Bi-objective Multicast Packing Problem

Romerito Campos de Andrade
Universidade Federal do Rio Grande do Norte - UFRN
Campus Universitário – Lagoa Nova – 59078-970 – Natal/RN
romerito@ppgsc.ufrn.br

Marco César Goldbarg
Universidade Federal do Rio Grande do Norte - UFRN
Campus Universitário – Lagoa Nova – 59078-970 – Natal/RN
gold@dimap.ufrn.br

Elizabeth Ferreira Gouvêa Goldbarg
Universidade Federal do Rio Grande do Norte - UFRN
Campus Universitário – Lagoa Nova – 59078-970 – Natal/RN
beth@dimap.ufrn.br

ABSTRACT

This paper addresses the Multicast Packing Problem under a bi-objective viewpoint. Given a set of multicast groups in a network, the problem considered here consists in creating multicast trees, one for each group, and accommodating them in a network. Two objectives are considered simultaneously: minimizing installation cost and maximizing residual capacity. Three metaheuristic approaches are proposed to tackle the problem based on Greedy Randomized Adaptive Search Procedure, Non-dominated Sorting Genetic Algorithm 2 and Strength Pareto Evolutionary Algorithm 2. Operators to create, recombine and mutate solutions are proposed. Results of a computational experiment on eighty-nine instances are reported.

KEYWORDS. Multicast packing, Multiobjective optimization, Network optimization.

Main area: Metaheuristics.

1. Introduction

Nowadays, a lot of applications in telecommunications run simultaneously in the Internet. Some examples are Internet TV, video-conferencing, on-line games (Wu, 2005) and software delivering (Han and Shahmehri, 2000), among others. These applications share a common interest: an efficient away for point-to-multipoint communication from a source to multiple destination nodes. Basically, there are two ways to establish connections between a source and a group of destination nodes: unicast and multicast. Unicast is the simplest way to connect, but it comes at a price. The main problems are related to sending several copies of a package in the same arc of the network (considering multiple destinations). In the multicast technique such limitations are dealt with the sharing of packages reducing the number of copies. Multicast transmission plays an important role in the utilization of network resources. It must create a route to send packages from a source to destinations. There are many ways to connect nodes regarding multicast transmission, such as Steiner trees, center based trees and ring based routing (Oliveira, 2004). A comparison between the ring based routing and the Steiner tree approaches indicates that more arcs are needed in the former than in the latter to establish a multicast connection (Medina *et al*., 2001). The center based tree approach has an additional problem as it is necessary to find a center in the network and it should be near to destination nodes.

The problem called multicast routing problem considers one multicast group only. It has been addressed with one (Resende and Pardalos, 2006) or multiple criteria (Cui *et al.*, 2003) and (Xu, 2011). The installation cost is the criterion investigated by Resende and Pardalos (2006). Four objectives are considered by Cui *et al*. (2003): end-to-end delay; link utilization; packet lost; delay jitter. The idea is to create a multicast tree with QoS (Quality of Service) requirements. Xu (2011) investigates the problem with five objectives: tree cost; maximal end-to-end delay; delay jitter; average delay and link utilization.

When more than one multicast group has to be configured simultaneously in a network, the problem is called Multicast Packing Problem (MPP). In this problem, one multicast tree must be assigned to each multicast group $k \in K$. The MPP was investigated by (Chen *et al.*, 2000) with the objective of minimizing the network congestion. Lee and Cho (2004) investigated the maximization of the residual capacity. Kang *et al.*(2009) consider the optimization of the cost to configure a finite number of multicast groups in a network. Other authors have also considered the problem with the objective of cost minimization (Jia and Wang, 1997; Low and Wang, 1999; Wang *et al.*, 2002).

The MPP has not been studied under a multi-objective viewpoint. In (Chen, Gunluk, & Yener, 2000) the authors, clearly, try to consider congestion (Among all edges in the netwok, consider the edge with the most load of traffic over it as the congestion of the network) and cost, but the latter is considered as a penalty added to the objective function. Kang *et al.* (2009) consider the problem of optimizing the cost with a hop-constraint as a QoS measure.

In this paper, Steiner trees are used to represent solutions. The MPP is considered with two objectives that are treated with equal importance. The objectives are: cost and residual capacity (Lee and Cho, 2004). The objective is to minimize the installation cost and maximize the residual capacity. These objectives represent a trade-off since the minimization of the installation directs solution to using few arcs, increasing the congestion. On the other hand, the maximization of the residual capacity may increase the cost.

