# Core-Based GRASP for Delay-Constrained Group Communications

Zrinka Lukač

Faculty of Economics and Business, University of Zagreb, Zagreb, Croatia
zrinka.lukac@zg.t-com.hr

Manuel Laguna

Leeds School of Business, University of Colorado Boulder, USA
laguna@colorado.edu

**Abstract**: The recent development in network multimedia technology has created numerous real-time multimedia applications where the Quality-of-Service (QoS) requirements are quite rigorous. This has made multicasting under QoS constraints one of the most prominent routing problems. We consider the problem of the efficient delivery of data stream to receivers for multi-source communication groups. Efficiency in this context means to minimize cost while meeting bounds on the end-to-end delay of the application. We adopt the multi-core approach and utilize SPAN [10] —a core-based framework for multi-source group applications— as the basis to develop greedy randomized adaptive search procedures (GRASP) for the associated constrained cost minimization problem. The procedures are tested in asymmetric networks and computational results show that they consistently outperform their counterparts in the literature.

**Keywords:** multicasting; QoS; core-based approach; routing algorithms, GRASP

December 10, 2012

## 1. Introduction

The recent proliferation of different multimedia real-time applications over the internet —such as Voice-over-IP (VoIP), videoconference, TV over the internet, radio over the internet, multipoint video streaming, distance learning, games, etc.— has created the need for scalable and efficient network support that is capable of providing the level of performance needed for these applications to function properly. The real-time transmission of multimedia information over the internet is characterized by large amount of data that have to be processed and transmitted simultaneously to multiple recipients through underlying computer networks. The transmission must be done under rigorous Quality-of-Service (QoS) constraints in order to ensure that audio and video data are delivered smoothly to the intended recipients. For instance, according to the International Telecommunication Union one-way transmission recommendations [8], data stream for video/audio conferencing with real-delivery of voice data should be delivered within 400ms. Such delay is acceptable in most situations, though users often start to become dissatisfied if the delay exceeds 200ms. Likewise, latency requirement for first-person shooter games is 100ms [8]. Since most of the multimedia applications are delay-sensitive, the problem of establishing the group communication with minimum cost while satisfying the delay constraints has become one of the most relevant QoS problems.

Simultaneous transportation of information between one or many senders and multiple recipients is implemented through a mechanism called *multicasting*, which avoids sending a copy of data stream to each recipient. Multicasting protocols are implemented through either building source- or core-based trees. In both of these protocols (i.e., source-based and core-based) each sender sends only one data packet, which is then duplicated at branching points and forwarded to multiple recipients. The first multicast protocols were source-based (DVMRP [20], MOSPF [14][15] and PIM-DM [3]). They establish group communication by building a separate shortest-path tree from source to all of the receivers. These protocols are suitable for small-scale applications only since they do not scale well. They tend to produce large message overhead because one piece of the state information per source and per group is kept in each router. They do not minimize the total cost of distribution and may fail if the underlying unicast routing is asymmetric. On the other hand, the core-based protocols are such that they choose one or more routers as cores and then build shortest-path trees from cores to all the multicast group members. Hereby senders transmit data to the cores, which then forward it to all the recipients. Depending on the number of cores the protocol is set to choose, they are classified as either single-core or multi-core based protocols. The first core-based protocols were single-core (CBT [1][2] and PIM [4]). Unlike source-based trees, single-core trees are scalable and much easier to maintain since state information is stored one per group instead of one per source. Also, they offer better bandwidth utilization and produce lower message overhead. They are suitable for sparsely distributed receivers and are preferred to source-based trees in case of multiple sources in the multicast group [21]. However, single core-based trees have some serious flaws when compared to source-based trees. They produce higher delay (since data has to travel from senders to the cores first), they suffer from traffic concentration on links that converge towards the core and have poor fault tolerance in case of core failure. The introduction of several independent cores in multi-core protocols (OCBT [17][18], “Sender-To-Many” [23], “Members-To-Many” [23]) has significantly improved the performance of core-based protocols, making them a viable alternative to source-based trees. Compared to single core-based

protocols, they result in less delay and incur in less total cost since nodes are more likely to locate nearby cores. They operate in a broader solution space and therefore are able to provide a solution in cases when a single-core solution does not exist. Moreover, they provide better fault tolerance in case of core failure and result in less traffic concentration around the cores. In general, for QoS-constrained applications with sparsely populated groups in a distributed routing environment the core-based approach is the preferred method providing more efficient solutions [23]. An extensive classification and comparative analysis of core-based multicast routing protocols can be found in [9].

We consider the problem of cost minimization of many-to-many multicast group communication in a distributed, asymmetric environment under a hard end-to-end delay constraint. The most popular approach for this QoS-constrained many-to-many problem is GREEDY, a procedure proposed by Salama [16], which operates in a symmetric, centralized deployment. A major improvement to GREEDY is the SPAN framework [10] and its extensions [11] proposed by Karaman and Hassanein, which has consistently shown better perfromance than other approaches in the literature. SPAN is a generic core-based framework for asymmetric, decentralized delay-constrained multicast routing in multi-source groups that consists of core selection and tree construction modules. It operates in an extended solution space for which a core is not necessarily serving all the sources in the group and where different sources can send data stream to the same receiver through different cores.

We develop and compare several core-based routing procedures operating in a distributed and asymmetric environment in which link weights are not necessarily equal in both directions. Each procedure consists of a core selection and a tree construction module applied sequentially. Procedures are created by different combinations of core selection and tree construction modules. Our core selection module embeds the selection component of SPAN within a GRASP (greedy randomized adaptive search procedures) framework. GRASP is a multi-start metaheuristic based on semi-greedy solution constructions and local search [5][6]. We consider two different tree construction modules, one of which is a modified version of the SMT algorithm [19] for the constrained problem. By combining different components, we have created and tested the performance of 21 heuristics —18 GRASP based heuristics and 3 variants of the SPAN framework obtained by modifying the tree construction module. The computational results show that, in terms of cost and QoS, most of these heuristics consistently outperform SPAN as well as alternative approaches known in the literature.

## 2. The SPAN Framework

SPAN operates on the local-distance information available at the routers and is the first distributed, asymmetric framework in the literature that provides solutions for constrained, core-based multi-source communication groups [10]. It chooses the set of cores and constructs multipoint trees so as to minimize cost under a hard constraint that limits the end-to-end delay. The solutions are constructed with the goal of optimizing the total hop-count, where the total hop-count measures the total number of links on a group communication tree.

Figure 1 depicts how the choice of an objective function affects the structure of the multicast transmission for a network with a total of 100 nodes, where each group of senders and receivers consists of 5 nodes and the groups do not overlap. The squares denote senders, while triangles denote receivers. Three minimization objective functions are considered in Figure 1: a) cost, b) maximum end-

to-end delay and c) hop-count. The solutions depicted in Figure 1 were obtained by solving with CPLEX the mixed integer program shown in the Appendix.

— Figure 1 —

We now describe the SPAN framework and adopt the same terminology as presented in [10]. Multi-core based protocols prior to SPAN explored solutions in what Karaman and Hassanein call singular solution space where each core serves all sources in the group and uniquely defines the shared tree rooted at that core. They introduce the extended solution space —non-singular solution space in their terminology— in which a core is not necessarily serving all the sources and where each receiver can be served by multiple cores for different sources in the group. Such a solution space extends the range of potential solutions for constrained multicast problem and may produce solutions that are not feasible in the singular solution space. Moreover, even if the singular solution space contains feasible solutions, the space may not necessarily contain the optimal solution. This is why non-singular solution spaces are preferred to singular solution spaces.

