Optimizing Multiple Objectives on Multicast Networks using Memetic Algorithms

Yezid Donoso[1], Alfredo Pérez[1], Carlos Ardila[1],
and Ramón Fabregat[2]

[1] Departamento de Ingeniería de Sistemas y Computación
Universidad del Norte
Barranquilla, Colombia
{ydonoso, perezaj, cardila}@uninorte.edu.co
[2] Institut d'Informàtica i Aplicacions, Universitat de Girona,
Girona, Spain
ramon@eia.udg.es

Abstract. In this paper, we propose the use of Memetics Algorithms for solving multiobjective network problems. The aim is to minimize total delay and total hop count on sending a multicast flow through a network. We propose a heuristic that avoid the creation of subgraphs for finding paths. The heuristic combined with multiobjective evolutionary algorithms have shown similar results to the results obtained by GAMS solver. The main contribution of this paper is the heuristic developed for the problem.

1 Introduction

With the increase of applications that use multicast flows, the management of network resources has become an important research topic. Researches on multiobjective approaches to manage these networks, offers an interesting alternative to integrate competitive objectives that are finally reflected on costs and quality of service.

As several problems with multicast networks have shown to be *NP- Complete*, metaheuristics become an alternative to solve these problems. This paper shows the metaheuristic called Memetic Algorithms to the problem of the simultaneous optimization of Total Delay and Total Hop count of sending a flow through a Multicast Network.

Memetic algorithms (MA's) present a paradigm for solving complex problems through the combination of heuristics and Evolutionary Algorithms (EA's). We present on this paper a heuristic that combined with well known Multiobjective Evolutionary Algorithms (SPEA2, M – PAES, RD – MOGLS) have shown a similar behavior than the results obtained by GAMS solver. The resulting EA's should be called Memetic Algorithms.

2 Multiobjective Optimization

Multiple objective problems can be seen everyday on real life. For instance, when we buy at the supermarket, we try to find products that have very good quality but at good price. As in real life, in engineering and other areas of knowledge, a variety of problems intents to optimize several and competitive objectives. In contrast with Simple Objective Optimization, where a solution to a problem is a point, in Multiobjective Optimization the solution is a set. This set represents the best solutions that optimize simultaneously all objectives, and allow us to decide *after* the process of optimization has been done.

2.1 Pareto Optimization

The process of finding one or many solutions that satisfy the constraints and optimize simultaneously a set of functions is known as Multiple Objective Optimization. Mathematically, this process can be described [1] as

$$\text{“}\textit{Optimize}\text{”}\ \mathbf{z} = \mathbf{F(x)}$$
$$\textit{Subject to}\ \mathbf{x} \in X_f$$

where $\mathbf{F(x)} = \{\mathbf{f_1(x)},\mathbf{f_2(x)},\ldots,\mathbf{f_k(x)}\}$, k is the number of functions that are being optimized and X_f is the feasible set.

In Pareto Optimality, a solution can be *dominated, non-dominated* and *non comparable* with another solution. A solution ***y*** is said to be *dominated* (*strong dominance* in context of minimization and symbolized as $x \prec y$) by other solution ***x*** if $\mathbf{f(x) < f(y)}$, this is, if a solution ***x*** has a better evaluation on all functions than ***y*** then ***y*** is dominated by ***x***. A solution is said to be *non comparable* with another solution if the solutions are not dominated between themselves, this is if $x \not\prec y$ and $y \not\prec x$.

Under this approach, the solution of a multiobjective problem is a set. The solutions in this set are called *Pareto Optimal* and the set of these *Pareto Optimal* solutions is called *Pareto Set.* The image of this set is called *Pareto Front*.

2.2 Mathematical Model

The model presented is an adaptation of the model proposed by Donoso *et al.* in [2]. We minimize in this paper the Total Delay and Total Hop Count on sending a multicast flow over a network. The model presented in this paper is static.