Three algorithms, Greedy Randomized Adaptive Search Procedure (GRASP) proposed by Feo and Resende (1995), Non-dominated Sorting Genetic Algorithm 2 (NSGA2) proposed by Deb et.al. (2002) and Strength Pareto Evolutionary Algorithm 2(SPEA2) proposed by (Zitzler, Laumanns, & Thiele, 2001), are presented for the MPP. To create these algorithms several operators are proposed: two operators to create initial solutions; one recombination operator and one mutation operator; one neighborhood and one local search procedure based on Pareto Local Search (PLS) (Paquete and Stützle, 2006). Eighty-nine instances were created to test the proposed algorithms. These algorithms were compared based on the hypervolume quality indicator (I_H) proposed in (Zitzler and Thiele, 1999). The Mann-Whitney's test was used (Conover, 1980).

Section II presents the problem formulation considering a mathematical model based on multi-objective optimization. Section III presents the solution representation and the algorithms. Section IV presents the experimental results. Finally, Section V presents conclusions.

2. Problem Formulation

Given a graph $G=(V,E)$, where V represents the set of vertexes and E the set of edges, two values are defined for each $e \in E$: the cost of using edge e in a solution, $c(e)$, and the maximum traffic supported by e, called capacity of e, $b(e)$. Consider K a set of multicast groups that must be accommodated simultaneously in the network represented by G. For each multicast group $k \in K$, s_k is the source node, D_k is the set of destination nodes and t_k is the traffic requirement of the k-th multicast group. The traffic requirement of a group is defined as the minimum capacity required of an edge of that group. A tree must be defined for each multicast group, where decision variable $x_e^k = 1$ if edge e is in the k-th multicast tree, otherwise $x_e^k = 0$. An edge e can be in more than one multicast tree. An MPP solution is a set $\mathbf{T} = \{T_1, T_2, ..., T_{|K|}\}$ of multicast trees. Consider E_T the set of edges used by the $|K|$ trees of an MPP solution, the problem consists in optimizing simultaneously the two objectives presented in equations 1 and 3. The first objective, presented in equation 1, is to minimize the installation cost. The second objective consists in maximizing the minimal residual capacity of an edge in E_T. The residual

capacity of $e \in E_T$ is defined as in Lee and Cho(2004) and presented in equation 2. The second objective function is to maximize the minimum residual capacity as presented in equation 3. The minimal residual capacity represents a bottleneck in the network. It can make the tree increase with try to maximize its value, considering the sharing the congestion of the network is made by using more edges increasing the cost. In this case, it is important maximize this value. The two objectives are considered simultaneously. The capacity of each edge cannot be violated. This constraint is expressed in equation 4.

$$f_1 = \sum_{k=1}^{|K|} \sum_{e \in E_T} c(e) x_e^k \quad (1)$$

$$z_e = b(e) - \sum_{k=1}^{|K|} t_k x_e^k \quad (2)$$

$$f_2 = \min\{z_e, \forall_e \in E_T\} \quad (3)$$

$$b(e) \geq \sum_{k=1}^{|K|} t_k x_e^k \quad (4)$$

Let V_k be the set of vertices of the *k*-th multicast tree. A bi-objective mathematical model for the MPP can be defined as:

$min\ f_1, max\ f_2$

subject to

$$b(e) \geq \sum_{k=1}^{|K|} t_k x_e^k$$

$$\{D_k \cup \{s_k\}\} \subset V_k,\ \forall\ T_k \in \mathbf{T}$$

$$x_e^k \in \{0,1\}$$

3. Algorithms

This section presents the proposed algorithms, their operators and methods. First, the solution representation is described in Section 3.1. Then, the GRASP algorithm and the evolutionary approaches are presented in Sections 3.2 and 3.3, respectively.

3.1 Solution Representation

Basically, an MPP solution consists of a list of Steiner Trees, one for each multicast group. Each Steiner Tree is represented by a list of edges. Then, the solution representation consists of a list of lists of edges.

Two heuristics were implemented to create Steiner trees. These heuristics are modified versions of others presented previously to the Steiner Tree Problem.

The first heuristic is a randomized version of the algorithm proposed in (Takahashi and Matsuyama, 1980), named *RandomTM*. The modified heuristics is presented in algorithm 1 and is used to create the Steiner tree corresponding to the *k*-th multicast group. In (Takahashi and Matsuyama, 1980) the algorithm starts adding a terminal node to an empty tree. Then, another terminal is added based on the shortest path between it and a node in the tree. In the version proposed in this paper each terminal node to be added to the tree is chosen at random. The set of terminal nodes of the *k*-th multicast group D_k is created in line 1 of algorithm 1. The source node of the *k*-th multicast group, s_k, is set in line 2. The first node added to the tree is the source (line 3). Then, a new node is selected at random from D_k. It is linked to the source by the shortest path between them (lines 4 and 5). The shortest path is obtained with Dijkstra's algorithm (Dijkstra,

1956). The remaining destination nodes are added to the tree in the main loop between lines 8 and 13. One node in the tree and one node out of the tree (line 9) are chosen randomly and the shortest path between them is computed (line 10). This process is repeated until all destination nodes are in the tree. The inclusion of a path in the tree may create cycles. Therefore procedure nodeNotInSolution( ) is called to avoid the creation of cycles.