SPAN operates in non-singular solution spaces and contains core-selection and tree construction modules. Since in a non-singular solution space a core is not necessarily serving all the sources in the group, the framework must keep track of which cores serve which source-receiver pairs and then construct and maintain core trees and source trees separately. Core trees and source trees may share links. In order to describe the methodology, we adopt Karaman's and Hassanein's terminology [10]. Let $R$ denote the set of all receivers and let $S$ denote the set of all sources in the multicast group. A receiver $r$ is said to be *dominated* by a core $c$ for a source $s$ if there exists a path $p$ from $s$ to $r$ through $c$ which does not violate the delay bound of the application. In that case we also say that a core $c$ serves $r_s$. We denote the set of all receivers dominated by the core $c$ for source $s$ as $D(c,s)$. Similarly, $D(c,S')$ denotes the set of all receivers dominated by the core $c$ for all sources from the subset of sources $S' \subseteq S$. If we examine the union of paths connecting each source in $S'$ with receivers in $D(c,S')$ through core $c$, we obtain a tree structure where all sources in $S'$ are connected to core $c$ and this core is connected to all the receivers in $D(c,S')$. The tree which connects core $c$ with the set of receivers $D(c,S')$ is also referred to as a core tree rooted at $c$. A *multipoint tree* is the union of trees corresponding to the sets $D(c',S)$ for $c' \in C'$ where $C'$ is the subset of cores such that $R = \bigcup_{c' \in C'} D(c',S')$. In such a union all the receivers in $R$ are dominated by some of the cores $c'$ in $C'$, where $c'$ serves identically all the sources in $S'$. Within the multipoint tree, each source in $S'$ is connected to each core $c'$ in $C'$ and each of these cores is the root of the core tree spanning receivers in $D(c',S)$. Note that a multipoint tree does not necessarily have a tree structure. However, for each source there is a distinct tree which connects it with the receiver set. We now illustrate these concepts with the following example.

**Example:** Let us consider a network with 2 sources, $S = \{s_1, s_2\}$, 5 receivers, $R = \{r_1, r_2, r_3, r_4, r_5\}$ and 3 cores, $C = \{c_1, c_2, c_3\}$, such that $D(c_1, s_1) = \{r_1, r_2, r_3\}$, $D(c_2, s_1) = \{r_2, r_3, r_4\}$, $D(c_3, s_1) = \{r_4, r_5\}$, $D(c_1, s_2) = \{r_1, r_2\}$, $D(c_2, s_2) = \{r_2, r_3, r_4\}$ and $D(c_3, s_2) = \{r_5\}$. For each solution in a singular solution space each core should serve all sources in the group. One such solution is determined by the choice of domination sets $D(c_1, S) = \{r_1, r_2\}$, $D(c_2, S) = \{r_3, r_4\}$ and $D(c_3, S) = \{r_5\}$ and is depicted in Figure 2. Figure 2a depicts the tree corresponding to $\bigcup_{c \in C} D(c, s_1)$, i.e. it shows the paths along which source $s_1$

sends data stream to the entire receiver set. Figure 2b depicts the tree corresponding to $\bigcup_{c\in C} D(c, s_2)$, i.e. the tree for which source $s_2$ sends data to all the receivers. Dashed lines indicate unused links. The multipoint tree corresponding to the overall solution, i.e. corresponding to $\bigcup_{c\in C} D(c, S)$, is depicted in Figure 2c. The multipoint tree contains two source-rooted trees, one for each source, and three core-rooted trees, one for each core. Each source-rooted tree spans the entire set of cores, while each core-rooted tree spans the receivers from the respective domination set $D(c_i, S)$. Each core tree identically serves all sources in $S$.

— Figure 2 —

Let us now consider a solution in a non-singular solution space. One such solution is defined by domination sets $D(c_1, s_1) = \{r_1, r_2, r_3\}$, $D(c_2, s_1) = \{\}$, $D(c_3, s_1) = \{r_4, r_5\}$, $D(c_1, s_2) = \{r_1\}$, $D(c_2, s_2) = \{r_2, r_3, r_4\}$, and $D(c_3, s_2) = \{r_5\}$. Figure 3a depicts how source $s_1$ serves the entire receiver set, i.e. it depicts the tree related to $\bigcup_{c\in C} D(c, s_1)$. Likewise, Figure 3b depicts the tree related to $\bigcup_{c\in C} D(c, s_2)$, i.e. it depicts the way source $s_2$ sends data to receivers. Again, dashed lines indicate the unused links in the tree. In a non-singular space, a receiver may be assigned to different cores for different sources. For example, receiver $r_3$ is dominated by core $c_1$ for source $s_1$, and at the same time, it is dominated by core $c_2$ for source $s_2$. Unlike in a singular solution space where there's a unique core cluster serving all the sources, here we have more than one core cluster for a single source. The overall solution relative to $\bigcup_{c\in C} D(c, S)$ is depicted in Figure 3c. The links serving sources $s_1$ and $s_2$ are denoted by solid lines and a thick solid line indicates that a link is serving both sources. For example, core $c_1$ is serving both sources $s_1$ and $s_2$ for receiver $r_1$. Note that in the combined multipoint tree the core trees are not identically serving all the sources in the group. Since in a non-singular solution space the receiver dominations are examined for each source separately, unlike in singular solution space where the domination of a particular receiver is examined for the whole source set, the non-singular solution space significantly expands the range of potential solutions and this is why it is preferred to singular solution spaces.

— Figure 3 —

As mentioned above, SPAN explores solutions in both singular and non-singular solution spaces. It's core selection component is run first, followed by the construction of source trees and core trees. The core selection procedure starts by creating a pool of potential cores, whereby a node $n$ is a potential core only if there exists a source-receiver pair $(s, r)$ such that $r$ is dominated by node $n$ for source $s$. Each candidate core reports its domination information to the node designated as session coordinator, which then runs the core selection algorithm. Additional terminology is necessary to describe this procedure. We denote the core tree rooted at core $c$ to serve source $s$ for all receivers in $D(c, s)$ as $T_{cs}$. Also, we say that $T_{cs}$ serves $s$ *totally* and we call $s$ its defining source. We say that $T_{cs}$ serves source $s'$ *partially* if $D(c, s) \cap D(c, s') \neq \emptyset$ and $D(c, s) \neq D(c, s')$. Cores are selected based on the higher *domination count*, where a domination count of a core-source tuple $(c, s)$ is defined as:

$$DominationCount(c, s) = \sum_{s'\in S} |D(c, s) \cap D(c, s')|$$

The domination count is simply the number of source-receiver pairs that $T_{cs}$ is capable to serve when the receiver set is limited to $D(c,s)$. A high domination count means high utilization of $T_{cs}$ in terms of number of sources it can serve partially or totally. It increases the link sharing across the sources and reduces the hop-count, thus improving the overall multipoint path structure. SPAN iterates to select a core-source pair with the highest domination count until the entire group is served by the selected trees. Algorithm 1 shows a simplified pseudo-code of core-selection component of SPAN.