Let *G(V,E)* a graph that represents the topology of a network. $T \subseteq V$, $T \neq \varnothing$, a subset that represents the egress nodes and $s \in V$ a node in the graph that represents the source of the flow. Let c_{ij} be the capacity of each link *(i,j)* and $f \in F$, be any multicast flow, where F is the flow set and T_f is the egress nodes subset to the multicast flow *f*. Note that $T = \bigcup_{f \in F} T_f$.

A solution to the problem is the tree $P = (p_1, p_2, p_3, \ldots, p_{|T|})$ where p_t represents a path from the node s to the node t, $t \in T_f$. Therefore, the problem is

$$Minimize\ \mathbf{z} = \{\mathbf{f_1(P)}, \mathbf{f_2(P)}\} \tag{1}$$

where

$$\mathbf{f_1(P)} = \sum_{f \in F} \sum_{t \in T_f} \sum_{(i,j) \in E} v_{ij} X_{ij}^{t_f} \tag{2}$$

$$\mathbf{f_2(P)} = \sum_{f \in F} \sum_{t \in T_f} \sum_{(i,j) \in E} X_{ij}^{t_f} \tag{3}$$

subject to

$$\sum_{(i,j) \in E} X_{ij}^{t_f} - \sum_{(i,j) \in E} X_{ji}^{t_f} = 1, t \in T_f, i = s, f \in F \tag{4}$$

$$\sum_{(i,j) \in E} X_{ij}^{t_f} - \sum_{(i,j) \in E} X_{ji}^{t_f} = -1, t \in T, f \in F \tag{5}$$

$$\sum_{(i,j) \in E} X_{ij}^{t_f} - \sum_{(i,j) \in E} X_{ji}^{t_f} = 0, t \in T_f, i \neq s, i \neq t, f \in F \tag{6}$$

$$\sum bw_f \cdot \max(X_{i,j}^{t_f}) \leq c_{i,j} \cdot a, a \geq 0, (i,j) \in E \tag{7}$$

(2) represents the Total Delay, (3) represents the Total Hop Count and $X_{ij}^{t_f} \in Z, \{0,1\}$ represents the use of the link (i, j) in the path from s to the egress node t (path p_t). (4), (5), (6) are the constraints for constructing paths on the graph and (7) stands for the bandwidth consumption that cannot exceed the maximum utilization per link (a), per link capacity c_{ij}.

3 Related Work

Multiobjective optimization in multicast networks has been studied by Donoso *et al.* in [2], [5] and [6]. In [6], Donoso *et al.*, propose a multiobjective traffic engineering schema using different distribution trees to several multicast flows. They combine into a single aggregated metric, the following weighting objectives: the maximum link utilization, the hop count, the total bandwidth consumption, and the total end-to-end delay. The authors formulate the multiobjective function as one with Non Linear programming with discontinuous derivatives (DNLP). Furthermore, a SPT (*Shortest Path Tree*) is proposed for solving the problem and the results of this algorithm are compared to the results obtained by SNOPT solver.

In [7], Banerjee *et al.,* present a genetic algorithm for solving the problem of routing and wavelength assignation in optical networks as a single objective problem and as a multiobjective optimization problem In the single objective problem, the

authors use a cost function based on the frequency of appearance of a link in different paths from a source to a destination to assign the *fitness* of a chromosome. The results for the single objective optimization of the genetic algorithm compared to the First-Fit Algorithm showed that with a small source-destination pairs, the results for both algorithms are similar, but for big source-destination pairs, the proposed genetic algorithm shows a better performance than the First-Fit Algorithm. The results for the multiobjective case, shows the capacity of the genetic algorithm to maintain the diversity of solutions.

Applications of multiobjective memetic algorithms on networks have been studied by Knowles *et al*. [8] for minimizing two objectives in telephone networks. Knowles' paper minimizes the cost of total routing (in terms of money per used link) and bandwidth consumption.

4 Memetic Algorithms

The concept of *meme* was created by Richard Dawkins in 1976 to explain how culture evolves. Dawkins made a parallel between Biological Evolution and Cultural Evolution and the result was the concept of *meme.* Dawkins defines a *meme* as a unit of cultural evolution that needs refinement [3]. A *meme* represents the way of doing things in human culture: the movements on martial arts, the cooking receipts, a way for cultivating crops, are example of *memes.* As result, *memes* have impact on physical reality. In Dawkins' theory, a *meme* can be combined with other *memes* for creating new *memes,* they can be mutated and they can evolve by themselves inside the mind of a human being.