Algorithm 1 – RandomTM

Input: MulticastGroup *k*, Graph *G*
Output: SteinerTree *S*
1: D_k ←*DestinationNodes* (k)
2: s_k ←*Source* (k)
3: S ← s_k
4: d ← *randNode* (D_k)
5: Path ←Dijkstra (d, s_k)
6: S← S ∪ { *NodeNotInSolution*(path) }
7: *remove*(d, D_k)
8: while *size* (D_k) > 0
9 d ← *randNode*(D_k)
10: path ←*Dijkstra* (d, *randNode*(S))
11: S ←S ∪{ *NodeNotInSolution*(path) }
12: remove (d, D_k)
13: end while

The second heuristic is based on Kruskal's algorithm to generate minimum spanning trees (Kruskal, 1956). It will be referred to as algorithm 2 in this paper. The goal is to create multicast trees that contribute to maximize the minimal residual capacity. Basically, it works like Kruskal's algorithm, but in this version a restricted candidate list (Martins *et al.*, 1999) is implemented from where the next edge to be added to the tree is picked. The edges are sorted in ascending order of capacities in list *L*. One edge is selected at random from the 30% first ones in list *L*. The selected edge is added to the tree if it does not induce a cycle. The procedure continues until a tree is built with all terminal nodes of the *k*-th multicast group. The trees created by each algorithm are submitted to a procedure to eliminate leaves that are not terminal nodes.

Algorithm 2 – Algorithm based on Minimum Spanning Trees

Input Graph *G(V,E)*, MultcastGroup *k*
1: DisjointSet DS ← *AddNodes*(V)
2: L ←*sort* (E)
3: while L < $V - 1$
4: pos ←*randNumber* (0, *size*(L) * 0.3)
5: e ←L[pos]
6: if DS.union(e.x , e.y) then //if a cycle is not created
7: S ←S ∪ {e}
8: end if
9: remove(e, L)
10: end while

3.2 GRASP

The construction phase of the GRASP algorithm is presented in Algorithm 3. The algorithm has four input parameters: a graph *G* representing the network; the set of multicast groups, *K*; a reference for a heuristics (algorithm 1 or 2); the length of the restricted candidate list, *NRCL*.

The algorithm contains a main loop in which |*K*| trees are created (one for each multicast group $k \in K$). The loop starts by choosing a multicast group *k* at random (line 2). All edges from *G*

that do not support the traffic required by the *k*-th multicast group, t_k , are removed (line 3). A restricted candidate list RCL is created to the *k*-th multicast group, heuristic h (algorithm 1 or algorithm 2) in line 4. The RCL contains NRCL Steiner trees. A multicast tree is chosen randomly for *k* (line 5). This multicast tree is added to the MPP solution (line 6). After, the graph is restored in line 7(the edges removed are reinserted). The loop iterates until a Steiner tree is associated with each multicast group.

Algorithm 3 – Solution creator based on Restricted Candidate List.

Input Graph *G*, MulticastGroups *K*, Heuristic *h*, *NRCL*
Output MPPSolution *T*
1: While $|K| > 0$ do
2: $k \leftarrow$ *chooseGroupRAndomly* (*K*)
3: *remove* (k, *G*)
4: RCL $\leftarrow$ *creatList* (*h*,*k*, *NRCL*)
5: *SteinerTree* $T_k \leftarrow$ *chooseComponent* (*RCL*)
6: $T \leftarrow T \cup T_k$
7: restore(*G*)
8: $K \leftarrow K - \{k\}$
9: end while

Algorithm 4 – Local Search Operator for the MPP

Input: MPPSolution *T*
Output: Non-dominated Archive *A*
1: $A \leftarrow A \cup T$
2: T.visited $\leftarrow$ false
3: while $\exists T' \in A \mid T'$.visited = false do
4: for $\forall T' \in$ *Neighborhood*(*T*') do
5: if not $(\exists S, S \in A) | S \prec T''$ then
6: *T''.visited* = false
7: for $S \in A$ do
8: if $T'' \prec S$ then
9: $A \leftarrow A\text{-}S$
10: end if
11: end for
12: $A \leftarrow A \cup T''$
13: end if
14: end for
15: *T'.visited* $\leftarrow$ true
16: end while