The tree construction part of SPAN is run on each source and on each core which acts as the tree root. Source trees are obtained by connecting each source with relevant cores along the shortest paths, thus leaving maximum delay-residue for the delay bounds on the core-trees. The delay bound is computed for each receiver to be spanned on the core-trees within the core selection part of the framework in the following manner. The $T_{cs}$ serves totally the $s$ source and serves partially sources from some set $S'$. Since source-based trees connect sources with the respective cores (leaves of the source-based trees) along the shortest-delay paths, the delay bounds on the leafs (receivers) of the $T_{cs}$ are computed as the difference between the delay bound of the application and the maximum of the delays along the shortest delay paths between sources from $S'$ and $c$. The delay bound for each leaf of the core-based tree is computed individually.

1. find the set of $candidateCores$
2. $coresSelected = \emptyset$
3. compute domination counts

**Repeat**

4. select the core tree $T_{cs}$ with the highest domination count
5. $coresSelected = coresSelected \cup \{c\}$
6. $candidateCores = candidateCores \backslash \{c\}$
7. update delay bounds of each leaf of the $T_{cs}$ tree (since $T_{cs}$ is serving some sources partially)
8. update the domination status for the selected core tree (exclude recently dominated $r_s$ from the domination sets of the current candidate cores)
9. update domination counts accordingly
10. notify $c$ to construct the specified core tree

**Until** all the receivers are dominated for all sources

11. notify each source to construct the source tree

**Algorithm 1.** Core-selection component of SPAN

Core-trees are constructed by using incremental SMT heuristics [19] modified for construction of constrained trees. The SMT heuristic can operate on local information available at the domain nodes and is therefore feasible for distributed implementation. It is used for construction of both source-rooted and core-rooted trees. It produces the required delay-bound core-based trees with the input information consisting of the root of the tree, the set of leaves to be spanned and the delay-bounds to be met for each leaf. For core-rooted trees the root is the core, the set of leaves is the corresponding domination set $D(c,s)$ and the delay bounds are computed within the core-selection component of the framework. Source based trees are built along the shortest delay paths between the source and the

leaves, where the tree is the source and the set of leaves is the set of cores which serve that source for at least one receiver. Figure 4 shows the sequence of core selection and tree construction algorithms [10].

— Figure 4 —

Algorithm 2 shows a pseudo-code of the tree-construction component of SPAN.

1. $T = \{root\}$

**Repeat**

2. Identify an unconnected $leaf$ that is closest to the tree

**If** within delay bound **then**

3. connect $leaf$ to $T$ using shortest cost path

**Else** connect it to $T$ along the shortest delay path

**Until** all leaves are connected to $T$

**Algorithm 2.** Tree-construction component of SPAN

The details on these two algorithms, as well as their distributed implementation, can be found in [10]. In terms of cost and QoS efficiency, the SPAN framework consistently outperforms the alternative procedures reported in the literature. We now proceed to the description of the algorithms that we have developed.

## 3. GRASP-Based Algorithms

We have embedded the SPAN framework in a GRASP metaheuristic [6]. GRASP is an iterative procedure for combinatorial optimization problems that in each iteration constructs a solution that is then used as starting point for a local search. The construction is semi-greedy, based on controlled randomization and a greedy function that depends on the context. In the original GRASP proposal, the iterations are independent. That is, each construction is independent from the others and the local searches are not linked. Variants include memory structures and searches that create dependencies between GRASP iterations. Our proposals retain the same architectural design of SPAN in terms of separating the core selection from the tree construction process. We build several variants with the combination of basic components.

The core-selection component of our procedure is obtained by applying GRASP principals to the core-selection of SPAN. In order to implement GRASP, we must be able to compare solutions according to their total cost and feasibility (i.e., meeting the delay bound of the application). Our algorithm is designed to operate in a distributed environment, namely in a unicast routing platform, where only local-distance information is available to the routers. This information includes the length of the minimum-distance paths —in different metrics such as cost, delay and hop count— from a router to any other router in the domain as well as the next hop node on that minimum-distance path. This means that the total cost of the multipoint path associated with a complete solution is not available until it is actually constructed. Note that only centralized environments allow for the accurate computation of

the induced total cost without actually building the associated trees. Since the construction of a multipoint path for every candidate solution would be computationally too expensive and would produce large message overhead, we develop several total cost estimations which use available local information only and do not require building a tree in order to assess and compare solutions.

Let $L(s)$ denote the set of all leaves of the source tree rooted at $s$, i.e. the set of all cores that dominate at least one receiver for source $s$. Let $cost_{delay}(s,c)$, $s \in S$, $c \in L(s)$, denote the cost of the minimum delay path from $s$ to $c$. Furthermore, for each $r \in D(c,s)$, $s \in S$, $c \in L(s)$, let

$$cost_{adjusted}(c,r) = \begin{cases} \text{length of the minimum-cost path from } c \text{ to } r, \text{ if delay bound met,} \\ \text{length of the minimum-delay path from } c \text{ to } r, \text{ otherwise} \end{cases}$$

By using local information sent to the coordinator node, after the completion of the core-selection algorithm it is possible to compute the values of the following functions as the basis for estimating the total cost of the resulting multipoint path, where $D(c,s)$ is the domination set associated with the solution:

$$cost_1 = \sum_{s \in S} \sum_{c \in L(s)} \left( cost_{delay}(s,c) + \frac{\sum_{r \in D(c,s)} cost_{adjusted}(c,r)}{|D(c,s)|} \right)$$

$$cost_2 = \sum_{s \in S} \sum_{c \in L(s)} \left( cost_{delay}(s,c) + \sum_{r \in D(c,s)} cost_{adjusted}(c,r) \right)$$

$$cost_3 = \frac{cost_2}{|R|}$$

In order to allow greater delay residue for the core tree leafs, source trees are constructed by building paths along the minimum-delay paths from source to corresponding cores from the $L(s)$ set. Since the lengths of such paths are contained in the local-distance information available to the router, it is possible to compute the exact cost of each source tree. The source tree rooted at source $s$ spans receivers from $L(s)$ and its cost is given by $\sum_{c \in L(s)} cost_{delay}(s,c)$. The cost of all source trees in the multipoint path is given by $\sum_{s \in S} \sum_{c \in L(s)} cost_{delay}(s,c)$. It remains to estimate the costs of core clusters associated with each source. Let us consider the core tree defined by source $s$ and core $c \in L(s)$. Its leaf set consists of receivers from $D(c,s)$. We have to estimate its cost by using local-distance information only. For each core-leaf pair we know the cost of the minimum-delay path and the cost of the minimum-cost path that connects them. Though the actual core tree is not necessarily being built along one of these two shortest paths, their associated costs may be used as estimates. If the minimum-cost path between core $c$ and leaf $r$ satisfies the delay-bound, we use its cost to estimate the cost of the portion of the core tree connecting $c$ and $r$. If the path does not satisfy the bound then we estimate the cost with the one associated with the minimum-delay path. This matches the definition of $cost_{adjusted}(c,r)$.

