	Time Out
modified i080-014	80	6	350	94408	23.87%	2143	2210	28.73s	Time Out
modified i080-015	80	6	350	65988	27.51%	2158	2230	19835.44s	Time Out
modified i080-021	80	6	3160	64730	39.36%	18802	18802	19501.97s	Time Out
modified i080-022	80	6	3160	63278	25.35%	18802	18802	452.78s	Time Out
modified i080-023	80	6	3160	88478	22.03%	18802	18802	382.26s	Time Out
modified i080-024	80	6	3160	53022	20.52%	18802	18802	661.30s	Time Out
modified i080-025	80	6	3160	43997	59.04%	18802	18802	992.75s	Time Out
modified i080-031	80	6	160	60478	0.01%	1009	1078	19.28s	7007.92s
modified i080-032	80	6	160	96560	2.49%	1024	1078	6.74s	Time Out
modified i080-033	80	6	160	111186	4.41%	1009	1078	2449.32s	Time Out
modified i080-034	80	6	160	83697	0.01%	1018	1090	502.78s	3573.91s
modified i080-035	80	6	160	91836	1.06%	1027	1102	7.16s	Time Out
modified i080-041	80	6	632	65550	17.96%	3808	3866	42.59s	Time Out
modified i080-042	80	6	632	109276	15.66%	3796	3850	17.65s	Time Out
modified i080-043	80	6	632	87213	16.01%	3814	3874	1169.27s	Time Out
modified i080-044	80	6	632	77182	24.07%	3826	3890	20767.52s	Time Out
modified i080-045	80	6	632	72863	21.58%	3820	3882	47.86s	Time Out

Table 3.2 Modified Steinlib's Testset I080 from $i080-001$ to $i080-i045$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Constraints	T_F	T_C
modified i080-101	80	8	120	100159	0.01%	778	850	10.31s	183.15s
modified i080-102	80	8	120	152735	0.01%	784	858	93.77s	2098.45s
modified i080-103	80	8	120	172107	0.01%	781	854	19.24ss	718.79s
modified i080-104	80	8	120	138697	0.01%	787	862	10.32s	359.61s
modified i080-105	80	8	120	117490	0.01%	790	866	7.65s	10.71s
modified i080-111	80	8	350	137090	14.03%	2146	2214	33.90s	Time Out
modified i080-112	80	8	350	129134	17.06%	2122	2182	5174.81s	Time Out
modified i080-113	80	8	350	108170	18.90%	2125	2186	19812.33s	Time Out
modified i080-114	80	8	350	108003	3.93%	2146	2214	300.94s	Time Out
modified i080-115	80	8	350	106514	3.77%	2149	2218	30.42s	Time Out
modified i080-121	80	8	3160	76236	20.85%	18802	18802	5171.87s	Time Out
modified i080-122	80	8	3160	117965	16.46%	18802	18802	19744.10s	Time Out
modified i080-123	80	8	3160	100750	35.05%	18802	18802	19613.44s	Time Out
modified i080-124	80	8	3160	120469	24.03%	18802	18802	0.44s	Time Outs
modified i080-125	80	8	3160	97394	21.77%	18802	18802	19472.73s	Time Out
modified i080-131	80	8	160	129510	0.01%	1009	1078	20.98s	19782.90s
modified i080-132	80	8	160	119161	4.66%	1012	1082	36.70s	Time Out
modified i080-133	80	8	160	159670	0.11%	1018	1090	648.97s	Time Out
modified i080-134	80	8	160	126998	0.01%	1021	1094	117.95s	906.16s
modified i080-135	80	8	160	119391	0.01%	1012	1082	2673.88s	14050.81s
modified i080-141	80	8	632	79623	26.88%	3811	3870	5582.23s	Time Out
modified i080-142	80	8	632	126017	19.75%	3811	3870	63.72s	Time Out
modified i080-143	80	8	632	87131	20.22%	3793	3846	58.63s	Time Out
modified i080-144	80	8	632	91120	23.50%	3835	3902	79.99s	Time Out
modified i080-145	80	8	632	66784	27.20%	3820	3882	53.30	Time Out

Table 3.3 Modified Steinlib's Testset I080 from $i080-101$ to $i080-i145$

Instance	$\|V\|$	$\|T\|$	$\|E\|$	Opt	Gap	# Vars	# Constraints	T_F	T_C
modified i080-201	80	16	120	297680	0.01%	784	858	16.24s	1987.05s
modified i080-202	80	16	120	283062	0.01%	775	846	14.96s	32.16s
modified i080-203	80	16	120	277641	0.01%	778	850	308.64s	2086.20s
modified i080-204	80	16	120	303053	0.01%	784	858	22.86s	2300.16s
modified i080-205	80	16	120	256616	0.01%	784	858	15.22s	1614.07s
modified i080-211	80	16	350	211496	18.99%	2146	2214	19529.33s	Time Out
modified i080-212	80	16	350	238695	13.68%	2149	2218	20741.75s	Time Out
modified i080-213	80	16	350	226435	17.34%	2128	2190	758.77s	Time Out
modified i080-214	80	16	350	205284	14.36%	2146	2214	19584.22s	Time Out
modified i080-215	80	16	350	164053	12.89%	2152	2222	123.40s	Time Out
modified i080-221	80	16	3160	123895	17.21%	18802	18802	19576.72s	Time Out
modified i080-222	80	16	3160	140113	17.40%	18802	18802	19526.19s	Time Out
modified i080-223	80	16	3160	169861	17.12%	18802	18802	21109.41s	Time Out
modified i080-224	80	16	3160	197226	13.79%	18802	18802	19494.15s	Time Out
modified i080-225	80	16	3160	139070	15.74%	18802	18802	19651.13s	Time Out
modified i080-231	80	16	160	252533	7.34%	1030	1106	42.61s	Time Out
modified i080-232	80	16	160	236170	9.31%	1021	1094	1249.45s	Time Out
modified i080-233	80	16	160	230929	6.51%	1003	1070	811.98s	Time Out
modified i080-234	80	16	160	207233	12.77%	1030	1106	360.77s	Time Out
modified i080-235	80	16	160	304894	12.20%	1018	1090	13182.11s	Time Out
modified i080-241	80	16	632	190376	21.22%	3814	3874	2773.10s	Time Out
modified i080-242	80	16	632	205946	17.65%	3829	3894	108.54s	Time Out
modified i080-243	80	16	632	197878	20.09%	3808	3866	19905.46s	Time Out
modified i080-244	80	16	632	220403	15.99%	3826	3890	120.77s	Time Out
modified i080-245	80	16	632	205865	20.21%	3826	3890	20474.30s	Time Out

Table 3.4 Modified Steinlib's Testset I080 from $i080-201$ to $i080-i245$

Instance	$\lvert V\rvert$	$\lvert T\rvert$	$\lvert E\rvert$	Opt	Gap	# Vars	# Constraints	T_F	T_C
modified i080-301	80	20	120	321694	0.01%	778	850	38.86s	498.53s
modified i080-302	80	20	120	399632	0.69%	784	858	760.97s	Time Out
modified i080-303	80	20	120	393728	1.31%	778	850	35.23s	Time Out
modified i080-304	80	20	120	319655	0.01%	778	850	33.73s	103.54s
modified i080-305	80	20	120	349572	3.37%	787	862	38.67s	Time Out
modified i080-311	80	20	350	249873	12.73%	2137	2202	107.14s	Time Out
modified i080-312	80	20	350	275934	14.43%	2131	2194	19939.97s	Time Out
modified i080-313	80	20	350	314705	13.50%	2140	2206	45.46s	Time Out
modified i080-314	80	20	350	227138	13.85%	2152	2222	15318.83s	Time Out
modified i080-315	80	20	350	274733	10.30%	2140	2206	19559.47s	Time Out
modified i080-321	80	20	3160	172325	13.71%	18802	18802	20111.31s	Time Out
modified i080-322	80	20	3160	223253	12.20%	18802	18802	6878.48s	Time Out
modified i080-323	80	20	3160	227635	11.20%	18802	18802	2958.93s	Time Out
modified i080-324	80	20	3160	204185	13.70%	18802	18802	1433.57s	Time Out
modified i080-325	80	20	3160	236926	12.01%	18802	18802	2935.38s	Time Out
modified i080-331	80	20	160	285610	7.14%	1009	1078	19468.31s	Time Out
modified i080-332	80	20	160	316045	3.47%	1018	1090	3942.14s	Time Out
modified i080-333	80	20	160	329069	7.40%	1003	1070	23.93s	Time Out
modified i080-334	80	20	160	292621	7.36%	1021	1094	26.51s	Time Out
modified i080-335	80	20	160	286319	6.16%	1027	1102	32.42s	Time Out
modified i080-341	80	20	632	210911	14.64%	3808	3866	3343.92s	Time Out
modified i080-342	80	20	632	261191	16.98%	3808	3866	179.68s	Time Out
modified i080-343	80	20	632	229186	16.36%	3787	3838	2849.52s	Time Out
modified i080-344	80	20	632	202966	18.40%	3811	3870	6550.25s	Time Out
modified i080-345	80	20	632	286760	16.91%	3802	3858	6513.35s	Time Out

Table 3.5 Modified Steinlib's Testset I080 from $i080-301$ to $i080-i345$

3.4 Conclusion

The QoSMT's model has been tested with a couple of sets of instances, which were modifications of the Steinlib's classes B and I080. As there is no database, we had to rely on the solver and its tools to validate the value of the optimum and the eventual gap of each instance. We consider this work the first step in the task of building a comprehensive test data for the QoSMT problem. Given that it is more demanding computationally than STP, it is unexpected that some instances reached the strict threshold time we set without getting the optimum. To conclude with the chapter, Table 3.6 shows a summary of computational results. In each one of them the first column contains the names of the instance and the entries from left to right are:

- The number of instances NI
- the range of the selected instances in terms of number of nodes $Nodes$,
- the maximum size of the selected instances in terms of number of edges $Edges$,
- the number of instances where the optimum was not obtained before reaching the threshold time of 6 hours $NOPT$,
- the percentage of instances, whose optimum was obtained within 1 minute by the solver P_F,
- the percentage of instances, whose optimum was verified within 10 minutes by the solver P_F,
- and the average gap $Average$.

Testset	NI	Nodes	Edges	NOPT	P_F	P_C	Average
Steinlib's B (Modified)	18	50-100	Up to 200	5	83.33%	50.00%	2.68%
Steinlib's I080 (Modified)	100	80	Up to 3160	88	38.00%	11.00%	12,65%

Table 3.6 Results with QoSMT's model

Chapter 4

Conclusions

This work benchmarks the performance of a classic flow-based mixed-integer problem formulation (MIP) for the STP using the SteinLib, a reference test-set repository. That MIP formulation is modified to solve the Quality of Service Multicast Tree problem (QoSTP). As for whom there is no MIP formulation previous to that presented here, existing approaches go the way of approximation algorithms to find solutions. Experimental results with this novel MIP formulation and standard optimization tools show very promising results when used with the same test-set. Optimal solutions are found for many instances, while very low feasible-to-optimal gaps were found for most of the remaining. Paraphrasing the saying *A picture is worth a thousand words*, let Figures 4.1 and 4.2 show the solutions to the instance *b*01 and *b*02 instances from the SteinLib's class B. Finally, let Figure 4.3 despict the solution to the QoSMT applied to the modified *b*01. Compare it with the *b*01 solution and note how the weights[1] affected both trees.

[1] The numbers inside the circles represent the weight of each node, while those in thick squares the terminal node and the thin one, the Steiner nodes.