The local search operator used in the GRASP algorithm is based on the PLS algorithm proposed by Paquete and Stützle (2006). The neighborhood structure consists of the systematic substitution of each multicast tree corresponding to a group in the input MPP solution. For example, given an MPP instance with five multicast groups, for each multicast group *k* a multicast tree is replaced by a new multicast tree relative to *k*. The number of neighbors is equal to the number of multicast groups. . For example, given an MPP solution $T = \{T_1,\ldots,T_{|K|}\}$, a neighbor $T' = \{T'_1,\ldots,T'_{|K|}\}$, of *T* consists of a solution where $\exists i$, $1 \le i \le |K|$, such that $T'_i \neq T_i$ and $\forall j$, $j \neq i$, $1 \le j \le |K|$, $T'_j = T_j$.

The local search procedure receives solution *I* as input parameter and adds it to archive *A*, the output of the algorithm. Solution *I* is marked "false", meaning that it has not been explored. The main loop of the local search is started. While there is, at least, one solution I' not

explored in *A* (line 3), each of its neighbors, *I"*, is generated (line 4). If *I"* is not dominated by any solution in *A* (line 5), the algorithm checks for solutions in *A* dominated by *I"* and remove them, if they exist (lines 7 and 8). I'' is added to *A* (line 12). Finally, the individual I' is marked as explored (line 15). The algorithm returns the set of non-dominated solutions A.

The GRASP is shown in algorithm 5. Variable *h* is set to 0 or 1, whether algorithm 1 or 2, respectively, is assigned to the construction procedure shown in algorithm 4. In the main loop an MPP solution *S* is created with heuristic *h* (line 5) – algorithm 1 or algorithm 2. Local search is applied to *S* (line 6) returning archive A_t of non-dominated solutions generated in the *t*-th GRASP iteration. The algorithm maintains an archive *Q* of non-dominated solutions generated along the GRASP iterations. Each iteration *Q* is updated to maintain only non-dominated solutions (line 5). The algorithm changes the reference of the heuristic used to construct a new solution (line 6). This process is repeated until the algorithm performs t_{max} iterations and the algorithm returns *Q*.

Algorithm 5 – GRASP
Input: t_{max}
Output: NonDominatedSet *Q*
1: $t \leftarrow 0$; $h \leftarrow 0$
2: While ($t < t_{max}$)
3: $S \leftarrow$ CreateSolution(h)
4: $A_t \leftarrow$ LocalSearch(S)
5: $Q \leftarrow Q \cup A_t$
6: $h \leftarrow$ not h
7: end_while
8: return(Q)

3.3 NSGA2 and SPEA2

This section presents the operators used in algorithms NSGA2 and SPEA2. Those were implemented following the directions given in the works of Deb *et al*. (2002) and Zitzler *et al*. (2002), respectively.

The procedure presented in Algorithm 6 was used to create MPP solutions. The input data are: a graph *G*, the set *K* of multicast groups and a reference for a constructive heuristics (algorithms 1 or 2). Algorithm 6 builds iteratively a tree for each multicast group in *K*. It starts choosing a group $k \in K$ at random and removing from *G* all edges that do not support the required traffic (line 3). A construction heuristic is randomly chosen and a multicast tree is created (lines 4 and 5). This process is repeated until the trees of all multicast groups are created.

Algorithm 6 – Solution creator
Input: Graph G(V,E) , MulticastGroups *K*, Heuristic *H*
Output: MPPSolution *T*
1: While |K| > 0 d
2: k ← *chooseGroupRAndomly* (K)
3: *remove* (k, G)
4: h ← chooseHeuristicRandomly (H)
5: $T_k \leftarrow$ createSteinerTreeh(h,k)
6: $T \leftarrow T \cup T_k$
7: restore(G)
8: K ← K – {k}
9: end while

The recombination operator for the MPP solutions uses the recombination procedure for Steiner tree presented in Algorithm 7 called STRecomb. It is a modified version of the algorithm proposed by Ravikumar and Bajpai (1998). Algorithm 7 recombines two Steiner trees

corresponding to the *k*-th multicast group of two MPP solutions T1 and T2 and returns Steiner tree T[*k*].