The value of $cost_1$ is the sum of the exact costs of all the source trees and a cost estimate of all the core trees. The average cost of a core tree is estimated as $\frac{\sum_{r \in D(c,s)} cost_{adjusted}(c,r)}{|D(c,s)|}$. That is, the average is the sum of estimated costs of all core-leaf pairs, $\sum_{r \in D(c,s)} cost_{adjusted}(c,r)$, divided by number of leafs,

$|D(c,s)|$. When adding over all core-source pairs $(c,s)$, $c \in L(s)$, we obtain the estimated cost of all core trees. Similarly, $cost_2$ estimates the overall multipoint path cost as the sum of the exact cost of all the source trees and the estimated cost of all the core-trees, where the cost estimate for a core-tree is obtained as sum of the estimated core-leaf costs for each core-leaf pair in the tree. $cost_3$ is obtained by dividing $cost_2$ by the number of receivers in the group.

Any of these cost estimates can serve as a criterion for comparing solutions within our GRASP implementation. Let $i_{max}$ be the limit on the number of GRASP iterations, each consisting of selecting the set of cores and estimating the cost of the corresponding multipoint path with one of the three cost estimates described above. The output of the procedure is the set of cores upon which the tree-construction component is then executed. Let $core$ denote the set of cores chosen in a given GRASP iteration and let $bestCore$ denote the core set with the best estimated cost. Algorithm 3 shows the GRASP procedure for the core selection problem.

1. $bestCore = \emptyset$

**Repeat**

2. Construct $core$
3. Improve $core$ with local search

**If** the estimated cost of $core$ is better than the estimated cost of $bestCore$ **then**

4. Make $bestCore$ equal to $core$

**Until** number of iterations equals $i_{max}$

**Algorithm 3.** Outline of GRASP for core selection

Step 2 of Algorithm 3 is performed following a semi-greedy scheme, typical of all GRASP implementations. The criterion for selecting cores is the same as in SPAN, that is, preference is given to core candidates for which their domination count is large. The pseudo code for the construction step is shown as Algorithm 4, which assumes that $LB$ and $UB$ are the lower and upper bounds on the domination count over the set of candidate cores for source-receiver pairs that have not been covered by previous core selections.

1. Randomly generate $\alpha = [0,1]$
2. $core = \emptyset$

**Repeat**

3. Construct $candidates = \{(c,s) | UB \geq DominationCount(c,s) \geq UB - \alpha(UB - LB)\}$
4. Randomly select $(c,s) \in candidates$
5. $core = core \cup \{c\}$
6. Update domination sets, domination counts, delays, leaf sets and information relative to core-tree and source-tree construction as in SPAN

**Until** all receivers from all sources are covered

**Algorithm 4.** Semi-greedy $core$ construction (step 2 of Algorithm 3)

The algorithm starts by choosing a random number $\alpha$ between 0 and 1. Initially, the set of selected cores is empty (step 2). The set of candidate cores is determined in step 3. A node $c$ is a candidate if

there exists source $s$ such that $c$ completely serves $s$ and the corresponding domination count $DominationCount(c,s)$ is within $[UB-\alpha(UB-LB), UB]$. Unlike SPAN that chooses the candidate with the largest domination count (i.e., the one corresponding to $UB$), we select one whose domination count is in the top $\alpha\%$ of all candidates. The core tree for a selected node $c$ to be added to $core$ spans the receivers in $D(c,s)$, where $s$ is completely served by $c$. The core tree coincides with $T_{cs}$. Since $c$ may partially serve other sources, the delay bounds on the leaf set $T_{cs}$ must be updated. Also, for each source $s'$ partially served by $c$ and $T_{cs}$, we must update $L(s')$, which is the set of leafs for the source based tree rooted at $s'$. We update domination sets and domination counts by excluding pairs of sources and receivers just covered by the last core selection. We also update all domination sets $D(c,s)$ to keep track of the cores that serve receivers for each source and therefore recover a solution at the end of the construction step.

The local search is loosely based on VNS (variable neighbourhood search) principles (see [13] and [7]). In our implementation, neighbourhoods of different complexity are explored, where the size is determined by the number of receivers that are reassigned at a time. Thus, the $k^{th}$ neighbourhood $N_k(T)$ of solution $T$ (representing the multipath along which sources serve receivers) is obtained by moving $k$ receivers from their current core to a different core within the set of selected cores. The number of neighbourhoods to explore is determined by the parameter $k_{max}$. Algorithm 5 shows the pseudo-code of the local search that corresponds to step 3 of Algorithm 3.

1. Set initial solution as $T$, $k=1$ and $noImprove=0$

**Repeat**

2. Obtain by random sampling $T' \in N_k(T)$

**If** $T'$ is better than $T$ **then**

3. Set $T=T'$ and update
4. $k=1$ and $noImprove=0$

**else**

5. $noImprove = noImprove + 1$

**If** $noImprove > maxNoImprove$ **then**

6. $k=k+1$
7. $noImprove=0$

**While** $k < k_{max}$

8. Remove cores that are not connected to any source

**Algorithm 5.** Local search (step 3 of Algorithm 3)

The search initiates from the solution constructed in step 2 of Algorithm 3 and proceeds to explore neighborhoods sequentially, starting from $N_1$. The exploration consists of randomly choosing $k$ source-receiver pairs and constructing a neighbor by reconnecting them through different cores. The reconnection is done in such a way that the $cost_{adjusted}$ is minimized. The resulting neighbor solution $T'$ is compared with the current solution $T$ on the basis of their cost, which is estimated with $cost_1$, $cost_2$ or $cost_3$. If $T'$ has a lower cost than $T$ then it becomes the new current solution and the neighborhood search is reset to $N_1$. Also, the variable counting the number of iterations without improvement, $noImprove$, is reset to zero. If, on the other hand, the chosen neighbor is not better than the current solution, the $noImprove$ counter is increased. The exploration remains in the same neighborhood until the $maxNoImprove$ limit is reached. At this point, the search moves to a larger

neighborhood and the process continues. The local search ends when the largest neighborhood is reached and no improvement at that level is possible within the specified search parameters. (Note that the process may also terminate if the execution time limit is reached at any time during the exploration.)

As in SPAN, the source-trees are obtained by connecting the root with each leaf along the minimum-cost path. In addition to using the SMT heuristic shown as Algorithm 2 for the core-tree construction component, we also test a slight modification that we refer to as SMT-reverse. The only difference between SMT and SMT-reverse is that, in step 2 of Algorithm 2, SMT-reverse chooses the unconnected leaf that is farthest away from the existing tree instead of choosing the closest leaf, as SMT does. The rationale behind reversing what SMT does is that faraway leafs are the most critical with respect to the delay bound, so if they are the first ones to be connected to the tree the chance increases that the minimum-cost path between that leaf and the tree will satisfy the delay bound of the application. In that way, SMT-reverse might achieve a low cost while meeting the delay bound.

In the next section, we present the computational experiments associated with 18 different versions of solutions procedures obtained by combining the components that we have described above. The core selection based on GRASP may be configured in 9 different ways by choosing from three costs estimates (i.e., $cost_1$, $cost_2$ and $cost_3$) and three local searches: none, VNS and SPAN/ADJUST (as proposed by Karaman and Hassanein in [11]). These core-selection procedures are then combined with two ways of constructing the corresponding trees: SMT and SMT-reverse. This gives a total of 18 variants.