Pablo Moscato in 1989 introduced the concept of *Memetics Algorithms* in [3], where he discusses how to adapt the *meme* to evolutionary computation. He defines that a MA is a marriage between global search made by the evolutionary algorithm and the local search (heuristic) made by each of the individuals. In this context, the *memes* are not the solutions of the problem, but the operators of local search that contains the *cultural information* about the problem that is being solved. From the figure 1, it can be seen that the difference between a MA and an EA is the incorporation of the procedure(s) of local search, for this reason, sometimes MA's are called *Hybrid Algorithms.*

```
Memetic Algorithm
1. generate initial solutions()
2. while(stop criteria is not met)
3.    local search()
4.    calculate fitness()
5.    selection()
6.    crossover()
7.    mutation()
8. end while
End
```

Fig. 1. Pseudocode of a Memetic Algorithm

Recently, a new approach to Memetic Algorithms was presented by Krasnogor [4] where he defines a representation of the local searches called *memeplexes*. In these algorithms, Krasnogor proposes that the local search should evolve with the solutions of the problem (individuals). As result, *memeplexes* can combine between themselves and mutate, having a closer approach to the concept proposed by Dawkins.

As Memetic Algorithms are the combination of heuristics and evolutionary algorithms, the chromosome representation, crossover operator and mutation operator must be defined for a problem. The chromosome representation, crossover operator and mutation operator will be defined as follows. Local search will be described in 4.3.

4.1 Chromosome Representation

A solution of the problem is a tree that represents the routing of the multicast flow. The tree is represented as a set of paths, each one from s to t, $t \in T_f$. The figure 2 shows the chromosome representation (individual) for a given routing tree.

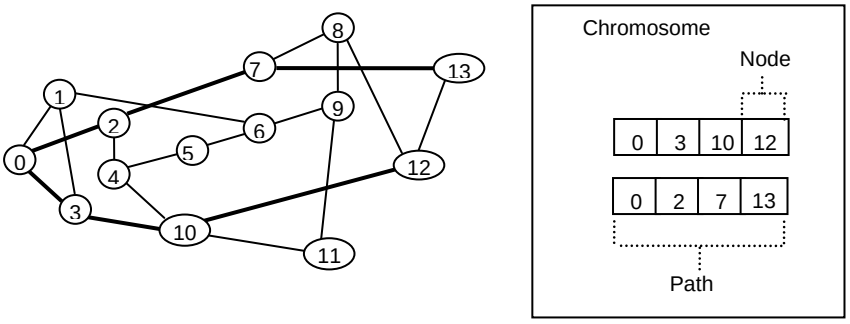

Fig. 2. Chromosome Representation

4.2 Crossover Operator

A Crossover operator is a function that takes a pair of chromosomes and produces a new chromosome that inheritates characteristics from its parents. Given two chromosomes, the crossover operator used chooses randomly an egress node and then it interchanges the path for that egress node between both chromosomes. One of the two generated trees is chosen randomly as the resulting chromosome.

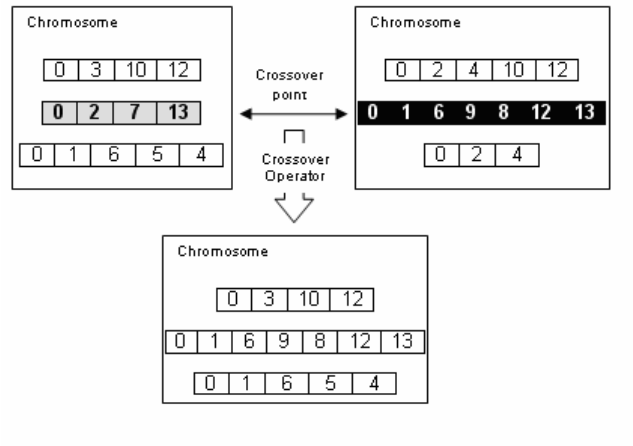

Fig. 3. Crossover Operator

4.3 Mutation Operator

The mutation operator takes an individual and makes a random change in it for creating a new individual. The mutation operator chooses randomly a destination and generates a new path using a Deep-First Search (DFS).

4.4 Local Search

Paths for a chromosome in the initial population are generated using a DFS search from the source to each destination. By using these randomly generated paths on a chromosome, new paths can be constructed by joining paths in intersection nodes; however two questions arise by doing this join. The first question is what happens when there are more than one intersection nodes and the second is how to generate a new path that does not contains loops using these intersection nodes?, for answering both questions, the concepts of *intersection point* and *distance of intersection function* are introduced.

Let *G(V,E)* a graph , *X, Y* a pair of acyclic paths between two nodes of *G* and *x* a node, $x \in V$. *x* is said to be an *intersection point* between *X* and *Y* if $x \in X - x \in Y$. An intersection point is denoted as $x_{XY}^{\angle}$. The set of intersection points will be denoted as I_{XY}.

Paths are generated by the heuristic choosing the best intersection point between a pair of paths inside an individual. For choosing the best intersection point, the *distance of interception function* is defined.

Let $x_{XY}^{\angle}$ be an intersection point of two paths from the same source, $Pos_X(x_{XY}^{\angle})$, $Pos_Y(x_{XY}^{\angle})$ a function that returns the position of a intersection point on the paths *X* and *Y* respectively. The *distance of intersection function* is defined as

$$D(x_{XY}^{\angle}) := \text{abs}(\text{Pos}_X(x_{XY}^{\angle}) - \text{Pos}_Y(x_{XY}^{\angle})) \quad (7)$$

where abs(x) stands for the absolute value function.

Using this function we can choose which interception point will be used to generate the join path. The $x_{XY}^{\angle}$ chosen is the one that maximizes $D(x_{XY}^{\angle})$ on the set I_{XY}. If there are more than one interception points with the maximum distance, any of these can be chosen. If $\forall x_{XY}^{\angle} \in I_{XY}, D(x_{XY}^{\angle}) = 0$ then the interception point with maximum $\text{Pos}_X(x_{XY}^{\angle})$ is chosen.

Once the interception point is chosen, the subpaths from the source to each $x_{XY}^{\angle}$ in the paths X and Y are generated. Let X_1 and Y_1 be these subpaths. If $X_1 \sim Y_1$ on the delay and hop count functions, the algorithm will not generate a new path. If $X_1 \prec Y_1$ then the subpath from $x_{XY}^{\angle}$ to the destination of the path of Y is generated and let this path be Y_2. The new path is created by joining the subpaths X_1 and Y_2 at the chosen intersection point. The same happens if $Y_1 \prec X_1$ but this time the new path is created by joining Y_1 with X_2.

The same algorithm applies between every pair of paths in the chromosome and an improved chromosome is obtained after the local search. The figure 4 shows the algorithm that generates the new path (heuristic).

A loop can be seen every time we intercept two paths in the interception points. By choosing the interception point with maximum distance, we are identifying the biggest loop in the subgraph formed by the two paths. This is the reason why the function generates acyclic paths; we are reducing the biggest loop.

```
Algorithm Path Generator (Path1,Path2)
1. I_Path1Path2 = get interception points (Path1,Path2)
2. If (I_Path1Path2 ≠ ∅) then
3.       x=select interception point(I_Path1Path2)
4.       Path1_1= get subpath(source,x,Path1)
5.       Path2_1= get subpath(source,x Path2)
6.      If(Path1_1 ≺ Path2_1 in delay and hop count) then
7.           Path2_2= get subpath (x, target(Path2), Path2)
8.           Return  Path1_1 Path2_2
9.      Else
10.          If(Path2_1 ≺  Path1_1 in delay and hop count) then
11.               Path1_2= get subpath (x, target(Path1), Path1)
12.               Return  Path2_1 Path1_2
13.          Else
14.               Return null.
15.          end If
16.    End If
17.Else
18.        Return null.
19. End if
End
```

Fig. 4. Heuristic for the problem