Algorithm 7 – STRecomb	Algorithm 8 - MPPRecomb
Input: SteinerTrees *T1[k], T2[k]* Output: SteinerTree *T* 1: E ← common_edges(T1[*k*], T2[*k*]) 2: if isEmpty (E) then 3: T ← lowestCost (T1[*k*], T2[*k*]) 4: else 5: if ConnectComponent(E) 6: T ← *E* 7: else 8: N ← DestinationNodes (*k*) 9: for *n* ∈ N do 10: if *n* NotIn(T) then 11: *p* ← getNodeRandomly (T) 12: path ←Dijkstra(*p*,*n*) 13: T ← T ∪ {path} 14: end if 15: end for 16: end if	Input: *S1*, *S2*, |*K*| Output: *S* 1: pos ← *random*(0,|*K*|) 2: for *i* ← 0; *i* < *pos*; *i*++ do 3: *S*[*i*] ← *STrecomb*(*S*1[*i*],*S*2[*i*]) 4: end for 5: for (*i* ← pos; *i* < |*K*|; *i*++) do 6: *S*[*i*] ← *S*2[*i*] 7: end for

The first step of algorithm 7 computes the set of common edges in Steiner trees T1[*k*] and T2[*k*]. If this set is empty (line 2), the Steiner tree with the lowest cost is set to T[*k*] (line 3). Function *ConnectComponents*( ) checks whether set *E* corresponds to a connected graph (line 4). If it is connected, then T is the tree corresponding to *E*. Otherwise, edges that link connected components are added to this set. After, if there are, at least, two disjoint sets, they are joined using the shortest path between them. The path is computed using Dijkstra's algorithm.

The loop from line 9 to 15 is executed to include destination nodes in group *k* that are not in tree T. The isolated destination node is added to T by the shortest path between it and a random node in T.

Algorithm 8 is used to recombine MPP solutions. It receives two MPP solutions, *S*1 and *S*2, and the number of multicast groups *K*. The output is MPP solution *S*. Initially, a cut point, *pos*, is defined (line 1). The trees corresponding to MPP Solutions *T*1 and *T*2 between positions 0 and *pos*-1 are recombined with algorithm 7 (line 3). The other Steiner trees of the offspring are copied from *T*2 (lines 5 to 7).

Mutation

The mutation operator uses one of the two heuristics (algorithm 1 or 2) to alter a given MPP solution, *S*. A multicast group *k* is randomly chosen from *S*. A tree associated to *k* is removed from *S* and a construction heuristic is randomly chosen to generate a new Steiner tree to the *k*-th component of *S*.

4. Experiments

The experiments were performed in an Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz using the Ubuntu operational system. All codes were implemented in C++. The process to create the instances for the experiments incorporated the Waxman's model (Waxman, 1988) and the Framework BRITE (Medina *et al.*, 2001). Instances are classified according to number of nodes and number of multicast groups. Instances were created with 30, 60 and 120 nodes and 5, 10, 15, 20 and 25 multicast groups. Other characteristics of those instances are presented in Table I, where

column nodes presents the number of nodes, column Node/group shows the percentage of nodes considered for each multicast group.

Nodes(N)	Groups	Node/Group	Capacity	Edges(E)
30	5,10,15,20,25	20-30%	15 a 85	2N
60	5,10,15,20,25	15-30%	25 a 85	2N
120	5,10,15,20,25	10-30%	35 a 85	2N

Table 1 – Instances characteristics

The approximation sets produced by each approach were compared with the hypervolume indicator (I_H). The Mann-Whitney's test was applied to the results.

The parameters of NSGA2 are: population size = 150, recombination rate = 0.5 and mutation rate =0.3. The parameters of SPEA2 are: population size = 150, archive size = 20, recombination rate = 0.5 and mutation rate 0.1. The size of the RCL in the GRASP algorithm was 0.3*n*, where *n* denotes the number of destination nodes.

Instance	I_H	Instance	I_H	Instance	I_H
B030_1	1	B060_1	1	B120_1	1
B030_2	1	B060_2	1	B120_2	1
B030_3	1	B060_3	1	B120_3	1
B030_4	1	B060_4	1	B120_4	1
B030_5	1	B060_5	1	B120_5	1
B030_6	1	B060_6	1	B120_6	1
B030_7	0,9999977	B060_7	1	B120_7	1
B030_8	0,9999974	B060_8	1	B120_8	1
B030_9	1	B060_9	1	B120_9	1
B030_10	1	B060_10	1	B120_10	1
B030_11	1	B060_11	1	B120_11	1
B030_12	1	B060_12	1	B120_12	1
B030_13	1	B060_13	1	B120_13	1
B030_14	1	B060_14	1	B120_14	1
B030_15	0,9999997	B060_15	1	B120_15	1
B030_16	0,9999384	B060_16	1	B120_16	1
B030_17	**3,75E-011**	B060_17	1	B120_17	1
B030_18	1	B060_18	1	B120_18	1
B030_19	**2,55E-007**	B060_19	0,1403482	B120_19	1
B030_20	1	B060_20	1	B120_20	1
B030_21	0,9879794	B060_21	**0,00170355**	B120_21	**6,06E-007**
B030_22	**1,27E-014**	B060_22	1	B120_22	1
B030_23	**1,50E-015**	B060_23	1	B120_23	0,9999999
B030_24	1	B060_24	1	B120_24	1
B030_25	**2,78E-010**	B060_25	**5,76E-007**	B120_25	**2,61E-006**
B030_26	**8,37E-015**	B060_26	**2,82E-033**	B120_26	1
B030_27	0,2566827	B060_27	0,999925	B120_27	0,892516
B030_28	**1,72E-030**	B060_28	1	B120_28	1
B030_29	**6,44E-024**	B060_29	0,9999999	B120_29	1
-	-	B060_30	**9,22E-005**	B120_30	**1,12E-010**

