Integer Linear Programming Models

for Media Streams Planning

Pavel Troubil and Hana Rudová
Faculty of Informatics
Masaryk University
Botanická 68a, 602 00 Brno
pavel@ics.muni.cz, hanka@fi.muni.cz

Abstract

Advanced collaborative environments frequently need to transfer highly demanding multimedia data streams with minimum possible latency. Since bandwidths of the streams are close to capacities of currently available links, routing of data transfers therefore requires planning with respect to link capacities and latency optimization. This paper describes integer linear programming techniques for optimal solution of the multimedia streams planning problem. Several methods for cycle avoidance in transmission graphs and network flows formulation are proposed. Performance of the methods was evaluated to identify their efficiency for real-time planning and to compare it against an earlier constraint programming approach applied in an application middleware called CoUniverse. According to the results, network flows formulation appears to be the most promising.

1. Introduction

High-end applications in the field of computer supported virtual collaboration are built from relatively high number of components performing specialized and resource intensive tasks (Holub et al. 2011). Configuration of many independent components (computational resources with data acquiring or consuming application, network routers and links, etc.) is intractable for an end-user. Communication has to be established in real-time, and also needs to seamlessly adapt to changing network environment, such as newly connected site or a failure of any component. Clearly the process has to be automated.

Decision of data distribution paths is a crucial part of the environment setup. Collaborative environments usually transmit several data streams concurrently, each of them from a single producer to several consumers. Bandwidth of the streams in the high-end environments is close to capacities of current network links (Holub et al. 2006). End-to-end latency is the crucial parameter for remote collaboration, hence needs to be minimized.

Transmission graph of each stream is a tree rooted at its producer with consumers at leafs. We suppose there are application-level data distributors at internal nodes of the tree. Although characteristic patterns of data transmissions might suggest the use of multicast, it is not suitable for needs of the collaborative environments. The multicast does not guarantee a protection from network congestion when several independent streams are distributed simultaneously. There are also concerns for its performance and availability. Several application-level multicast replacements have been proposed, be it standalone packet reflectors (Hladká et al. 2004) or whole architectures like OMNI (Banerjee et al. 2003).

The CoUniverse (Liška and Holub 2009) has been a pioneering application in the field of multimedia applications orchestration. It has been deployed for setup of environments for

distributed classrooms and low-latency videoconferences, which both transmit high quality videostreams. The routing problem has been called Media Streams Planning Problem (MSPP) by its authors. It is understood as a tree placement (Holub et al. 2011) rather than common path placement (Simonis 2006), since there are new copies of stream packets created in the network.

The MSPP is close to a multicommodity network flows problem (Ahuja et al. 1993). Unfortunately, solution methods for the network flows cannot be applied to the MSPP without further modifications, because distributors create new copies of data. However, they cannot be considered sources in the network flow problem, since we do not know which of them will create new packets in advance.

Integer linear programming has been widely used for various network optimization problems, like minimizing makespan of large files transfers (Zerola et al. 2009), or power optimizations of broadcasting in wireless networks (Das et al. 2003). Authors of (Das et al. 2003) used network flow constraints and also MTZ formulations of cycle avoidance (Miller et al. 1960) similar to those presented in this paper for the MSPP.

Although multicast is not a proper technology for purposes of the MSPP applications, its routing techniques might provide an inspiration for our work. Known methodologies for many variants of the multicast routing problems were overviewed in (Oliveira and Pardalos 2005). The OMNI architecture presented in (Banerjee et al. 2003) solves the problem for application-level multicast similar to ours, yet it considers single multicast tree only. The MSPP is the most closely related to a multicast packing problem (Chen et al. 2000). The multicast packing problem does not only consider placement of a single multicast tree, but handles simultaneous transmission in several multicast groups over a single underlying network. Optimal integer programming solution of the routing problem for multiple multicast groups was presented in (Noronha and Tobagi 1994). Authors propose model which combines binary variables for (stream, link) tuples with binary variables for (stream, link, consumer) triples. We have also tested this approach, yet did not include it in the paper due to space limitations and worse performance than the presented models.

Heuristics are mostly used to solve the multicast packing problem due to its high complexity. They are frequently based on Steiner trees (Wang et al. 2002), or genetic algorithms (Sanna Randaccio and Atzori 2007). Multicast packing has also been considered in form of multicast congestion problem (Lee and Cho 2004, Lu and Zhang 2005), where maximum congestion of network links is minimized instead of the latency.