## 4. Computational Experiments

We have tested the performance of our proposed heuristics against GREEDY [16], the most prominent algorithm for the QoS-constrained many-to-many problem in question, as well as against SPAN [10] and its extensions SPAN/COST and SPAN/ADJUST [11], which to the best of our knowledge offer the best performance in terms of cost metrics known in the literature. The test network was created with the *nem* network topology generator [12] and Waxman model [22]. We have created 100 domains of size 60. The average node degree within the domains was taken from the interval [3, 5], with a mean average node degrees equaling 4.085. We have tested the performance of the algorithms on five groups of problem types with senders and receivers sparsely distributed throughout the network. In the first type, the number of receivers is fixed to 16, with the number of senders set to 8, 10, 12, 14 and 18. The delay bound is set to the critical delay

$$delay_{critial} = \max_{s \in S, r \in R} \{d(s,r)\}$$

where $d(s,r)$ is the minimum delay-distance between nodes $s$ and $r$. In other words, critical delay is the smallest possible delay allowed for which it is possible to establish the group communication satisfying that delay bound. In the second type, both the number of receivers (4, 8, 12,16, 20 and 24) and the number of senders (2, 4, 6, 8, 10 and 12) vary, with the ratio of senders to receivers fixed at 1:2. The delay bound is also set to the critical delay. Furthermore, the second type is divided into two groups. In the first one, exactly half of the sources are also receivers, while in the second one all sources are also receivers. In the third type, the delay bound is set to the critical delay and the senders are a

subset of the receivers. The third type also has two groups. In the first one, there are 12 receivers and 2, 4, 6, 8, 10 and 12 senders. In the second one, there are 24 receivers and 4, 8, 12, 16, 20 and 24 senders. The fourth type has 4, 6, 8, 10, 12 and 14 receivers and 2, 3, 4, 5, 6 and 7 senders, there is no intersection between the two sets and there is a ratio of senders to receivers of 1:2. The delay bound is the critical delay in the fourth group of instances. Finally, in the fifth type, the delay bound varies as follows

$$delay_{\beta} = delay_{critical} + \beta(delay_{max} - delay_{critical})$$

with $\beta = 0.0,0.2,0.4,0.6,0.8,1.0$ and

$$delay_{max} = \max_{s \in S, r \in R, c \in C} \{d(s,c) + d(c,r)\}$$

where $delay_{max}$ is the delay bound for which any transmission will satisfy the delay bound of the application. The first group of this type has 16 receivers and 4 senders with no intersection between the two sets. The second group has also 16 receivers but 8 senders, of whom 4 are also receivers.

Table 1 shows a summary of the types and groups of problem instances that we generated. The "overlap" refers to the percentage of senders that are also receivers. The delay bound is either "critical" (i.e., given by $delay_{critical}$) or "β" (i.e., given by $delay_{\beta}$). Since we generated 100 instances for each combination of receiver set's and senders set's sizes within each type of problem, the total number of instances is 3,700.

| Type | Instances | (Receivers, Senders) | Overlap | Delay Bound |
|---|---|---|---|---|
| 1 | 500 | (16, 8) (16, 10) (16, 12) (16, 18) | | Critical |
| 2 (a) | 600 | (4, 2) (8, 4) (12, 6) (16, 8) (20, 10) (24, 12) | 50% | Critical |
| 2 (b) | 600 | (4, 2) (8, 4) (12, 6) (16, 8) (20, 10) (24, 12) | 100% | Critical |
| 3 (a) | 600 | (12, 2) (12, 4) (12, 6) (12, 8) (12, 10) (12, 12) | 100% | Critical |
| 3 (b) | 600 | (24, 2) (24, 4) (24, 6) (24, 8) (24, 10) (24, 12) | 100% | Critical |
| 4 | 600 | (4, 2) (6, 3) (8, 4) (10, 5) (12, 6) (14, 7) | | Critical |
| 5 (a) | 100 | (16, 4) | | β |
| 5 (b) | 100 | (16, 8) | 50% | β |

**Table 1.** Summary of characteristics of problem instances used for experimentation

In the first set of experiments, we simply replace SMT with SMT-reverse and test the existing procedures SPAN, SPAN/COST and SPAN/ADJUST with this modification. We compare their performance with the original procedures (i.e., using SMT) and with GREEDY. For ease of comparison, we normalize all values to the results obtained by SPAN, which is set to 1. Tables 2 and 3 show a summary of results by problem type, where the "R" represents the modified version of the original procedures in which we have replaced SMT with SMT-reverse. Table 2 shows average cost (relative to SPAN) and Table 3 shows average hop count (relative to SPAN).

The results show that the reverse versions of the SPAN-based algorithms consistently outperform their SPAN counterparts in terms of cost. However, the improvement in cost is achieved at the price of

a higher hop-count. In general, the best results for all the test groups in terms of cost were achieved by SPAN/COST-R, followed by SPAN/ADJUST-R. Therefore, just by changing the way the trees are being built we have achieved improvements over the existing algorithms. In general, the cost improvement was of about 10%. Because of the way the trees are being built, the most significant improvement, up to 45%, was achieved in test group 5 in which the delay bounds are relatively loose, allowing shortest cost paths to be utilized.

| Type | SPAN COST | SPAN ADJUST | SPAN | SPAN COST-R | SPAN ADJUST-R | GREEDY |
|---|---|---|---|---|---|---|
| 1 | 0.973 | 1 | 1 | 0.879 | 0.909 | 1.342 |
| 2 (a) | 0.983 | 1 | 1 | 0.891 | 0.922 | 1.347 |
| 2 (b) | 0.994 | 1 | 1 | 0.898 | 0.911 | 1.342 |
| 3 (a) | 0.969 | 1 | 1 | 0.929 | 0.973 | 1.381 |
| 3 (b) | 0.996 | 1 | 1 | 0.884 | 0.906 | 1.361 |
| 4 | 0.945 | 1 | 1 | 0.920 | 0.981 | 1.308 |
| 5 (a) | 0.924 | 1 | 1 | 0.726 | 0.779 | 1.402 |
| 5 (b) | 0.921 | 1 | 1 | 0.686 | 0.782 | 1.554 |
| Avg. | 0.963 | 1 | 1 | 0.852 | 0.895 | 1.380 |

**Table 2.** Summary of cost relative to SPAN

| Type | SPAN COST | SPAN ADJUST | SPAN | SPAN COST-R | SPAN ADJUST-R | GREEDY |
|---|---|---|---|---|---|---|
| 1 | 1.149 | 1 | 1 | 1.194 | 1.046 | 1.355 |
| 2 (a) | 1.115 | 1 | 1 | 1.161 | 1.050 | 1.322 |
| 2 (b) | 1.149 | 1 | 1 | 1.196 | 1.047 | 1.321 |
| 3 (a) | 1.194 | 1 | 1 | 1.229 | 1.042 | 1.307 |
| 3 (b) | 1.140 | 1 | 1 | 1.192 | 1.045 | 1.343 |
| 4 | 1.085 | 1 | 1 | 1.117 | 1.043 | 1.259 |
| 5 (a) | 1.032 | 1.001 | 1 | 1.148 | 1.126 | 1.131 |
| 5 (b) | 1.013 | 1 | 1 | 1.136 | 1.126 | 1.124 |
| Avg. | 1.110 | 1 | 1 | 1.172 | 1.066 | 1.270 |

**Table 3.** Summary of hop count relative to SPAN

The second group of algorithms was created by combining —within the GRASP framework— the building blocks associated with core selection and tree construction. As mentioned above, for the core selection, we consider three choices of cost estimates and three choices of local search and we combine them with two tree constructions, resulting in 18 different procedures. The procedures are identified as $GRASP/x/y/z$, where $x = 1,2,3$ (corresponding to cost1, cost2 or cost3), $y = N, A, V$ (corresponding to no local search, SPAN/ADJUST and VNS) and $z = S, R$ (corresponding to SMT and SMT-reverse). Since all of these algorithmic variants include random elements, for these set of tests, each instance within a problem type was run 20 times, resulting in a total of 20*3,700 = 74,000 runs.

Statistical pairwise comparisons of these 18 variants indicate that the best performance in terms of cost (when considering all problem types) is achieved by $GRASP/3/V/R$. However, these results are found at a computational cost that is 23 times of the one required by SPAN. Since this computational effort is not practical, we turned our attention to $GRASP/3/N/R$ and $GRASP/3/A/R$, which our statistical tests showed to be the best among all of those variants that do not use the VNS-based local search. Table 4 shows a summary of results associated with these procedures. For each procedure and problem type, the table shows the average cost normalized to the SPAN results (see column labeled Cost). It also shows the percentage of times (out of 20 runs) that the GRASP variants improved upon the solutions found by SPAN/COST-R and SPAN/ADJUST-R, which proved to be the best among those tested in our previous experiment. Those percentages appear in the columns labeled SPAN/COST-R and SPAN/ADJUST-R under each of the GRASP variants.

| Type | $GRASP/3/N/R$ | | | $GRASP/3/A/R$ | | |
|---|---|---|---|---|---|---|
| | Cost | SPAN COST-R | SPAN ADJUST-R | Cost | SPAN COST-R | SPAN ADJUST-R |
| 1 | 0.668 | 87.75% | 88.92% | 0.683 | 84.85% | 86.02% |
| 2 (a) | 0.692 | 86.97% | 87.63% | 0.706 | 83.23% | 84.61% |
| 2 (b) | 0.693 | 89.11% | 87.76% | 0.707 | 85.23% | 84.94% |
| 3 (a) | 0.722 | 84.94% | 86.28% | 0.721 | 84.92% | 86.51% |
| 3 (b) | 0.686 | 88.82% | 88.38% | 0.701 | 84.54% | 84.93% |
| 4 | 0.706 | 84.91% | 88.91% | 0.705 | 85.00% | 89.06% |
| 5 (a) | 0.657 | 66.01% | 85.31% | 0.664 | 64.27% | 84.31% |
| 5 (b) | 0.704 | 22.81% | 73.53% | 0.708 | 21.88% | 72.32% |
| Avg. | 0.691 | 76.42% | 85.84% | 0.699 | 74.24% | 84.09% |

**Table 4.** Performance comparison between GRASP variants and SPAN variants

The results in Table 4 show that on average, the GRASP variants outperform SPAN. On average, the solutions found by GRASP variants are less than 0.7 of the cost of the solutions found by SPAN. This compares well with the average of 0.852 and 0.895 corresponding SPAN/COST-R and SPAN/ADJUST-R, respectively (see Table 2). For all problem types except 5 (b), the best results were obtained with $GRASP/3/N/R$, with $GRASP/3/A/R$ a close second. Due to the relaxed delay bounds, the best results for 5 (b) instances were obtained by SPAN/COST-R, with an average normalized cost of 0.686. Note that only about 22.8% of the solutions found by the GRASP variants were better than the solutions found by SPAN/COST-R in this set of problem. The performance of $GRASP/3/N/R$ in problem sets 1 to 4 is very robust, with about 85% or more solutions being strictly better than SPAN/COST-R.

We analyzed $GRASP/3/N/R$ further in order to gain additional understanding of its behavior and the effect of the random elements within the procedure. Figure 5 shows box plots that summarize the distribution of costs for each type of problem. The upper edge of the vertical lines in the graph denote the maximum cost value attained, the upper box lines denote the third quartile of the distribution, the lower box lines denote the first quartile of the distribution, while the lower edge of the vertical lines denote the minimum cost value attained. The results were normalized by the cost performance of SPAN. Also, the boxes include the median of the distributions.

— Figure 5 —

The box plots show that, for most data sets, three quarters of all solutions (i.e., the third quartile) obtained by $GRASP/3/N/R$ have an objective function value that is 0.8 of the SPAN objective function value. The best results show cost improvements of more than 80% compared to SPAN and only in a couple of instances (set 3(a) and 4) the cost was significantly larger than the SPAN cost. The fairly narrow width of the boxes across all data sets shows the robustness of the procedure in the sense that it is quite predictable the amount of cost savings that could expected over the benchmark cost determined by SPAN. In terms of computing time, $GRASP/3/N/R$ runs are completed, on the average, in as little as a fraction of a second and as much as 5 seconds (on an Intel Xeon CPU X5670 @ 2.93 GHz and 3GB of RAM). Although the longest GRASP execution times are an order of magnitude larger than the SPAN times, they are still within the time required in practice. We are aware, however, that additional efficiencies are possible with the implementation of advanced data structures and strategic use of memory to minimize the number of calculations and updates from iteration to iteration.

## 5. Conclusions

We have presented several new core-based algorithms for the multicast routing under a stringent end-to-end delay constraint which operate in a distributed, asymmetric environment. We have suggested a modification of the standard tree-construction algorithm known as SMT and tested it by incorporating it in the tree construction module of SPAN, SPAN/COST and SPAN/ADJUST. On average, the proposed modification (SMT-reverse) improves the performance of the existing algorithms by 10% (in terms of cost). The greatest improvement, up to 45%, was achieved for the case of relaxed delay bounds. A new set of solution procedures was developed around a framework based on GRASP principles. Altogether we created 18 new variants by combining building block associated with selection of cores and construction of trees. Our extensive experimentation shows that the best variant of our proposal is a significant improvement over the methods reported in the literature for the problem of interest.

## References

[1] Ballardie, A. (1997) “Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification,” RFC 2198, September 1997.

[2] Ballardie, A. (1997) “Core Based Trees (CBT) Multicast Routing Architecture,” RFC 2201, September 1997.

[3] Deering, S., D. Estrin, D. Farinacci and V. Jacobson (1994) “Protocol Independent Multicasting (PIM), Dense Mode Protocol Specification,” Technical Report, IETF-IDMR.

[4] Deering, S., D. L. Estrin, D. Farinacci, V. Jacobson, C. Liu and L. Wei (1996) “The PIM Architecture for Wide-area Multicast Routing,” *IEEE/ACM Transactions on Networking*, vol. 4, no. 2, pp. 153–162.

[5] Feo, T. A. and M. G. C. Resende (1989) “A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem,” *Operations Research Letters*, vol. 8, pp. 67-71.

[6] Feo, T. A. and M. G. C. Resende (1995) "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, pp. 109-133.

[7] Hansen P. and N. Mladenović (2001) "Variable Neighborhood Search: Principles and Applications," *European Journal of Operational Research*, vol. 130, pp. 449–467.

[8] ITU-T Recommendation G.114 (2003) "One-way transmission time", International Telecommunication Union, Geneva, Switzerland.

[9] Karaman, A. and H. Hassanein (2006) "Core Selection Algorithms in Multicast Routing: Comparative and Complexity Analysis," *Computer Communications*, vol. 29, pp. 990-1014.

[10] Karaman, A. and H. Hassanein (2007) "An Extended Core-based Framework for Delay-constrained Group Communication," *International Journal of Communication Systems*, vol. 20, pp. 1179-1213.

[11] Karaman, A. and H. Hassanein (2007) "QoS-Constrained Core Selection for Group Communication," *Computer Communication*, vol. 30, pp. 1600-1612.

[12] D. Magoni (2002) "*nem*: A Software for Network Topology Analysis and Modeling," *Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems* (MASCOTS.02), pp.1526-7539.

[13] Mladenović, N. and P. Hansen (1997) "Variable Neighborhood Search," Computers and Operations Research, vol. 24, pp. 1097–1100.

[14] Moy, J. (1994) "Multicast Extensions to OSPF", RFC 1584, International Engineering Task Force.

[15] Moy, J. (a994) "Multicast Routing Extensions for OSPF," *Communications of the ACM*, vol. 37, no. 8, pp. 61–114.

[16] Salama, H. F. (1996) "Multicast Routing for Real-time Communication on High-speed Networks," Ph.D., Thesis, Department of Electrical and Computer Engineering, North Caroline State University.

[17] Shields, C. (1996) "Ordered Core Based Trees," MSc Thesis, University of California, Santa Cruz.

[18] Shields, C. and J. J. Garcia-Luna-Acevez (1997) "The Ordered Core-based Tree Protocol," Proceedings of INFOCOM'97, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 884-891.

[19] Takahashi, H. and A. Matsuyama (1980) "An Approximate Solution for the Steiner Problem in Graphs," *Mathematica Japonica*, vol. 24, pp. 573–577.

[20] Waitzman, D., C. Partridge and S. Deering (1988) "Distance Vector Multicast Routing Protocol", RFC 1075, International Engineering Task Force.

[21] Wei, L. and D. Estrin (1994) "The Trade-offs of Multicast Trees and Algorithms," Proceedings of ICCCN'94, San Francisco, CA, USA, Sep. 1994, pp. 902-926.

[22] Waxman, B. N. (1988) "Routing of Multipoint Connections," *IEEE Journal of Selected Areas in Communication*, vol. 6, no. 9, pp. 1617–1622.

[23] Zappala, D., A. Fabbri, and V. Lo (2002) "An Evaluation of Shared Multicast Trees with Multiple Cores," *Journal of Telecommunication Systems*, vol. 19, no. 3, pp. 461-479.

## Appendix

The multicasting problem for a multi-source communication group may be formulated as mixed-integer program. This is a generalization of the single-source multiple-receiver formulation in order to include multiple receivers. The objective function may be to minimize total cost, to minimize the maximum end-to-end delay or to minimize hop-count.

The problem has the following parameters:

| | | |
|---|---|---|
| $N$ | : | set of nodes |
| $E$ | : | set of links |
| $S$ | : | set of senders |
| $R$ | : | set of receivers |
| $c_{ij}$ | : | cost of link $(i,j)$ |
| $d_{ij}$ | : | delay of link $(i,j)$ |
| $b_{ij}$ | : | bandwidth of link $(i,j)$ |
| $d_{max}$ | : | delay bound |

The decision variables are as follows:

$x_{ij}^{sr} = \begin{cases} 1; & \text{if link } (i,j) \text{ is used to connect sender } s \text{ with receiver } r \\ 0; & \text{otherwise} \end{cases}$

Note that by definition $x_{ij}^{sr} = 0$ if $s = r$ and therefore those variables are not defined.

$z_{ij}^{s} = \begin{cases} 1; & \text{if link } (i,j) \text{ is included in the source tree rooted at } s \\ 0; & \text{otherwise} \end{cases}$

The cost minimization model is:

$$\min \sum_{(i,j)\in E} \sum_{s\in S} c_{ij} z_{ij}^{s}$$

Subject to

Conservation of flow at the sources

$$\sum_{j:(s,j)\in E} x_{sj}^{sr} - \sum_{i:(i,s)\in E} x_{is}^{sr} = 1 \qquad \forall s \in S, r \in R$$

Conservation of flow at intermediate nodes

$$\sum_{j:(i,j)\in E} x_{ij}^{sr} - \sum_{j:(j,i)\in E} x_{ji}^{sr} = 0 \qquad \forall s \in S, r \in R, i \in N\backslash\{s,r\}$$

Conservation of flow at the receivers

$$\sum_{j:(r,j)\in E} x_{rj}^{sr} - \sum_{i:(i,r)\in E} x_{ir}^{sr} = -1 \qquad \forall s \in S, r \in R$$

Delay bound

$$\sum_{(i,j)\in E} d_{ij} x_{ij}^{sr} \leq d_{max} \qquad \forall s \in S, \forall r \in R$$

Source tree structure

$$x_{ij}^{sr} \leq z_{ij}^{s} \qquad \forall (i,j) \in E, s \in S, r \in$$

$$\sum_{r\in R} x_{ij}^{sr} - z_{ij}^{s} \geq 0 \qquad \forall s \in S, (i,j) \in E$$

Variable restrictions

$$x_{ij}^{sr} \in \{0,1\}, z_{ij}^{s} \in \{0,1\} \qquad \forall (i,j) \in E, s \in S, r \in R$$

To formulate a hop-count minimization model, the following binary variables are defined:

$$y_{ij} = \begin{cases} 1; & \text{if link } (i,j) \text{ is included in the solution} \\ 0; & \text{otherwise} \end{cases}$$

The objective function is changed to:

$$\min \sum_{(i,j)\in E} y_{ij}$$

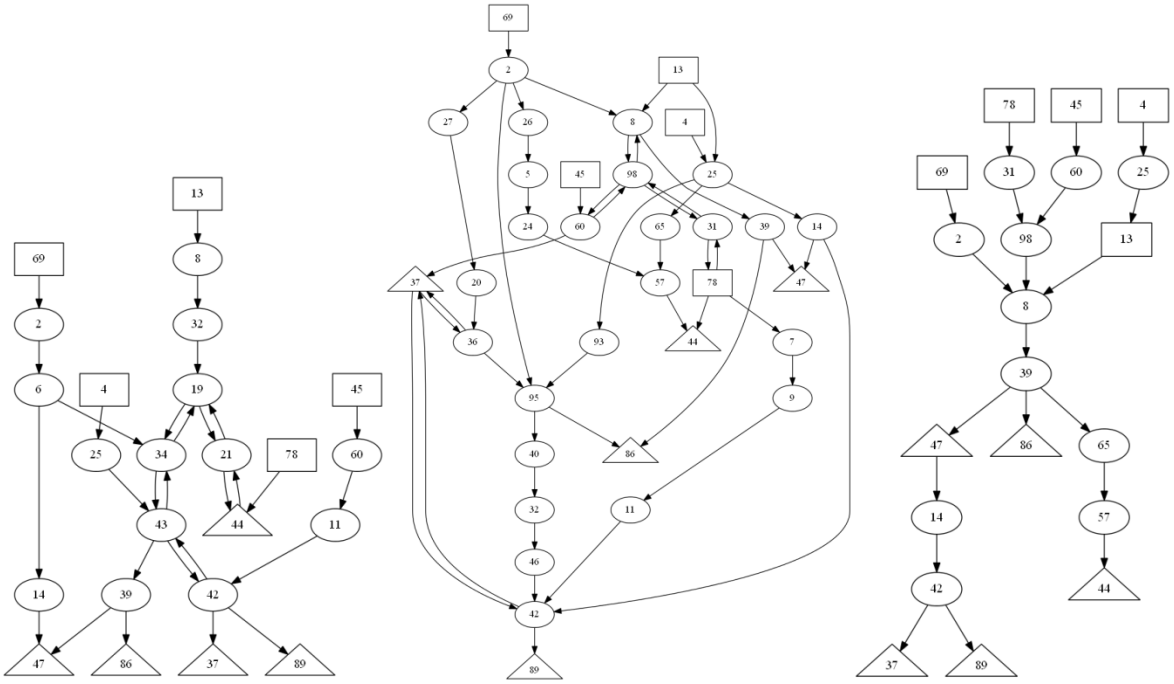
And the following constraints are added to the model:

$$x_{ij}^{sr} \leq y_{ij} \qquad \forall (i,j) \in E, \forall s \in S, \forall r \in R$$

$$y_{ij} \in \{0,1\} \qquad \forall (i,j) \in E$$

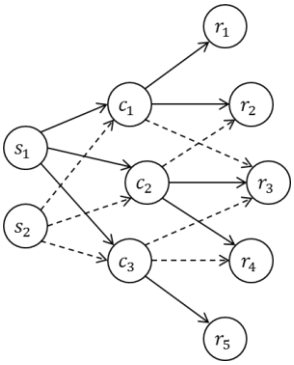
Finally, to formulate the minimization of the maximum delay, the $d_{max}$ parameter is turned into a nonnegative continuous variable and the objective function is changed to:

$$\min d_{max}$$

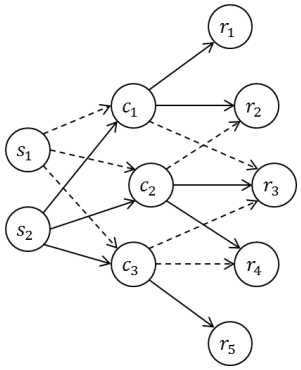



(a) (b) (c)

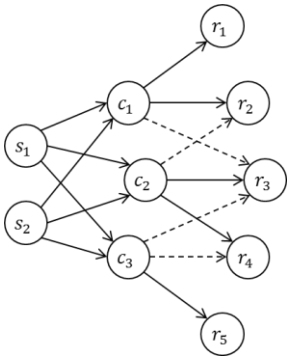
**Figure 1.** Optimal transmission for a multicast group with 5 senders and 5 receivers with the objective of minimizing a) cost, b) maximum end-to-end delay and c) hop-count




(a) tree serving receivers for source $s_1$

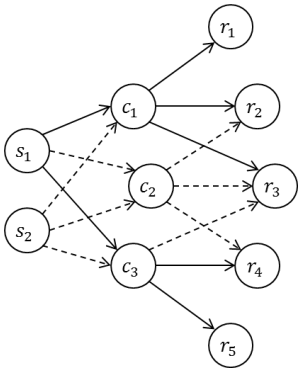



(b) tree serving all receivers for source $s_2$

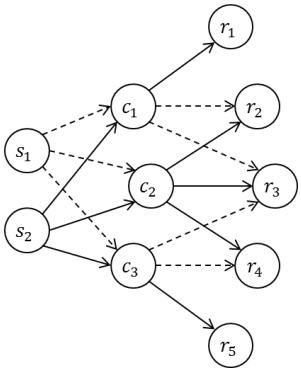



(c) combined multipoint tree

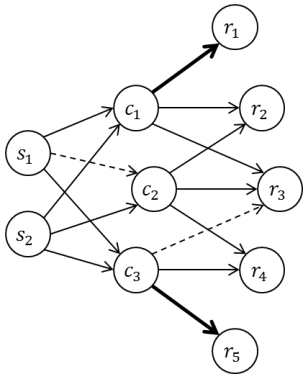
**Figure 2.** An example of a solution in a singular solution space




(a) tree serving receivers for source $s_1$




(b) tree serving all receivers for source $s_2$




(c) combined multipoint tree

**Figure 3.** An example of a solution in a non-singular solution space

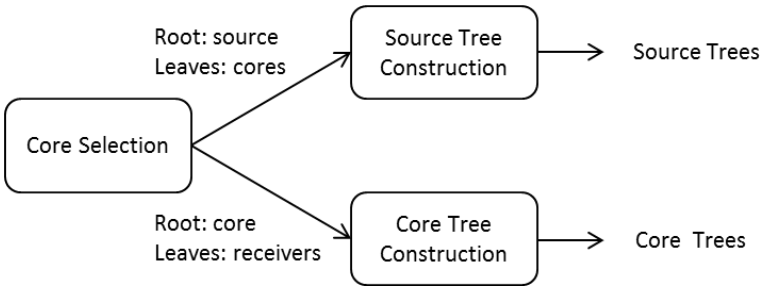



**Figure 4.** An overview of multipoint construction

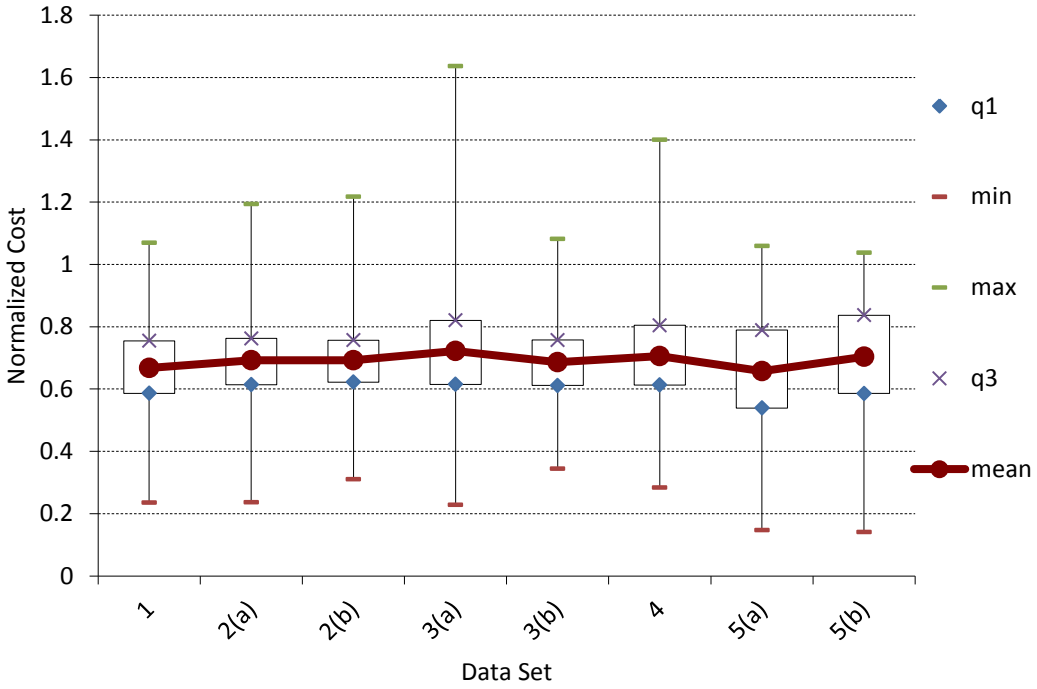



**Figure 5.** Box plot of distribution of costs obtained by $GRASP/3/N/R$