Table 2 - Comparison between NSGA2 and GRASP

The processing time was fixed in 60s, 120s and 300s for instances with 30, 60 and 120 nodes, respectively. One hundred independent executions of each algorithm were performed for each instance. Tables 2-3 present the p-values returned by the Mann-Whitney's test for pairwise comparisons.

Table 2 shows the comparison between GRASP and NSGA2. The underlined values indicate that the NSGA2 produced approximation sets better than the ones produced by GRASP. Bold values indicate the opposite, i.e., GRASP is better than NSGA2. Considering significance level 0.01, the results show that the NSGA2 outperforms the GRASP in 74 instances.

Instance	I_H	Instance	I_H	Instance	I_H
B030_1	**5,91E-030**	B060_1	**1,32E-034**	B120_1	**1,28E-034**
B030_2	**2,01E-034**	B060_2	**1,44E-034**	B120_2	**1,28E-034**
B030_3	**2,63E-034**	B060_3	**1,36E-034**	B120_3	**1,28E-034**
B030_4	**1,28E-034**	B060_4	**1,28E-034**	B120_4	**1,28E-034**
B030_5	**1,33E-032**	B060_5	**1,32E-034**	B120_5	**1,81E-033**
B030_6	**2,60E-032**	B060_6	**1,28E-034**	B120_6	**1,28E-034**
B030_7	**3,74E-015**	B060_7	**6,42E-030**	B120_7	**1,28E-034**
B030_8	**1,19E-005**	B060_8	**1,49E-034**	B120_8	**6,33E-028**
B030_9	**6,43E-031**	B060_9	**1,28E-034**	B120_9	**1,28E-034**
B030_10	**6,14E-025**	B060_10	**1,28E-034**	B120_10	**1,28E-034**
B030_11	**8,53E-031**	B060_11	**1,36E-034**	B120_11	**1,28E-034**
B030_12	**1,28E-034**	B060_12	**1,28E-034**	B120_12	**1,28E-034**
B030_13	**1,68E-034**	B060_13	**1,89E-034**	B120_13	**5,28E-023**
B030_14	**2,55E-014**	B060_14	**6,79E-015**	B120_14	**1,34E-031**
B030_15	**0,00542905**	B060_15	**3,44E-034**	B120_15	**1,78E-034**
B030_16	**5,25E-005**	B060_16	**8,16E-034**	B120_16	**4,44E-022**
B030_17	1	B060_17	**7,03E-034**	B120_17	**1,28E-034**
B030_18	**8,73E-032**	B060_18	**4,69E-008**	B120_18	**2,79E-034**
B030_19	1	B060_19	0,9996539	B120_19	**1,54E-032**
B030_20	**1,48E-024**	B060_20	0,1201878	B120_20	**5,22E-007**
B030_21	**0,00061597**	B060_21	0,9861926	B120_21	1
B030_22	1	B060_22	**3,65E-034**	B120_22	**2,79E-034**
B030_23	1	B060_23	**2,35E-015**	B120_23	0,9999906
B030_24	**0,00093021**	B060_24	**2,10E-033**	B120_24	**5,03E-023**
B030_25	1	B060_25	0,9999996	B120_25	1
B030_26	1	B060_26	1	B120_26	**9,24E-032**
B030_27	0,9684295	B060_27	0,5403838	B120_27	1
B030_28	1	B060_28	**1,64E-026**	B120_28	**5,25E-005**
B030_29	1	B060_29	**0,0041443**	B120_29	**1,36E-034**
		B060_30	0,9999999	B120_30	1

Table 3 – Comparison between SPEA2 and GRASP

Table 3 shows the comparison between SPEA2 and GRASP. Bold values indicate that the SPEA2 has approximation sets better than the GRASP. Underlined values indicate the opposite, GRASP has better approximation sets. In the same way of the NSG2, the SPEA2 outperforms the GRASP algorithm in almost all instances. The SPEA2 outperforms the GRASP in 70 instances.