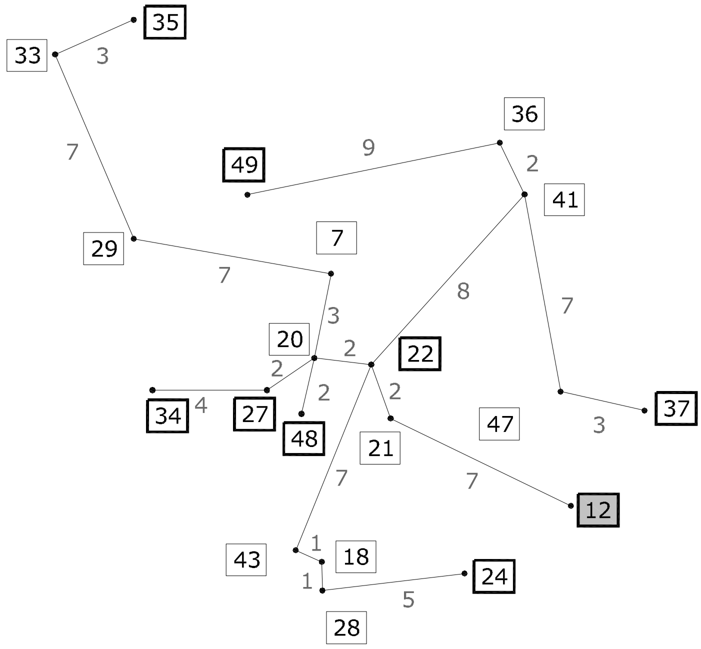

Fig. 4.1 Optimal tree for instance Steinlib's b01

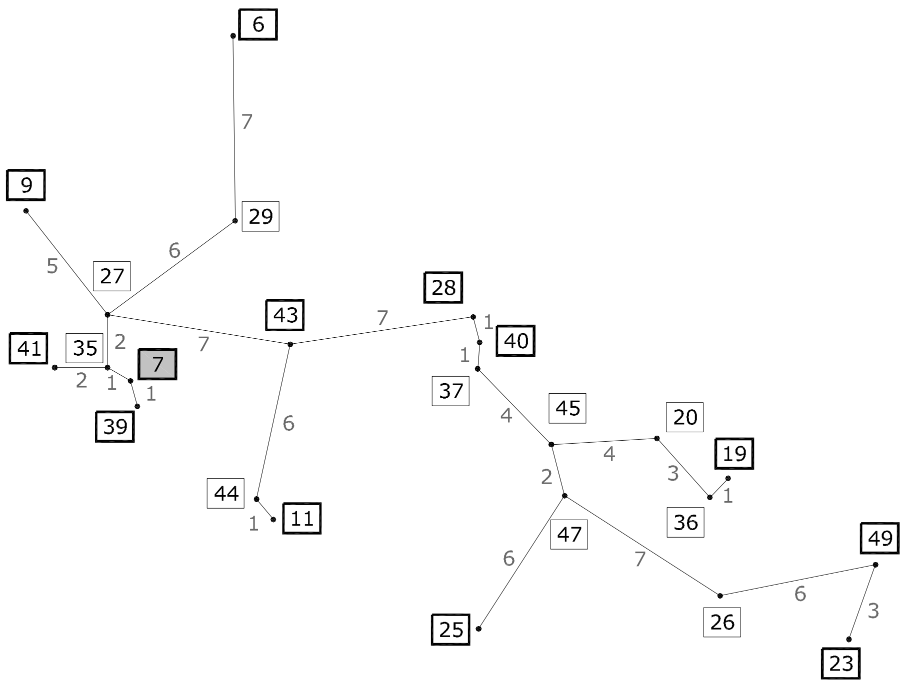

Fig. 4.2 Optimal tree for instance Steinlib's b02

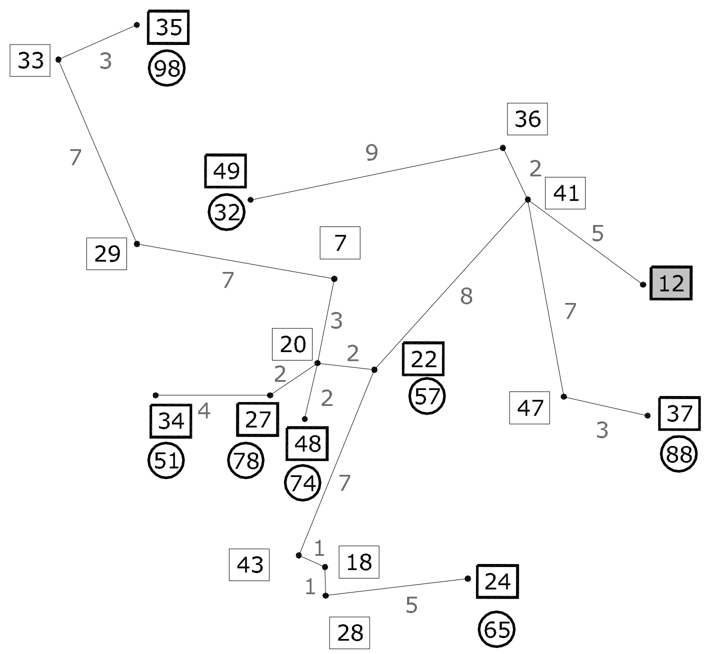

Fig. 4.3 Optimal tree for the modified instance Steinlib's b01

References

[Aneja 1980] Yash Aneja.
An Integer Programming Approach to the Steiner Problem in Graphs.
Networks, 10:167–178, 1980.

[Baïou 1996] Mourad Baïou.
Le Problème du Sus-graphe Steiner 2-arête Connexe: Approache Polyédrale.
PhD thesis, Université de Rennes 1, 1996.

[Beasley 1984] John Beasley.
An Algorithm for the Steiner Problem in Graphs.
Networks, 14:147 – 159., 1984.

[Beasley 1989] John Beasley.
A SST-based Algorithm for the Steiner Problem in Graphs.
Networks, 19:1 – 16, 1989.

[Bienstock 1990] Daniel Bienstock, Ernest F. Brickell and Clyde L. Monma.
On the Structure of Minimum-weight K-connected Spanning Networks.
SIAM J. Discret. Math., vol. 3, no. 3, pages 320–329, May 1990.

[Borradaile 2009] Borradaile, G., Klein, P., Mathieu, C., 2009.
An O(n log n) Approximation Scheme for Steiner Tree in Planar Graphs.
ACM Trans. Algorithms 5, 3, 31:1–31:31.

[Charikar 2004] Charikar, M., Naor, J.S., Schieber, B.
Resource Optimization in QoS Multicast Routing of Real-time Multimedia.
Journal IEEE/ACM Transactions on Networking (TON) archive Volume 12 Issue 2, April 2004 Pages 340-348.

[Chopra 1992] Sunil Chopra, Edgar Gorres,M.R. Rao.
Solving a Steiner Tree Problem on a Graph using Branch and Cut.
ORSA Journal on Computing, 4:320 – 335, 1992.

[Diané 2006] Mamadi Diané, Ján Plesník.
An Integer Programming Formulation of the Steiner Problem in Graphs.
ZOR - Methods and Models of Operations Research 37:107-111.

[Dreyfus 1971] S. Dreyfus, R. Wagner.
The Steiner Problem in Graphs.
Networks, 1:195 – 207, 1980.

[Duin 1994] Duin, C., Voß, S., 1994.
Steiner Tree Heuristics: A Survey.
In *Operations Research Proceedings 1993*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 485–496.

[Garey 1977] Michael Garey, David Johnson.
The Rectilinear Steiner Tree Problem is NP-Complete.
SIAM Journal on Applied Mathematics, 32:826 – 834, 1977.

[Goemans 1993] Goemans, M.X., Myung, Y.S., 1993.
A Catalog of Steiner Tree Formulations.
Networks 23, 1, 19–28. https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230230104.

[Hauptmann 2013] Hauptmann, M., Karpinski, M., 2013.
A Compendium on Steiner Tree Problems.
Technical report, Department of Computer Science and Hausdorff Center for Mathematics University of Bonn.

[Hua 2018] Hua, Hao. (2018).
A Flow Formulation for Steiner Tree Problem.
Preprint. DOI: 10.13140/RG.2.2.21163.75047.

[Hwang 1992] Frank Hwang, Dana Richards.
Steiner Tree Problems.
Networks, 22:55 – 89, 1992.

[Hwang 1992] Frank Hwang, Dana Richards, Pawel Winter.
The Steiner Tree Problem.
Annals of Discrete Mathematics 53, North-Holland, Amsterdam, 1992.

[Karp 1972] Richard Karp.
Reducibility among Combinatorial Problems.

In Miller, R. E. and Thatcher, J. W., editors, Complexity of Computer Computations, pages 85 – 103. Plenum Press, New York, 1972.

[Karpinski 2005] Karpinski, M., Mandoiu, I., Olshevsky, A. et al. *Improved Approximation Algorithms for the Quality of Service Multicast Tree Problem.*
Algorithmica (2005) 42: 109. https://doi.org/10.1007/s00453-004-1133-y

[Koch 2001] Koch T., Martin A., Vos S., 2001.
An Updated Library on Steiner Tree Problems in Graphs.
Combinatorial Optimization book series 11. https://doi.org/10.1007/978-1-4613-0255-1_9

[Kruskal 1956] Kruskal, Joseph B.
"On the shortest spanning subtree of a graph and the traveling salesman problem." Proceedings of the American Mathematical society 7, no. 1 (1956): 48-50.

[Lawler 1976] Edward Lawler.
Combinatorial Optimization: Networks and Matroids.
Holt, Rinehart and Winston, New York, 1976.

[Leggieri 2014] Valeria Leggieri, Mohamed Haouari, and Chefi Trikib.
The Steiner Tree Problem with Delays: A compact formulation and reduction procedures.
Discrete Applied Mathematics, Vol. 164, No. 1, pages 178-190, February 2014.

[Lucena 1993] Abilio Lucena.
Tight Bounds for the Steiner Problem in Graphs.
Preprint, IRC for Process Systems Engineering, Imperial College, London, 1993.

[Maculan 1987] Nelson Maculan.
The Steiner Problem in Graphs.
Annals of Discrete Mathematics, 31:185 – 212, 1987.

[Martins 2000] *S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A parallel GRASP for the Steinertree problem in graphs using a hybrid local search strategy.*
Journal of Global Optimization, 17:267–283, 2000.

[Minoux 1990] M. Minoux.
Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity.
INFOR, 28:221–233, 1990.

[Monma 1990] ClydeL. Monma, BethSpellman Munson and WilliamR Pulleyblank.
Minimum-weight Two-connected Spanning Networks.
Mathematical Programming, vol. 46, no. 1-3, pages 153–171, 1990.

[Prim 1957] Prim, Robert Clay.
Shortest connection networks and some generalizations.
The Bell System Technical Journal 36.6 (1957): 1389-1401.

[Polzin 2003] Polzin, T., 2003.
Algorithms for the Steiner Problem in Networks.
Ph.D. thesis, Saarland University.

[Ribeiro 2002] C.C. Ribeiro, E. Uchoa, and R.F. Werneck.
A hybrid GRASP with perturbations for the Steinerproblem in graphs.
INFORMS Journal on Computing, 14(3):228–246, 2002.

[Siebert 2018] Matías Siebert, Shabbir Ahmed, and George L. Nemhauser.
A Linear Programming Based Approach to the Steiner Tree Problem with a Fixed Number of Terminals.
Published in ArXiv 2018. URL: https://arxiv.org/pdf/1812.02237.pdf.

[Stanojevic 2006] Milan Stanojevic and Mirko Vujoševic.
An Exact Algorithm for Steiner Tree Problem on Graphs.
International Journal of Computers, Communications & Control Vol. I (2006), No. 1, pp. 41-46.

[Thorsten 1998] Thorsten Koch and Alexander Martin.
Solving Steiner Tree Problems in Graphs to Optimality.
An International Journal 32 (3), 207-232, 1998.

[Wang 2006] Xinhui Wang.
PhD Thesis. Exact Algorithms for the Steiner Tree Problem.
Department of Applied Mathema tics, Faculty of Electrical Engineering, Mathematics and Computer Science of the University of Twente, the Netherlands. URL: https://ris.utwente.nl/ws/portalfiles/portal/6039877.

[Winter 1987] Pawel Winter.
Steiner Problem in Networks: A survey.
Networks 17(2), 129–167, 1987.

[Wong 1984] Richard Wong.
A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph.
Mathematical Programming, 28:271 – 287, 1984.