Integer programming has also been used for problem of Steiner trees packing, where a maximal set of weighted Steiner trees should be placed to the network (Saad et al. 2008). Heuristic approaches to multicast packing might inspire our future work in further search for performance improvements.

The original implementation of the MSPP solver in the CoUniverse was based on constraint programming (Holub et al. 2011). Unfortunately, performance limitations allowed only for medium-sized input topologies. This paper presents and evaluates new integer linear programming (ILP) approaches to solve the MSPP, which aim to improve efficiency in comparison to the constraint programming (CP) solver. We have linearized the CP formulation into form of an integer linear program. Main contribution of this paper lies in proposal and performance analysis of several new approaches to avoidance of cycles in transmission graphs. We have also employed network flows formulation to the problem. Similarly to the CP approach, we aim to construct solution within a short time period (i. e.,

few seconds). It allows fast replanning acceptable by end users and adaptation to changing network environment.

2. Elements of the Problem and Terminology

The MSPP is motivated by real-world user's view of a network, which is the higher-level abstraction of physical topology. Actual physical topology is not generally known to users due to two main reasons: (1) the topology is subject of change, as network administrators might reconfigure it anytime, and (2) public exposure of the topology is considered to be a security threat, since potential attackers would know weak spots of the network. Components of the problem are consequently motivated and also limited by the high-level view of the network.

The aim of media streams planning is to find routes for transmission of multimedia data, i. e., continuous *streams* ($s \in S$). Each stream requires fixed (i. e., not changing in time) *bandwidth* $bw(s)$ on the network. Streams are handled by three kinds of *applications*. Each stream is produced by a single *producer* ($producer(s) \in P$), received by a non-empty set of *consumers* ($consumers(s) \in C, consumers(s) \neq \emptyset$), and on their route forwarded by *distributors* ($d \in D$). Distributors may also create multiple copies of the stream packets, substituting multicast functionality on application level of the ISO/OSI. Each application can handle one stream at most.

Applications are running on a server or desktop computer. These computers make up a set of network *nodes* ($v \in V$). Lower level network elements (like switches or routers) are not included in the set V, as their number and placement are not known to the users. There are two restrictions on what applications are allowed to run at a same node: (1) if there is a distributor running at a node, no other application is allowed to run there, and (2) several consumers and/or producers may run at a single node as long as they handle different streams. If an application $a \in P \cup D \cup C$ runs on the node v, we write $a \in v$. Nodes are organized into *sites*, which typically represent geographical collocation of nodes, e. g., at a videoconferencing room. In a typical configuration, there are several nodes at each site, each of them running a single application.

Several network interfaces ($i \in I$) are configured on each node v (denoted by $i \in v$). An interface has a limited transmission *capacity* $capI(i)$. Each interface also belongs to exactly one of user-defined *subnetworks*. All interfaces within a subnetwork are considered mutually reachable in the underlying network.

The mutual reachability is expressed by a couple of directed *links* between each pair of interfaces within a subnetwork. Each link $l \in L$ has *latency* $latency(l)$ and transmission *capacity* $cap(l)$. Since physical network links are not known to the users, it might occur that several of these links share one physical link, decreasing their actually available capacity. The CoUniverse periodically measures link capacities. If it is insufficient on some links, replanning from scratch is issued.

We say that link l between interfaces (i_1, i_2) connects nodes (v_1, v_2) if and only if $i_1 \in v_1 \wedge i_2 \in v_2$. Source and target nodes of a link l are denoted as $begin(l)$ and $end(l)$, respectively. A set of all links connected to an interface i is called $links(i)$.

For an application $a \in P \cup D \cup C$ on a node v, we define $inlinks(a)$ as a set of links ending in the node v, i. e., $l \in inlinks(a) \Leftrightarrow a \in end(l)$. We extend this notation to a set of

applications $A \subseteq P \cup D \cup C$: $l \in inlinks(A) \Leftrightarrow \exists a \in A: a \in end(l)$. We also define sets of outgoing links $outlinks(a)$ and $outlinks(A)$ analogously.

Before describing our solution approaches, we need to clearly explain what are capabilities and restrictions of distributors. They have two important roles: (1) forwarding the stream between nodes which are not directly connected by links, and (2) creating multiple copies of the stream to allow reception by multiple consumers, since producers always create only a single copy. On the other hand, split streaming is not allowed on distributors. It is not allowed to transfer 1.5Gbit stream through two 1Gbit links, or use split streaming for load balancing. All packets from a producer to a single consumer have to be routed along a single path. Split streaming could lead to undesirable packet reorderings, which could not be handled by buffering without significant delay.

3. Core Constraints

In this section, we describe a set of constraints, which are further used in all subsequent methods. Individual methods (described in next sections) add various constraints upon this core constraints set. There are also differences in solution strategies, yet first step always is to post the core constraints in a backend solver.

This set of constraints is strongly based on the CP model presented in the previous work on the MSPP (Holub et al. 2011). However, some constraints in the CP model are not linear and had to be reformulated. This especially holds for constraints maintaining the tree shape of data distribution graph (see constraints (9) to (11)).

For each stream s and link l, there is a binary decision variable $x_{s,l}$ called *streamlink*. The streamlink $x_{s,l}$ equals to 1 if and only if the stream s is transferred over the link l. We also call streamlink *active* or *inactive* if it equals to 1 or 0, respectively.

Overall transmission latency is an optimization criterion in the MSPP. We formulate the objective function as a sum of latencies of all active streamlinks.

$$min \sum_{s \in S} \sum_{l \in L} latency(l) \cdot x_{s,l} \tag{1}$$

There are three sets of constraints handling network capacity limitations. Total bandwidth of streams transferred over each interface i has to be lower than capacity of that interface (2). Similarly, total bandwidth of streams transferred over each link l cannot exceed capacity of the link (3). It is not allowed to transfer a stream s through a link l of insufficient capacity (4). This constraint is redundant (follows directly from (3)).

$$\sum_{s \in S} \sum_{l \in links(i)} bw(s) \cdot x_{s,l} \leq capI(i) \qquad \forall i \in I \tag{2}$$

$$\sum_{s \in S} bw(s) \cdot x_{s,l} \leq cap(l) \qquad \forall l \in L \tag{3}$$

$$x_{s,l} = 0 \qquad \forall s \in S \, \forall l \in L \; s.t. \; bw(s) > cap(l) \tag{4}$$

An application cannot send a stream s through arbitrary outgoing links. There has to be either consumer of the stream s or a distributor on a target node of the link (5). Similarly, an application cannot receive a stream s over any incoming link. There has to be either

distributor or producer of the stream s on a source node of the link (6). The former constraint cuts off only suboptimal solutions. Target nodes of links deactivated by constraint (5) cannot receive any stream or forward it further, hence are not usable in any distribution tree. If there is a feasible solution with any of these streamlinks active, there still has to be a distribution tree which does not include any of them. Since their activation increases value of objective function, they would be deactivated in optimization process otherwise.

$$x_{s,l} = 0 \qquad \forall s \in S \, \forall l \notin inlinks(D \cup consumers(s)) \tag{5}$$

$$x_{s,l} = 0 \qquad \forall s \in S \, \forall l \notin outlinks(\{producer(s)\} \cup D) \tag{6}$$

Each producer creates a single copy of the stream and sends it to another application. It is not capable to create multiple copies of the data, hence each producer sends its stream over exactly one streamlink (7).

$$\sum_{l \in outlinks(producer(s))} x_{s,l} = 1 \qquad \forall s \in S \tag{7}$$

Each consumer handles single application only and split streaming is forbidden, thus consumers are not allowed to receive their stream by more than one link (8).

$$\sum_{l \in inlinks(c)} x_{s,l} = 1 \qquad \forall s \in S \, \forall c \in consumers(s) \tag{8}$$

Since each distributor can handle only one stream, there may be at most one incoming link active (9). Distributor has to forward an incoming stream, i. e., the number of active outgoing links has to be at least the same as number of incoming links (10). This constraint cuts off only suboptimal solutions. We also need to forbid each distributor to send a stream it does not receive. It is clear that the maximum possible number of active streamlinks beginning in a distributor d equals to $outlinks(d)$. Consequently, the constraint (11) is guaranteed to hold independently on the number of active outgoing links if a stream is received and forwarded.

$$\sum_{s \in S} \sum_{l \in inlinks(d)} x_{s,l} \leq 1 \qquad \forall d \in D \tag{9}$$

$$\sum_{l \in inlinks(d)} x_{s,l} \leq \sum_{l \in outlinks(d)} x_{s,l} \qquad \forall s \in S \, \forall d \in D \tag{10}$$

$$|outlinks(d)| \cdot \sum_{l \in inlinks(d)} x_{s,l} \geq \sum_{l \in outlinks(d)} x_{s,l} \qquad \forall s \in S \, \forall d \in D \tag{11}$$

4. Cycle Constraints

The core constraints presented in Sec. 3 do not avoid existence of cycles among distributors in feasible solutions. An example of possible cycle in a solution is depicted in Fig. 1. There is a consumer receiving the stream from a distributor in the cycle, but not any path from the producer to this consumer. Such cycle might occur among several distributors, which are close to each other (on low-latency links), yet distant from the producer (residing in another city, or continent). A cycle may therefore improve value of the objective function compared to solution where all consumers are connected to the producer properly. We proposed two

alternative strategies of cycles handling: immediate avoidance posts all cycle avoidance constraints in the backend solver together with the core constraints, while delayed elimination posts only some of them just in case when cycles actually appear in solution implied by the core constraints.

Figure 1 An example of possible cycle in solution. Black node is a producer, gray ones are distributors, and producers are the white ones.

4.1. Immediate Avoidance

Similarly to Holub et al. (2011), we implemented two alternative methods of cycle avoidance: one close to subtour formulation known from the travelling salesman problem, and the other related to the MTZ formulation (Miller et al. 1960). A basic solution strategy is to post all cycle avoidance constraints in backend solver at once, preventing cycles from appearing. The subtour formulation sets upper bounds on the number of active streamlinks among distributor tuples. Each graph with n vertices and more than $n-1$ edges contains certainly a cycle. If we force each k-vertices subgraph of the graph to contain at most $k-1$ edges, there will be no cycle in the graph. For each k-tuple ($2 \leq k \leq D$) of mutually connected distributors, we put an upper bound $k-1$ on number of active links among them (12).

$$\sum_{l \in outlinks(D_k) \cap inlinks(D_k)} x_{s,l} \leq k-1 \qquad \forall s \in S \, \forall D_k \; s.t. \; D_k \subseteq D, k \in \{2,\dots,n\}, |D_k| = k \tag{12}$$

In the symmetric travelling salesman problem, it is sufficient to post the subtour avoidance constraints for $k \leq n/2$, since existence of a cycle among more than $n/2$ vertices implies existence of a smaller cycle. Unfortunately, the assumption does not hold for the MSPP.

The MTZ exploits tree structure of data distribution. In a tree, each node has greater distance from the root than its parent. In other words, target node of each link has a greater distance than its source node. If there is a cycle in the graph, there also has to be an edge from a higher distance node to a lower distance one. The formulation adds a set of continuous decision variables $y_{s,v}$ to the model. They denote distance of a node v from root in the transmission graph of the stream s. Their bounds are set to $0 \leq y_{s,v} \leq |D| - 1$. The following set of constraints can hold only if the graph is a tree.

$$y_{s,begin(l)} - y_{s,end(l)} + 1 \leq |D| \cdot (1 - x_{s,l}) \qquad \forall s \in S \, \forall l \in L \tag{13}$$

4.2. Delayed Elimination

In our preliminary performance evaluations, we have observed that the immediate avoidance constraints decrease performance of the solver significantly. In the subtour formulation, the number of constraints is very high. The MTZ formulation uses lower number of constraints,

yet the formulation is not as tight (Pataki 2003). The performance for the MSPP is similar for both formulations. Instead of avoiding cycles before they can emerge, we proposed to eliminate them by a simple separation algorithm only if they actually appear in the solution.

The core constraints are solved without any cycle avoidance constraints. Once the solution is found, cycles are detected in the distribution graphs per stream. We use the algorithm for enumeration of strongly connected components instead of more performance demanding algorithm for cycles enumeration, since core constraints imply equivalence between strongly connected components and cycles.

All cycles found in the previous step have to be eliminated. However, adding all cycle avoidance constraints and solving the model from scratch would require the same time as running whole model from Sec. 4.1. Instead, we fix the solution of non-cycled streams and eliminate cycles only in the affected ones. In the delayed subtour method, we enumerate all cycles and add single constraint for each of the cycles. However, we add all constraints for the relevant streams in the delayed MTZ method.

Once we apply the MTZ constraints on all cycled streams, they guarantee to imply solution without cycles. In contrast, adding particular subtour elimination constraints does not imply cycle-free solution. Naturally, other cycles might appear after next run of the solver. If it happens, we repeat the procedure until an acyclic solution is found.

The performance is increased at the cost of potentially suboptimal solution. In extreme cases, a feasible solution might not be found even if it exists. In such case, we fall back to the model with cycle avoidance constraints to find out whether there actually is a feasible solution or not.

5. Flow Constraints

Current mixed integer programming solvers perform very efficiently on problems with network flow structure. Attempting to exploit this functionality, we proposed a set of network flow constraints. These constraints also naturally cope with cycles and no other cycle handling methods are needed.

However, Kirchhoff's laws do not hold for the MSPP streams in distributor nodes. Therefore, we introduce the second set of integer variables called linkflows in the model: $z_{s,l}$. For each stream s and link l, variable $z_{s,l}$ equals to the number of consumers, which receive the stream s through the link l. On the only active link from $producer(s)$, $z_{s,l}$ equals to $|consumers(s)|$, which is also an upper bound for all $z_{s,l}$ variables handling the given stream s. On all active links to consumers, the linkflows are equal to 1. Kirchhoff's laws (and consequently network flow constraints) may hold for linkflows in contrast to streamlinks. We keep all core constraints, but do not apply either of above mentioned cycle avoidance or elimination methods.

We set sums of the linkflows values for links adjacent to producers and consumers, similarly as for sources and sinks in network flows problem.

$$\sum_{l \in outlinks(producer(s))} z_{s,l} = |consumers(s)| \qquad \forall s \in S \tag{14}$$

$$\sum_{l \in inlinks(consumers(s))} z_{s,l} = 1 \qquad \forall s \in S \tag{15}$$

We also add flow conservation constraints at distributors.

$$\sum_{l \in inlinks(d)} z_{s,l} = \sum_{l \in outlinks(d)} z_{s,l} \qquad \forall s \in S \, \forall d \in D \qquad (16)$$

And finally, the linkflows are tied with streamlinks.

$$z_{s,l} \geq x_{s,l} \qquad \forall s \in S \, \forall l \in L \qquad (17)$$

$$|consumers(s)| \cdot x_{s,l} \geq z_{s,l} \qquad \forall s \in S \, \forall l \in L \qquad (18)$$

6. Evaluation and Results

We implemented all solution methods described in Secs. 3–5 as an MSPP solving module in the CoUniverse. The module is written in the Java programming language as well as the whole CoUniverse. We used the Gurobi Optimizer version 4. 0. 1 as a backend mixed integer programming solver. To allow a comparison, we also measured performance of the constraint programming model presented in (Holub et al. 2011). The CP model was solved by the Choco library version 2.1.1 in our experiments. On the contrary, we do not present results of the MTZ-based immediate cycle avoidance method for sake of simplicity. They are generally not better then results of the subtour-based method.

6.1 Data Sets and Experimental Setup

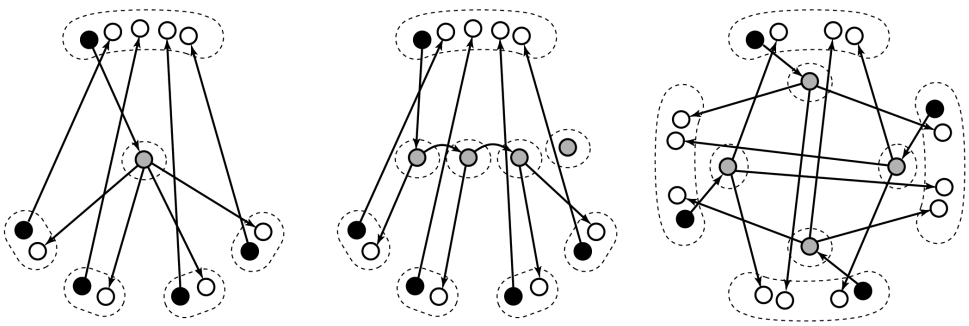
We used three types of input data sets (further called topologies) which simulate typical patterns of multimedia transmissions in collaborative environments:

a) *1:n-s* topologies: one site si transmits a stream to all other sites through a single distributor. Each of the other sites transmits their stream back to si.

b) *1:n-r* topologies: they are similar to the previous type with an exception of higher number of distributors, there is one for each site except si. Both *1:n* topologies are typical cases for distant teaching and presenting.

c) *m:n* topologies: each site transmits a stream to all other sites. There is at least one distributor for each site (exactly one for input sizes presented in this paper).

Figure 2 Schemes of topologies used for measurements (taken from (Holub et al. 2011)): (a) 1:n-s, 5 sites (b) 1:n-r, 5 sites (c) m:n, 4 sites



Schemes of all three types of topologies are depicted in Fig. 2. These topologies were taken from (Holub et al. 2011) to allow comparison with their results. Their paper also presents more detailed description of the topologies, which could not be written out in detail here due to space limitations.

Numbers of nodes and links in instances of the topologies (parameterized by the number of sites) are shown in Table 1. The number of links is presented after an elimination process (same as the one applied in (Holub et al. 2011)), which decreases number of decision variables by removing unusable links.

Table 1: Parameters of selected topologies used for performance measurement

Topology	1:n-s-2	1:n-s-4	1:n-s-8	1:n-s-16	1:n-s-32	
Nodes	5	11	23	47	95	
Edges	6	28	120	496	2,016	
Topology	1:n-r-2	1:n-r-4	1:n-r-8	1:n-r-12	1:n-r-16	1:n-r-20
Nodes	5	13	29	45	61	77
Edges	6	54	294	726	1350	2,166
Topology	m:n-2	m:n-3	m:n-4	m:n-5	m:n-6	m:n-7
Nodes	6	12	20	30	42	56
Edges	6	45	112	225	396	637

All measurements were performed on a PC equipped with Intel Xeon 5160 @ 3.0 GHz quadcore processor and 6GB RAM, running Linux with 2.6.22-17 kernel and Java SDK 1.6.0 in a virtualized Xen environment. Options for java were set to `-server -da -dsa`. We ran 30 continuous runs of each measurement, and only the last 15 of them were taken into account. We did not limit the number of processor cores available for the Gurobi optimizer to allow improved efficiency for larger topologies.

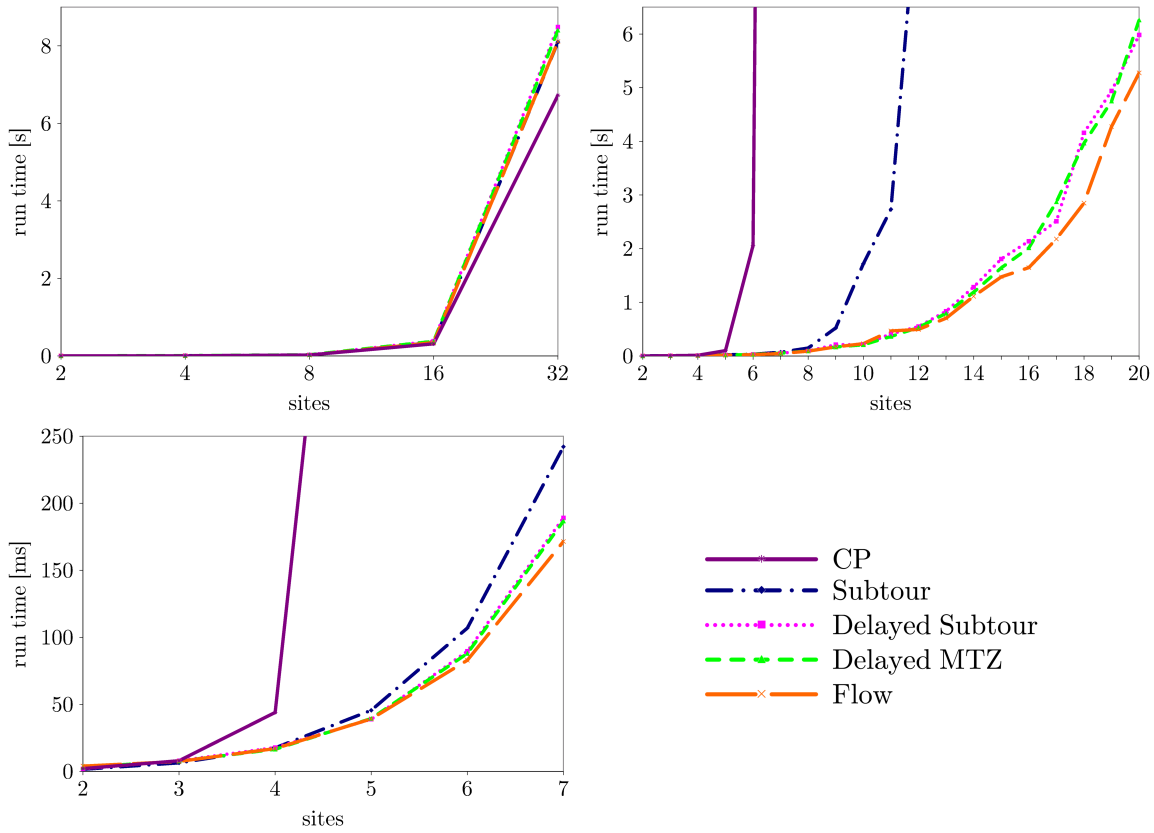
6.2 Performance Measurements

We consider performance the main indicator of quality of all presented methods, as they produce equivalent solutions (although delayed elimination methods are not guaranteed to solve the problem to optimality, they do so for our data sets). We measured time needed for each of them to solve several instances of each topology type. Size of the instances is given by the number of sites, as they generally correspond to the number of participants connected to a collaborative application. We are trying to find the biggest instance of each type which can be solved in real-time (i. e., within about 5 seconds). The graphs in Fig. 3 depict average times needed to solve the individual topologies.

Top left chart in Fig. 3 shows that all presented solution methods are virtually equal in performance for the *1:n-s* topologies. Performances of all ILP-based methods are very close to each other. The CP method performs slightly better for the largest presented topology. We identified that the difference lies in preparation of the model, i. e., performance of the backend solver API, which takes at least 95% of the required time. The model preparation procedure is slightly faster for the Choco library than the Gurobi solver.

Top right chart in Fig. 3 shows much greater performance differences among the methods in solution of the *1:n-r* topologies. The CP method performs clearly the worst, solving just 6 sites within 5 seconds. The core constraints with subtour-based cycle avoidance adhere with the time limit up to 11 sites. The three remaining methods reach the limit at 19–20 sites. Both delayed cycle elimination methods seem to perform very similarly. On larger topologies, they require slightly more time than the flow constraints, which exceed the 5 seconds limit by

Figure 3 Average computational time for the topologies. Top left: ***1:n-s***; top right: ***1:n-r***; bottom left: ***m:n***.



a few tens of milliseconds for 20 sites. The difference between flow constraints and delayed cycle elimination is greater than average time needed to detect and possibly eliminate the cycles.

When delayed cycle elimination methods are used, no cycles appear in solutions up to 12 sites. From 13 sites to 18, the numbers of cycles in the solutions rises, while it decreases for 19 and 20 sites. Out of the 15 experimental runs, number of needed additional cycle elimination runs is depicted in Table 2.

Table 2 Overall numbers of additional solver runs required to eliminate cycles in 1:n-r topologies

	13	14	15	16	17	18	19	20
Subtour	4	8	7	11	11	31	7	8
MTZ	1	3	5	7	12	12	11	7

Bottom left chart in Fig. 3 shows that all presented ILP-based methods outperform the CP model for the *m:n* topologies. Five sites are the limit for the CP model (solved within 700 ms), while the ILP-based methods manage to solve up to 7 sites within 250 ms. However, difficulty of the problem rises extremely with adding of the 8th site. One distributor per site is insufficient for 8 sites, since the distributor cannot manage more than six clients for highly-demanding multimedia applications, e. g., uncompressed HD-video transmissions. Sudden increase of the distributors number from 7 to 16 causes also sudden increase in time required to solve the model. It takes about 1 minute to solve the model with the flow constraints, which are the most effective ones.

Redundant constraints. We have also evaluated influence of the redundant constraints on performance of each method. We divided the core constraints into three groups: (1) redundant, (2) only suboptimal solutions cutting, and (3) the others. We measured the performance when (a) all of them were applied, (b) the redundant ones were not applied, and finally (c) only the last group was applied.

Unfortunately, we cannot include full results and evaluation due to space restrictions. It generally seems advantageous to include all constraints which tighten the formulations, even if they only cut off the suboptimal solutions. Results without these constraints are worse than results with them for almost all topologies. However, effect of the redundant constraints is much less clear, especially on both *1:n* topologies. They sometimes slightly improve the performance, yet there are topologies where the results are marginally better without the redundant constraints. Their effect also rises with higher number of possible solutions.

Although significance of the redundant constraints is questionable for the *1:n* topologies, we keep them in the model for their clearly observable effect on the *m:n* topology. They improve performance of all by order of magnitude methods on these topologies with an exception of the flow method, where the effect is much more subtle.

7. Conclusions

We have introduced several ILP-based methods to solve the media streams planning problem. We suggested a set of core constraints partially based on the CP model published by the CoUniverse authors (Holub et al. 2011). The core constraints make a basis of all solution methods. These methods differ in the set of constraints added to the core constraints. Two well-known cycle avoidance methods were applied as well as in the CP model. We proposed to replace generally applied cycle avoidance constraints by elimination of only those cycles, which actually emerge in the solution. We also proposed solution method exploiting network flows formulation.

We have evaluated performance of all methods and compared it to performance of the CP model. There were three types of topologies used for the evaluation of the methods. For the simplest type, all methods perform similarly to the CP model. However, all proposed ILP-based methods significantly outperform the CP model on the two more complex types of topologies. We consider this an evidence of their better suitability for more difficult instances of the problem. Out of the ILP-based methods, the one using network flow constraints has shown itself to perform the best.

In a near future, we plan to work on heuristic algorithms for solution of this version of the MSPP to allow further performance improvements of the solver. Further, we aim to solve more general versions of the problem. We plan to develop solvers capable of coping with partial knowledge of physical topology and availability of multicast in some subnetworks. We also plan to handle transcoding of multimedia streams in the network to influence their quality by available network throughput. We will also consider more complex objective functions dealing with quality of service.

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J., 1993. *Network Flows.* Prentice Hall.

Banerjee, S., Kommareddy, C., Kar, K., Bhattacharjee, B., and Khuller, S., 2003. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of IEEE INFOCOM 2003*, pp 1521–1531.

Chen, S., Gűnlűk, O., and Yener, B., 2000. The multicast packing problem. *IEEE/ACM Transactions on Networking*, 8(3):311–318.

Das, A. K., Marks, R. J., El-Sharkawi, M., Arabshahi, P., and Gray, A., 2003. Minimum power broadcast trees for wireless networks: Integer programming formulations. In *Proceedings of the IEEE INFOCOM 2003*, pp 245–248.

Hladká, E., Holub, P., and Denemark, J., 2004. User empowered virtual multicast for multimedia communication. In *3rd International Conference on Networking (ICN'04)*, pp 338–343.

Holub, P., Matyska, L., Liška, M., Hejtmánek, L., Denemark, J., Rebok, T., Hutanu, A., Paruchuri, R., Radil, J., and Hladká, E., 2006. High-definition multimedia for multiparty low-latency interactive communication. *Future Generation Computer Systems*, 22:856–861.

Holub, P., Rudová, H., and Liška, M., 2011. Data transfer planning with tree placement for collaborative environments. *Constraint,* 16(3):283–316.

Lee, C. Y. and Cho, H. K., 2004. Multiple multicast tree allocation in IP network. *Computers & Operations Research*, 31(7):1115 – 1133.

Liška, M. and Holub, P., 2009. CoUniverse: Framework for building self-organizing collaborative environments using extreme-bandwidth media applications. In *8th International Conference on Networks (ICN'09),* pp 259–265.

Lu, Q. and Zhang, H., 2005. Implementation of approximation algorithms for the multicast congestion problem. In *Experimental and Efficient Algorithms*, volume 3503 of Lecture Notes in Computer Science, pp 285–294. Springer Berlin / Heidelberg.

Miller, C. E., Tucker, A. W., and Zemlin, R. A., 1960. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7:326–329.

Noronha, Jr., C. A. and Tobagi, F. A., 1994. Optimum routing of multicast streams. In *IEEE INFOCOM 1994*, pp 865–873.

Oliveira, C. A. S. and Pardalos, P. M., 2005. A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8):1953 – 1981.

Pataki, G., 2003. Teaching integer programming formulations using the traveling salesman problem. *SIAM Review*, 45:116–123.

Saad, M., Terlaky, T., Vannelli, A., and Zhang, H., 2008. Packing trees in communication networks. *Journal of Combinatorial Optimization*, 16(4):402–423.

Sanna Randaccio, L. and Atzori, L., 2007. Group multicast routing problem: A genetic algorithms based approach. *Computer Networks*, 51(14):3989–4004.

Simonis, H., 2006. Constraint applications in networks. In *Handbook of Constraint Programming*, pp 875–903. Elsevier.

Wang, C.-F., Liang, C.-T., and Jan, R.-H., 2002. Heuristic algorithms for packing of multiple group multicasting. *Computers & Operations Research*, 29(7):905–924.

Zerola, M., Barták, R., Lauret, J., and Šumbera, M., 2009. Efficient multi-site data movement in distributed environment. In *10th IEEE/ACM International Conference on Grid Computing*, 2009, pp 171–172.