Table 4 shows the comparisons between the NSGA2 and the SPEA2 algorithms presented for the MPP, bold values indicates that SPEA2 is better than NSGA2, underlined values indicate the opposite, i.e., NSGA2 is better than SPEA2 . The results on instances with 30 nodes do not support conclusions of superior performance of one of the two tested algorithms. Nevertheless, NSGA2 exhibits better performance than NSGA2 on groups of instances with 60 and 120 nodes. This conclusion is supported by the fact that the SPEA2 does not outperform the NSGA2 algorithm in any instances including groups of instances with 60 and 120 nodes. On the other hand, the NSGA2 outperforms the SPEA2 algorithm 32 instances.

The evolutionary algorithms outperform the GRASP algorithm in the previous comparisons. Therefore, a comparison between them can indicate which is better. The table 4 illustrates the comparison. For instances with size equal 30 nodes there was not an algorithm

outperforming other algorithm, but for instances with size equal 60 and 120 nodes the NSGA2 algorithm outperforms the SPEA2 in most of instances.

Instance	I_H	Instance	I_H	Instance	I_H
B030_1	0,9254646	B060_1	0,9988946	B120_1	1
B030_2	0,5233811	B060_2	0,8601958	B120_2	0,6167178
B030_3	0,6204433	B060_3	0,9219611	B120_3	0,2622244
B030_4	0,7824996	B060_4	0,9997759	B120_4	0,9810372
B030_5	0,9925508	B060_5	0,9376903	B120_5	0,8478945
B030_6	0,7903224	B060_6	0,2694384	B120_6	0,3519924
B030_7	**0,03577276**	B060_7	1	B120_7	0,07282936
B030_8	0,4936643	B060_8	0,9999526	B120_8	0,9996251
B030_9	0,06661799	B060_9	0,9456523	B120_9	0,9999665
B030_10	0,3767698	B060_10	0,9856672	B120_10	0,141989
B030_11	0,09871858	B060_11	0,470297	B120_11	0,9782803
B030_12	0,8841593	B060_12	0,4153529	B120_12	0,6443734
B030_13	0,700747	B060_13	0,299253	B120_13	0,9805803
B030_14	0,9945328	B060_14	0,9898295	B120_14	0,9961169
B030_15	0,8655574	B060_15	0,9890795	B120_15	0,9999859
B030_16	0,4441345	B060_16	0,9719017	B120_16	0,9802316
B030_17	0,1555198	B060_17	0,8812738	B120_17	0,8094727
B030_18	0,4470316	B060_18	0,9868516	B120_18	0,9898295
B030_19	0,9240779	B060_19	0,9938045	B120_19	1
B030_20	0,7986646	B060_20	0,9999981	B120_20	0,9998594
B030_21	0,1299976	B060_21	0,4239471	B120_21	1
B030_22	0,4153528	B060_22	0,1287099	B120_22	0,9806954
B030_23	0,6653568	B060_23	0,9990225	B120_23	1
B030_24	0,996838	B060_24	0,9991368	B120_24	0,9999998
B030_25	0,9437415	B060_25	0,8000353	B120_25	1
B030_26	0,9581777	B060_26	0,2284346	B120_26	0,9980993
B030_27	0,8348949	B060_27	0,999964	B120_27	1
B030_28	0,7828607	B060_28	0,6167178	B120_28	1
B030_29	0,6352308	B060_29	0,9947581	B120_29	0,07282936
		B060_30	0,8995716	B120_30	1

Table 4 - Comparison between NSGA2 and SPEA2.

5. Conclusions

This paper presented a study on the Multicast Packing Problem under the viewpoint of multi-objective optimization. A mathematical formulation was presented, incorporating two important requirements in multicast trees construction: cost and residual capacity. An NSGA2, a SPEA2 and a GRASP algorithm were proposed in this paper for the MPP. Eighty-nine instances were created for the computational experiments. They were compared based on the results of the Mann-Whitney's test performed on the values of the hypervolume indicator of the approximation sets generated by the tested algorithms. There is statistical evidence that NSGA2 outperformed the other algorithms on the set of instances considered in the experiment.

Acknowledgments

This research was partially supported by CNPq and CAPES.

References

Chen, S., Gunluk, O., and Yener, B. (2000). The Multicast Packing Problem. *IEEE/ACM TRANSACTIONS ON NETWORKING, 8*, 311-318.

Conover, W. (1980). *Practical nonparametric statistics.* Wiley.

Cui, X., Lin, C., and Wei, Y. (2003). A Multiobjective Model for QoS Multicast Routing Based on Genetic Algorithm. *Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing* (pp. 49--). Washington, DC, USA: IEEE Computer Society.

Feo, T. and Resende, M. (1995), Greedy Randomized Adaptive Search Procedures. J. of Global Optimization, v. 6, p. 109133.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*, 182-197.

Dijkstra, E. W. (1956). A Note on two problems in connection with graphs. *Numerische math, 1*, 269-271.

Han, L., and Shahmehri, N. (2000). Secure multicast software delivery. *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, (pp. 207-212).

Jia, X., and Wang, L. (1997). A group multicast routing algorithm by using multiple minimum Steiner trees. *Computer Communications, 20*(9), 750-758.

Kang, J., Park, K., and Park, S. (2009). Optimal multicast route packing. *European Journal of Operational Research, 196*, 351-359.

Kruskal, J. B. (February de 1956). On the Shortest Subtree of a Graph and the Traveling Salesman Problem. 48-50.

Lee, C. Y., and Cho, H. K. (June de 2004). Multiple multicast tree allocation in IP network. *Comput. Oper. Res., 31*, 1115-1133.

Low, C. P., and Wang, N. (1999). An Efficient Algorithm for Group Multicast Routing with Bandwidth Reservation. *Proceedings of the 7th IEEE International Conference on Networks* (pp. 43--). Washington, DC, USA: IEEE Computer Society.

Martí, R., Campos, V., Resende, M. C., and Duarte, A. (June de 2011). *Multi-Objective Grasp With Path-Relinking.* Tech. rep., ATT Labs Research Technical Report, Florham Park, NJ 07932.

Martins, S. L., Pardalos, P. M., Resende, M. G., and Ribeiro, C. C. (1999). Greedy Randomized Adaptive Search Procedures For The Steiner ProblemIn Graphs. (pp. 237-261). American Mathematical Society.

Medina, A., Lakhina, A., Matta, I., and Byers, J. (2001). BRITE: An Approach to Universal Topology Generation. *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (pp. 346--). Washington, DC, USA: IEEE Computer Society.

Medina, A., Lakhina, A., Matta, I., and Byers, J. (2001). *BRITE: Universal Topology Generation from a User''s Perspective.* Boston, MA, USA: Boston University.

Oliveira, C. A. (2004). *Optimization problems in telecommunications and the internet.* Gainesville, FL, USA: University of Florida.

Paquete, L., and Stützle, T. (2006). A study of stochastic local search algorithms for the biobjective QAP with correlated flow matrices. *European Journal of Operational Research*, 943-959.

Ravikumar, C., and Bajpai, R. (1998). Source-based delay-bounded multicasting in multimedia networks. *Computer Communications, 21*(2), 126-132.

Resende, M. G., and Pardalos, M. P. (2006). *Handbook of Optimization in Telecommunications* (1 ed.). (M. G. Resende, and M. P. Pardalos, Eds.) Springer.

Resende, M., and Ribeiro, C. (2010). Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. In: M. Gendreau, and J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (Vol. 146, pp. 283-319). Springer US.

Takahashi, H., and Matsuyama, A. (1980). An approximate solution for the Steiner problem in graphs. *Math Japonica, 24*(6), 573-577.

Wang, C.-F., Liang, C.-T., and Jan, R.-H. (June de 2002). Heuristic algorithms for packing of multiple-group multicasting. *Comput. Oper. Res., 29*, 905-924.
Waxman, B. M. (1988). Routing of Multipoint Connections. *IEEE Journal of Selected Areas in Communications, 6*(9), 1617-1622.
Wu, Z. (2005). Performance modeling of multicast groups for multiplayer games in peer-to-peer networks. *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on*, (pp. 105-112).
Xu, Y. (2011). *Metaheuristic Approaches for QoS Multicast Routing Problems*. Ph.D. dissertation, University of Nottingham.
Zitzler, E., and Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach*.
Zitzler, E., Laumanns, M., Thiele, L., Fonseca, C. M., and Grunert da Fonseca, V. (2002). {Why Quality Assessment Of Multiobjective Optimizers Is Difficult}. *Genetic and Evolutionary Computation Conference {(GECCO 2002)}* (pp. 666-674). New York, NY, USA: Morgan Kaufmann Publishers.
Zitzler, E., Laumanns, M., & Thiele, L. (2001). {SPEA2: Improving the Strength Pareto Evolutionary Algorithm}. TIK Report, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland.