INT J COMPUT COMMUN, ISSN 1841-9836
Vol.7 (2012), No. 3 (September), pp. 417-431

GASANT: An ant-inspired least-cost QoS multicast routing approach based on genetic and simulated annealing algorithms

M. Damanafshan, E. Khosrowshahi-Asl, M. Abbaspour

Morteza Damanafshan, Ehsan Khosrowshahi-Asl
Department of Computer and Network,
Institute for Research in Fundamental Sciences (IPM),
Tehran, Iran.
E-mail: damanafshan@iranet.ir

Maghsoud Abbaspour
Department of Computer Engineering,
Faculty of Electrical and Computer Engineering,
Shahid Beheshti University, G.C., Tehran, Iran.
E-mail: maghsoud@sbu.ac.ir

Abstract:
Computing least-cost multicast routing tree while satisfying QoS constraints has become a key issue especially by growing communication networks. To solve this problem, a triplex algorithm called GASANT which is based on Ant Colony Optimization (ACO), Genetic Algorithm (GA), and Simulated Annealing (SA) has been proposed in this paper. Through ACO, we have both provided improved initial population to feed GA and reduced search process. Besides, SA has been deployed to refrain GA from getting stuck into local optimum solutions. Simulation results assert that GASANT not only has high speed convergence time, but also generates least-cost multicast routing trees of high QoS.
Keywords: Multicast Routing; Quality of Service (QoS); Ant Colony Optimization (ACO); Genetic Algorithm (GA); Simulated Annealing (SA)

1 Introduction

Multicasting service is a technique in which the same information is sent concurrenly from a source node to a subset of all possible destinations (multicast group) in a computer network. The current approach to provide such a service is to establish a multicast tree. This tree includes a route node (sender), some internal nodes (intermediate routers) and some leaf nodes (recipients). To carry large numbers of multicast sessions, a network must minimize the sessions' resource consumption [1]. Therefore, it is important for a multicast session to adopt a multicast tree whose network cost is minimal. By network cost we mean the accumulation of the costs of resource usages of all the links constructing the multicast tree. This problem immediately is reduced to finding a Steiner Tree [2] which is one of the Karp's 21 NP-complete problems [3]. The tree cost should be minimized to the most possible extent. This is due to the fact that after the multicast tree is built, all the network traffic flows along the links of the tree, especially in real-time applications which are intrinsically connection-oriented. The less the tree cost, the less the valuable resources are used during the whole connection time. Finding minimal multicast tree gets more problematic when some Quality-of-Service (QoS) constraints such as delay (end-to-end delay), and bandwidth constraints are also to be considered at the same time. Finding the multicast tree or the Steiner tree under any of the aforementioned QoS constraints converts the problem into finding a constrained Steiner tree (QoS multicast routing) problem, which is NP-Complete itself [4].

Several methods [4–9] have applied heuristic to solve QoS multicast routing problem. KPP [4], BSMA [5], and some others [6–8] are notable heuristic works in computing multicast trees. However, a comprehensive research in [9] has shown that most of the heuristic algorithms are notorious either for working too slowly or failing in computing of an optimized solution or both.

Some works [1, 10–13, 15–20] have mainly focused on applying GA to find constrained multicast routing trees. In [1] a bandwidth delay constrained least-cost multicast routing algorithm based on conventional GA has been proposed. It has used tree structure coding for chromosome representation and penalty functions for those candidate solutions that violate predefined thresholds. Besides, [10, 11] have solely emphasized on conventional GA. However, all these methods [1, 10, 11] suffer from lack of local search and also problem of premature convergence. Also, all these approaches generate their initial population mainly based on a randomized depth-first search algorithm [12, 13].This method suffers from applying uninformed search which most of the time performs worse than a good heuristic based informed search [14].

Some others [16, 17] apply Shimamoto's approach [18] for coding routing tables and multicast trees. In this method for each pairs of (source, multicast-destination) several paths are stored, and the final multicast tree is yielded through combining these paths. As the network size grows, maintaining these paths can itself be a problem.

The closest work to ours is [20] which presents a method namely NGSA for least-cost QoS multicast routing based on both GA and Simulated Annealing (SA) algorithm. This paper ([20]) adopts a rather new population initialization method mostly the same way as [12,13] which has two steps: trunk-creating and limb-appending. In trunk creating phase, a path is found from a source node to one of the multicast destinations. Then, in the limb-appending phase, other multicast destination nodes are appended to the trunk through randomly discovered paths. [20] also uses SA to escape from premature convergence, one of the eminent shortcomings of GA. However, finding paths in both phases is done through random uninformed selection of neighbors which suffers from deficiencies inherent in uninformed search methods mentioned before.

Considering the fact that multicast tree creation and maintenance time is crucial and meanwhile GA's evolution time toward better solution can be unpredictable [21], it is important to improve convergence speed. Generating improved initial population [22], and reducing search process are two effective approaches to achieve this goal.

[22] has shown that improving generation of initial population and being meticulous about it can significantly improve convergence time. All the aforementioned GA based researches applied random selection approaches in their initial population generation. Adopting such a randomized behavior may cause the algorithm to go astray in establishing a multicast tree at least for a while. Therefore, GA must compensate its improper selections by making further attempts, and this prolongs the convergence time. Therefore, instead of passing the buck to next generations in GA and expecting the next generations to compensate the primitive generations' probable fault, it is reasonable to make the first decision more scrupulously. This becomes more important when we cope with large networks.

Reducing search process can also be considered as another method to improve convergence time. Actually, reducing search process hinders GA algorithm-at least for a great extent-from blundering and moving back and forth and revisiting the same links for an excessive number of times in hope of finding a solution. Applying a randomed paradigm automatically causes GA to undesirably adopt try and error behavior to acheive a satisfactory solution.

In this paper, we have proposed and impelmented a new algorithm called GASANT which takes the aforementioned two improvements on GA into consideration. To put it in a nutshell, we have both provided improved initial population and reduced search process by deploying Ant Colony Optimization (ACO). By improved initial population, we mean that links constructing initial population are more likely to appear in optimal multicast tree. By reducing search process, we mean that for finding a satisfactory solution, GASANT visits edges of the network graph for a small number of times rather than excessive number of times. Besides, SA has been used to escape from getting stuck into local optimum solutions.

ACO is based on distributed society of autonomous agents called ants. Ants provides a valuable approach of exploring the network and collecting information about link statuses like link-state protocols, but in an efficient and deliberate way. Ants can provide better initial population for GA, since they traverse through network and discover (near) optimal paths among nodes. The produced least-cost multicast tree more likely contains a subset of edges comprising these optimal paths. Thus, considering such paths while we want to establish a multicast trees can result in a solution in higher speed. Consequently, the search process is reduced. The experiment results proves this claim.

Also, note that ants are small packets and put low load burden on network nodes; their lightweight approach can be very effective for gathering information for generating QoS-aware initial population [23]. The experiments certify this claim, too.

The rest of the paper is organized as follows: Section 2 explains the problem formulation and modeling. Section 3 describes the proposed multicast routing algorithm (GASANT) in detail. In section 4, the convergence of GASANT is investigated. We evaluate GASANT comprehensively in section 5, and finally section 6 concludes the paper.

2 Problem Formulation and Modeling

In this paper, the network is expressed as an undirected weighted graph namely $G = (V, E)$, where V is the set of network nodes and E is the set of links connecting network nodes to each other. The link $e \in E$ with source node m and destination node n is denoted by (m, n). Multicast tree which is denoted as $MT(s, M)$ consists of two main parts: $s \in V$ as the source node of multicasting, and $M \subseteq V - \{s\}$ as the multicast destination nodes. Each link e is characterized by a QoS 3-tuple $(B(e), D(e), C(e))$ representing bandwidth, delay and cost associated with it respectively. Here, $B(e) > 0$, $C(e) \geq 0$ and $D(e) \geq 0$. This paper tries to discover a multicast tree with the minimum cost subject to two QoS constraints namely bandwidth, and delay constraints. If $p(s, d_i)$ is a path in the tree MT starting from the source node s and ending at a multicast destination node d_i, then bandwidth and delay constraints and cost function are defined as follows:

- *Bandwidth constraint*
  It is required that the minimum value of the link bandwidth in the multicast tree MT, along the path (B_{path}) originating at the source node s and ending at any multicast destination node $d_i \in M$ be greater than or equal to the predefined required bandwidth Ω_b.

- *Delay Constraint*
  It is required that the end-to-end delay in the multicast tree MT, along the path (D_{path}) originating at the source node s and ending at any multicast destination node $d_i \in M$ be smaller than or equal to the predefined maximum end-to-end delay Ω_d.

- *Cost Function*
  We define the cost of the (multicast) tree as the sum of the costs of all the links of the tree. Formula 1 formalizes it:

$$C_{tree}(MT) = \sum_{e \in MT} C(e) \tag{1}$$

  $C(e)$ typically represents the cost of the link monetarily or any administratively interested cost. The proposed algorithm of this paper aims at constructing a least-cost multicast tree subject to delay and bandwidth constraints defined above. Formalizing this problem as a constrained optimization problem, we have

$$if \quad C_{tree}(MT) = \sum_{e \in MT} C(e) \quad Then \quad Minimize(C_{tree}(MT))$$

$$subject \quad to \quad B_{path}(p(s,d_i)) \geq \Omega_b \quad and \quad D_{path}(p(s,d_i)) \leq \Omega_d$$

3 The proposed multicast routing algorithm

This section explains how GASANT utilizes the cooperation between GA and ACO to produce multicast trees. As shown in Fig. 1, GASANT consists of both reactive and proactive components. The proactive component itself has an ACO Module within. The reactive component is composed of two main modules namely Path Setup (PS) Module and Genetic and Simulated Annealing (GSA) Module.

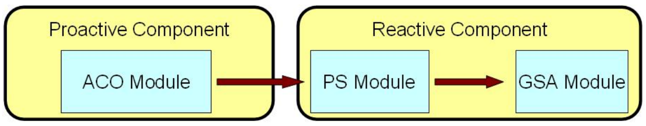


Figure 1: GASANT Architecture

In this paper, a modified version of AntNet [23] has been implemented in ACO Module. This modified version increases AntNet's performance and adapts it to multicast nature of routing. ACO Module proactively sends ants through network in order to find feasible paths to different destinations and keep nodes aware of network dynamism. This process continues periodically during the whole network uptime.

Beside this, the reactive component is responsible for multicast tree construction process. The process is carried out through two modules called PS Module and GSA Module. When a multicast tree creation request with a certain set of QoS criteria is issued by an application, PS Module starts to find paths between the multicast source node and each destination node. This process is done through propagating control messages throughout the network. These messages try to discover feasible paths considering the QoS metrics by fetching the information provided by ACO Module stored in intermediate nodes. Each of these paths is called trunk which is in fact a simple path that connects the source node to a multicast destination node. The result of PS Module is in fact a set of highly crowded paths which were frequently traversed by ants. The discovered paths are fed into the GSA Module. The GSA Module uses these paths as raw data to construct its initial population. It then tries to find a multicast tree through running crossover and mutation operators in iterations. If a multicast tree with required QoS metrics is found, then the mission is complete. Otherwise, GASANT commences a negotiation with the application that triggered the strict request in order to ask it to relax its QoS requirements. In the following sub-sections, we have explained these three modules in more detail.

3.1 ACO Module

At regular intervals, from every network node, ACO sends forward ants toward a destination node in order to discover feasible and satisfactory paths. Multicast destination nodes have higher chance of being selected as forward ant's destination nodes. Analogously to AntNet, ants will travel toward destination node and if successful, then they will return back to source node. While returning back to the source node, ants update routing table of the intermediate nodes en route. This update is based on a reinforcement value r (refer to section 4 of [23] for more detailed information) which is a function of the goodness of the path that the ant has just traversed. In GASANT, the value of r is calculated based on ant's traversed path trip time ($TripTime$), bandwidth ($Bandwidth$) and cost ($Cost$) through Formula 2:

$$r = c_1 \times (\frac{TripTime_{best}}{TripTime}) + c_2 \times (\frac{Bandwidth}{Bandwidth_{best}}) + c_3 \times (\frac{Cost_{best}}{Cost}) \tag{2}$$

Here, $TripTime_{best}$ is the best trip time experienced by the ants, and $Bandwidth_{best}$ and $Cost_{best}$ are the best bandwidth and the best cost discovered while travelling toward same destination over the last observation window (in this paper we considered last 20 samples), respectively. The coefficients c_1, c_2 and c_3 weigh the importance of each term. In our implementation, we have set each of these constants euqally to 0.33. In this way, all three QoS parameters are treated equally. Ants can easily keep track of bandwidth and cost of paths by saving minimum link bandwidth and summing the cost of individual links they are traversing.

3.2 PS Module

The multicast route discovery process triggers upon receiving a multicast tree setup request from a source to a number of destinations. The purpose of this reactive process is to find a set of trunks according to QoS metrics. This is accomplished by actually sending Multicast Tree Discovery (MTD) messages through network. These messages find links with high pheromones which can provide better trunks. In fact, this procedure mainly uses the information provided by ACO Module to find the best single paths to destinations. As soon as a multicast tree setup request is received, the source node starts propagating MTD messages to find feasible and shortest paths to multicast destinations. The source node broadcasts a limited number of MTD messages to its connected neighbors. This number is set through $branchFactor$ variable which is the cardinality of a subset of the node's neighbors. Using this variable, the number of packets being broadcast throughout the network can be limited. Therefore, the traffic overhead caused by MTD messages can be controlled. This subsetting is carried out through choosing the neighbors which are connected via links with highest pheromones to the source node.

Each MTD message is composed of seven fields: 1.$requestID$, 2.*sourcenodeID*, 3.$cost$, 4.*bandwidth*, 5.*delay*, 6.*path*, 7.$branchFactor$. A MTD message stores the traversed path and its corresponding accumulated cost into its *path* and *cost* fields, respectively. The minimum bandwidth and total delay of the path are being kept track in *bandwidth* and *delay* fields. Each MTD message is associated with a *uniquerequestID* field; together with the *sourcenodeID* field, a network node can uniquely identify a MTD message. These two IDs together are considered as the $Identifier(ID)$ pair. When an intermediate node receives an MTD message, the node inspects the message's ID pair to check whether the message is a duplicate. In GASANT, the intermediate node is allowed to forward at most a limited number of duplicated MTD messages (messages with same ID pair) defined by *allowedDupNumber*. It is critical to somehow limit the number of packets being generated and forwarded through network or the network will get inundated with extraneous traffic.

Our experiments show that even with the $branchFactor$ of 2 for a network of size 100 nodes, when the number of passing messages is not limited, more than millions of MTD messages of the same ID are created and passes the same nodes for more than tens of thousands times. Also, our experiments show that values around 50 for the *allowedDupNumber* for networks with more than 100 nodes can significantly reduce the message overhead while still providing vast amount of available trunks from source to destination nodes. After forwarding *allowedDupNumber* number of MTD messages of the same identifier pair, any subsequent MTD messages of this identifier pair is discarded by that intermediate node. To keep track of the number of times a particular MTD message has been forwarded, each node maintains a table of counters for each of the MTD messages the node has forwarded. If the counter of the received MTD message does not violate the *allowedDupNumber*, the node updates some fields of the message before forwarding. The *cost* field is increased by the cost of the link the message has just traversed. Also the current node's address is appended to the *path* list. The *bandwidth* field is set to minimum of the current bandwidth field value and the bandwidth of the traversed link. Once the update is complete, like the source node, this node forwards the MTD message to selected $branchFactor$ number of its neighbors. The selection is based on the pheromone level entries of the probabilistic routing table [23]. The higher values have the higher chance of being selected. The value of $branchFactor$ can significantly affect the amount of messages overhead generated in the network. Higher *branchFactor* values will make messages explore more parts of the network and consequently gather more possible paths to destinations in expense of exponentially increasing message overhead.

In this paper, the *branchFactor* is initially set to a constant large value compared to average degree of the network graph nodes and then it is reduced on next hops. This way most likely all neighbors of the source node will receive the MTD message. As a result, search area is widened and more parts of the network get explored while still having reasonable load. Two logarithmic and linear methods have been tested for reducing the *branchFactor* along the path. In the logarithmic approach, the reduction is fast and *branchFactor* converges to one just after passing a few hops. When *branchFactor* value is one, the node forwards one copy of the received MTD message to its best candidate neighbor. In this way, the *branchFactor* and *allowedDupLimit* together can control the amount of MTD message overhead. A constant *branchFactor* through the whole network is also investigated. However, our experiments show that the logarithmic approach surpasses the other two approaches in controlling the overhead while still producing nearly the same trunks as the others (constant and linear reduction approaches) produce.

When a MTD message reaches its destination node, a reply message containing the same data as the MTD message is generated and is sent back to source node. The reply message traverses the same path, but will be queued in high priority queues. When the first message reaches back to the source node, the source node triggers a timer to collect as many routes as possible from different destinations. As the timeout expires, a set of trunks are extracted from the *path* field of the collected reply messages. These trunks are then sent to the GSA Module.

3.3 GSA Module

In the following subsections, the specifications and operators of the GSA Module are explained in detail.

Coding

The tree structure coding has been chosen in this paper. In this method, every chromosome represents a multicast tree. As a result, the coding space is greatly reduced and coding-decoding operation is omitted and the meaning of genetic operations becomes more visual and the time of conversion between encoding and solution spaces is saved [1].

Pruning

In this phase, those links with a bandwidth less than the predefined bandwidth threshold are deleted. It is probable that the pruned graph gets decomposed into several smaller connected subgraphs. If all the multicasting nodes including the source node are all in the same subgraph, this means that the pruned network satisfies the bandwidth threshold. Otherwise, the source node should start a new round of negotiations with the applicant program in order to reach a mutual satisfactory agreement with more relaxed threshold.

Initial population formation

Initial population is performed in two main stages: trunk-selection and limb-appending.

Trunk-selection: In this stage, one of the trunks created by the ACO Module is selected randomly. A trunk is a simple path that connects the source node to a multicast destination node (as in section 3). This trunk is supposed as the current multicast tree.

Limb-appending: In this stage, a multicast destination node which is not part of the current multicast tree (obtained from previous stage) is selected as the current node. Then, one of the links leaving this current node is selected. The more pheromone deposited on the link, the more probable the link is selected. After this, the other end of the link is selected as the current node, and the same action is done about it. This procedure continues until the current node becomes one of the nodes of the current multicast tree. At this time, all of these nodes which in turn were as current nodes and also the current multicast tree nodes together are set to the new current multicast tree. On the condition that the entire multicast destination nodes are in the current multicast tree, the mission is complete; otherwise the whole limb-appending procedure is repeated for the rest of the multicast destination nodes isolated from the current multicast tree.

Applying selections based on high pheromone values is more efficient than that of [20] which uses a random paradigm. This seems reasonable since the trunk-selection and limb-appending methods use previously gathered global information in making their decisions. The experiments in section 5 also prove this claim.

Fitness Function

Basically, penalty functions are used to handle the constraints [24]. Through these functions the constrained problems are transformed to unconstrained problem. In fact, these functions are used to penalize the individuals based on their constraint violation. The penalty imposed on infeasible individuals can range from completely rejecting the individual to decreasing its fitness based on the degree of violation [16]. In this paper, we have adopted the same fitness function formulae as in [20]. We have utilized penalty function in determining the fitness of each individual. Here, the fitness of the multicast tree *MT* is defined as Formula 3:

$$Fitness(MT) = e^{\frac{-Penalty(MT)}{T}} \tag{3}$$

Where, T is the temperature in which this fitness is calculated and *Penalty* is the amount of penalty that has been considered for MT. The *Penalty* itself is calculated via Formula 4:

$$Penalty(MT) = k_c \times Cost_{tree}(MT) + k_b \times BV(MT) + k_d \times DV(MT) \tag{4}$$

Here, $Cost_{tree}(MT)$ is calculated via formula 1; $BV(MT)$, and $DV(MT)$ are determined via Formulae 5 and 6; k_c , k_b, and k_d are the constants to weigh the importance of each of the cost, and the violations occurred from bandwidth and delay constraints, respectively.

$$BV(MT) = \sum_{d_j \in MT} maximum(\Omega_b - B_{path}(s, d_j), 0) \tag{5}$$

$$DV(MT) = \sum_{d_j \in MT} maximum(D_{path}(s, d_j) - \Omega_d, 0) \tag{6}$$

As it can be understood from all above, the less violation an MT makes (Formulae 5,6), the fewer penalties are considered for it (Formula 4), and consequently, such an MT is fitter to the delay and bandwidth constraints (Formula 3).

Selection Method

In this paper, fitness proportionate selection (roulette wheel selection) has been applied. In this method, the probability of selecting a chromosome C_i as a parent is defined as follows (Formula 7):

$$SelectionProbability = \frac{Fitness(Fitness(C_i))}{\sum_{j=1}^{PopulationSize} Fitness(C_j)} \tag{7}$$

Where $Fitness(C_i)$ is the fitness of chromosome i. In fact, more elitist chromosomes have higher chances to be selected in comparison with their counterparts.

Adaptive Crossover Method

Crossover is a genetic operator that combines two chromosomes called parents to produce a new chromosome namely offspring. Since experiments in [25] indicated that adaptive crossover performed as well or better than a traditional crossover, in this paper, adaptive crossover probability is applied. Here, the crossover probability is calculated through Formula 8:

$$CrossoverProbability = \begin{cases} c_1 + c_2 \times \frac{fit_{max} - fit\prime}{fit_{max} - fit_{avg}} & fit\prime \geq fit_{avg} \\ c_3 & fit\prime < fit_{avg} \end{cases} \tag{8}$$

Here, fit_{avg} is the average of the fitness of the population in the current generation; fit_{max} is the biggest fitness existent in the current generation; fit' is the maximum of the fitnesses associated with the two parents to be crossovered; c_1. c_2, c_3 are constants and $c_3 = c_2 + c_1$.

In this paper, for crossovering two parent trees, somewhat the same method as in [12] has been adopted. For more clarity, the crossover and mutation (next section) processes for a hypothetical network has been illustrated in Fig. 2. Suppose that the network graph is the same as in Fig. 2.a in which node 0 (green node) is the source node and the nodes 2 and 4 are the multicast destination nodes (yellow nodes). Also, assume that Fig. 2.b.1 are the two possible multicast trees. Now, for crossovering, first of all, the common edges of the two parent trees (Fig. 2.b.1) are extracted and are inserted to the offspring tree as the first set of edges (Fig. 2.b.2). The produced offspring is not necessarily a single connected tree and may consist of several disconnected components. In this offspring, it is likely that some of the multicast destination nodes fall into graph components (orphan components) other than the component which the source node is in (main component). Therefore, somehow these orphan components must be connected to the main component. To this end, at first, an orphan component is randomly selected. Then, it is tried to connect this orphan component to the main component through highly pheromone bridge links (links existent in network graph that can connect two different components to each other). While doing this procedure, these selected bridge links may also connect other orphan components to each other. If there may still be orphan components that are remained disconnected, the same procedure should be performed about them. The produced multicast tree for our example is now the same as Fig. 2.b.3. Of course, some extra nodes may appear as the leaf nodes of offspring tree. In fact, these nodes are not member of the multicast tree destinations (node 5 in Fig. 2.b.3); thus, such nodes and their entering links should be removed after crossover (Fig. 2.b.4).

Adaptive Mutation Method

Mutation is a genetic operator that alters one or more gene values in a parent and produces a new offspring. This operator can prevent the population from stagnating at any local optima. In this paper, a simulated annealing technique has been adopted for mutation operator. Regarding this, each new offspring produced by the mutation operator is considered as a neighbor of the initial parent. This new offspring replaces its parent according to a probability calculated in Formula 9:

$$MutationProbability = \begin{cases} e^{\frac{-(fit - fit\prime)}{T}} & fit > fit\prime \\ 1 & fit \leq fit\prime \end{cases} \tag{9}$$

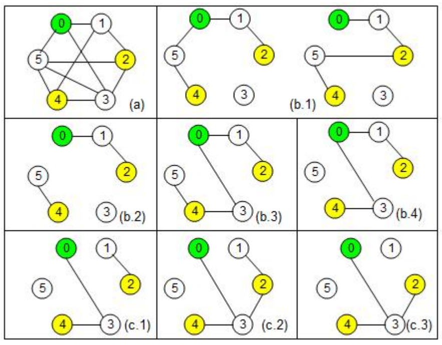


Figure 2: Crossover and Mutation Operations. (a) The hypothetical network graph. (b.1-4) Stages of crossover operator. (c.1-3) Stages of mutation operator.

Here, fit and fit' are the fitnesses associated with the parent and offspring, respectively, and T is the temperature. As can be inferred from Formula 9, if the offspring is better than the parent, it replaces the parent in the next generation, otherwise this replacement is done through an exponential probability. The temperature decreases gradually from a high initial degree. As this decreasing continues, the probability of the replacement of worse offspring decreases accordingly. Continuing our example, suppose that we want to apply mutation operator for Fig. 2.b.4. Mutation for a parent tree (Fig. 2.b.4) which leads to finding a neighbor offspring tree is done through the following procedure: a random edge is deleted from the parent tree (edge (0,1) from Fig. 2.b.4 is deleted). Then, it is tried to reconnect these two subtrees (Fig. 2.c.1) with the same procedure explained in section 3.3. This new connected tree (Fig. 2.c.2) is considered as the offspring (neighbor) of the parent tree. This offspring replaces its parent according to a probability calculated in formula 9. The same as the crossover, some extra links may appear as the leaf nodes of offspring tree. In fact, these nodes are not member of the multicast tree destinations (node 1 in Fig. 2.c.2); thus, as mentioned in crossover, they should not be included in the final multicast tree, and as a result Fig. 2.c.3 is the real neighbor of the Fig. 2.b.4.

4 Analysis of Convergence

In Theorem 2.7 of [26], Guoliang et al. has shown that applying GA can finally lead the problem to converge to a global optimized solution. Achieving an optimal solution for the aforementioned multicast QoS routing can sometimes take up a great deal of time. This is due to the NP-completeness property of the problem. Nevertheless, most of the times, achieving a solution is possible by well-adjusting various parameters in the proposed algorithm.

5 Experiments

GASANT has been implemented in C++. The ACO Module of GASANT has been implemented by extending and modifying the AntNet package provided in [27]. The whole program is simulated in $OMNeT++$ environment on a Pentium IV 2.4 GHz CPU, 2 GB RAM. To make the results independent of the networks under simulation and also in order to run experiments on a reasonable amount of graphs, we have used random graph generator introduced by Waxman [28]. In this method, the network nodes are randomly scattered in a rectangular environment. The probability of existence of an edge between two nodes u, v is calculated via Formula (10):

$$P(u, v) = \alpha e^{\frac{-d(u,v)}{\beta L}} \tag{10}$$

Here, $d(u, v)$ is the distance between two nodes u and v; L is the maximum distance between any two nodes in the generated graph. $\alpha \in (0, 1]$ is responsible for controlling the average degree of the random graph. The greater α results in a denser graph; a graph with more links. In the same way but rather different, $\beta \in (0, 1]$ effects the number of longer edges. The larger values of β increase the number of longer edges. Also, the randomly generated edges of the graph accepts their cost from [1,100], delay from [0.01,0.1] ms, and bandwidth values from range [10,50] Mbps. GASANT was run on 20, 40, 60, 80, and 100-node network graphs. Also, three different percentages of network nodes were selected to be multicast destinations in the runs: 10%, 20%, and 30%. We call these percentages as multicast percentages, or briefly mp. For the sake of convenience, the delay and bandwidth bounds are supposed to be the same for all multicast destinations. Besides, GSA module of GASANT iterated for 15 generations, and the population size of each of these generations was 30. All the experiments were run until we reach a confidence interval of less than 5%, using 95% confidence level. Before delving into the experiements, we need to define routing request *Success Ratio* [29] which is defined as Formula 11:

$$SuccessRatio = N_{ack}/N_{req} \tag{11}$$

where the N_{req} is the number of multicast tree requests issued by the application, and N_{ack} is the number of these requests that are successfully answered. In the next subsections, we compare GASANT and NGSA through a comprehensive set of experiments.

5.1 Success Ratio Analysis

In this paper, we have compared GASANT to NGSA [20] which is one of the recent works in QoS multicast routing literature. Table 1 illustrates the *Success Ratio* of GASANT to NGSA as a function of different parameters.

Table 1 displays the *Success Ratio* of GASANT to NGSA as a function of network size and multicast percentage (mp). Increasing multicast percentage and network size usually results in higher $SuccessRatios$ of GASANT to NGSA. This is due to the fact that as network size or multicast percentage increases, applying random paradigm used in trunk creation and limb appending is more likely to adopt improper links toward multicast destination nodes. This figure can be regarded as a scalability performance, too. Table 1 and Table 1 illustrate the *Success Ratios* of GASANT to NGSA as functions of minimum possible bandwidth and maximum possible end-to-end delay constrained on the problem, respectively. As it can be inferred from these figures, independent from these constraints, the *Success Ratio* of GASANT is most often higher than NGSA's one.

Table 1: (a) *Success Ratio* of GASANT To NGSA as a function of network size and multicast percentage. (b) *Success Ratio* of GASANT To NGSA as a function of minimum possible bandwidth.(c) *Success Ratio* of GASANT To NGSA as a function of maximum possible end-to-end delay.

Net. Size	mp=0.1	mp=0.2	mp=0.3
20	0.83	1	1
40	0.78	1.05	1.2
60	0.83	1.15	1.25
80	0.96	1.05	1.08
100	1	1.25	1.66

Min. BW.	Success R.
10	1.1
15	1.2
20	1.23
25	1.22
30	1.17
35	1.02
40	1.07

Max. Delay	Success R.
0.10	1.4
0.20	1.0
0.30	1.5
0.40	1.8
0.50	1.22
0.60	0.81
0.70	1.04

Table 2: (Part 1) Average time ratio of GASANT to NGSA for detecting the first satisfactory multicast tree. (Part 2) Average cost ratio of GASANT to NGSA for the first detected satisfacroty multicast tree. (Part 3) Average maximum end-to-end delay ratio of GASANT to NGSA for the first detected satisfacroty multicast tree. (Part 4) Average minimum bandwidth ratio of GASANT to NGSA for the first detected satisfacroty multicast tree.

	Part	1			2			3			4		
Network Size	mp	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3
20		0.11	0.16	0.49	0.78	0.91	0.89	0.57	0.80	0.85	1.12	0.98	1.04
40		0.14	0.43	0.01	0.80	0.97	0.72	0.67	0.93	0.78	0.99	0.98	1.00
60		0.08	0.01	0.54	0.78	0.80	0.85	0.72	0.81	0.93	1.02	1.07	1.00
80		0.26	0.01	0.13	0.83	0.89	0.82	0.85	0.82	0.97	0.98	1.02	1.01
100		0.74	0.67	0.06	0.87	0.92	0.66	0.96	0.76	0.91	1.08	1.01	0.95

5.2 Comparison of Execution Times, Quality of Solutions and Generations

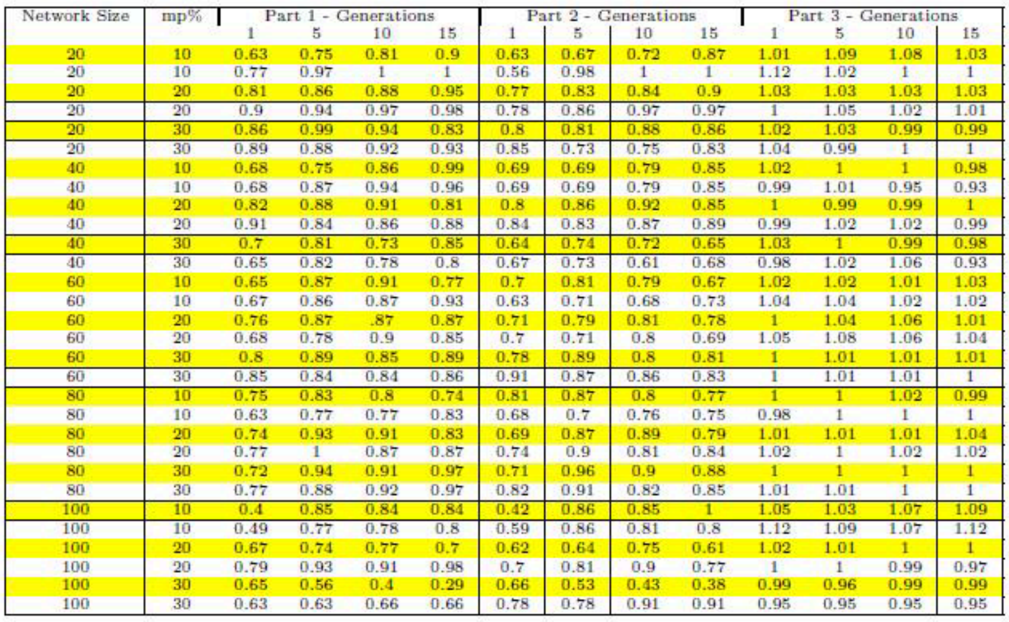
Part 1 of the Table 2 illustrates the average time ratio of GASANT to NGSA for detecting the first satisfactory multicast tree. As can be seen, the average required time for achieving the first solution by GASANT is almost less than half of the time required by NGSA. The average cost and average maximum end-to-end delay associated with the first solution discovered by GASANT, as shown in Parts 2 and 3 of the Table 2, are always less than that of NGSA's. Also, Part 4 of the Table 2 demonstrates that the average minimum existent bandwidth is nearly the same for both multicast trees discovered by GASANT and NGSA. Part 1 of Table Table 3 illustrates the average cost ratio of GASANT to NGSA for every fifth generation (yellow rows) and the average cost ratio of GASANT to NGSA for the least cost multicast tree existent in that generation (white rows) as a function of network size and multicast percentage.

As it is obvious, the cost of the trees in each generation associated with GASANT is less than that of NGSA's. Also, the least cost trees existent in each generation of GASANT has smaller cost in comparison with NGSA's. As can be inferred from this part of the table, ACO Module helps GSA start the production of generations with smaller cost trees and this continues along the rest of the generations till end.

Part 2 of Table Table 3 illustrates the average maximum end-to-end delay ratio of GASANT to NGSA for each generation (yellow rows) and the average maximum end-to-end delay ratio of GASANT to NGSA associated with the least cost multicast tree existent in that generation (white rows) as a function of network size and multicast percentage. Part 3 of Table Table 3 illustrates the average minimum bandwidth ratio of GASANT to NGSA for each generation (yellow rows) and the average minimum bandwidth ratio of GASANT to NGSA associated with the least cost multicast tree existent in that generation (white rows) as a function of network size and multicast percentage.

All the parts of the Table Table 3 illustrate that the QoS of multicast trees discovered by GASANT is superior to the trees discovered by NGSA by having smaller cost, end-to-end delay and nearly the same bandwidth. It can be inferred from this table that ACO module donates a more global view of the network to GASANT in comparison with the NGSA which only relies on bare GA and SA.

Table 3: (Part 1)Yellow rows: Average cost ratio of GASANT to NGSA for every fifth generation as a function of network size and multicast percentage. White rows: Average cost ratio of the least cost multicast tree for every fifth generation as a function of network size and multicast percentage. (Part 2) Yellow rows: Average maximum end-to-end delay ratio of GASANT to NGSA for every fifth generation as a function of network size and multicast percentage. White rows: Average maximum end-to-end delay ratio of the least cost multicast tree for every fifth generation as a function of network size and multicast percentage. (Part 3) Yellow rows: Average minimum bandwidth ratio of GASANT to NGSA for every fifth generation as a function of network size and multicast percentage. White rows: Average minimum bandwidth ratio of the least cost multicast tree for every fifth generation as a function of network size and multicast percentage.



Network Size	mp%	Part 1 - Generations				Part 2 - Generations				Part 3 - Generations			
		1	5	10	15	1	5	10	15	1	5	10	15
20	10	0.63	0.75	0.81	0.9	0.63	0.67	0.72	0.87	1.01	1.09	1.08	1.03
20	10	0.77	0.97	1	1	0.56	0.98	1	1	1.12	1.02	1	1
20	20	0.81	0.86	0.88	0.95	0.77	0.83	0.84	0.9	1.03	1.03	1.03	1.03
20	20	0.9	0.94	0.97	0.98	0.78	0.86	0.97	0.97	1	1.05	1.02	1.01
20	30	0.86	0.99	0.94	0.83	0.8	0.81	0.88	0.86	1.02	1.03	0.99	0.99
20	30	0.89	0.88	0.92	0.93	0.85	0.73	0.75	0.83	1.04	0.99	1	1
40	10	0.68	0.75	0.86	0.99	0.69	0.69	0.79	0.85	1.02	1	1	0.98
40	10	0.68	0.87	0.94	0.96	0.69	0.69	0.79	0.85	0.99	1.01	0.95	0.93
40	20	0.82	0.88	0.91	0.81	0.8	0.86	0.92	0.85	1	0.99	0.99	1
40	20	0.91	0.84	0.86	0.88	0.84	0.83	0.87	0.89	0.99	1.02	1.02	0.99
40	30	0.7	0.81	0.73	0.85	0.64	0.74	0.72	0.65	1.03	1	0.99	0.98
40	30	0.65	0.82	0.78	0.8	0.67	0.73	0.61	0.68	0.98	1.02	1.06	0.93
60	10	0.65	0.87	0.91	0.77	0.7	0.81	0.79	0.67	1.02	1.02	1.01	1.03
60	10	0.67	0.86	0.87	0.93	0.63	0.71	0.68	0.73	1.04	1.04	1.02	1.02
60	20	0.76	0.87	.87	0.87	0.71	0.79	0.81	0.78	1	1.04	1.06	1.01
60	20	0.68	0.78	0.9	0.85	0.7	0.71	0.8	0.69	1.05	1.08	1.06	1.04
60	30	0.8	0.89	0.85	0.89	0.78	0.89	0.8	0.81	1	1.01	1.01	1.01
60	30	0.85	0.84	0.84	0.86	0.91	0.87	0.86	0.83	1	1.01	1.01	1
80	10	0.75	0.83	0.8	0.74	0.81	0.87	0.8	0.77	1	1	1.02	0.99
80	10	0.63	0.77	0.77	0.83	0.68	0.7	0.76	0.75	0.98	1	1	1
80	20	0.74	0.93	0.91	0.83	0.69	0.87	0.89	0.79	1.01	1.01	1.01	1.04
80	20	0.77	1	0.87	0.87	0.74	0.9	0.81	0.84	1.02	1	1.02	1.02
80	30	0.72	0.94	0.91	0.97	0.71	0.96	0.9	0.88	1	1	1	1
80	30	0.77	0.88	0.92	0.97	0.82	0.91	0.82	0.85	1.01	1.01	1	1
100	10	0.4	0.85	0.84	0.84	0.42	0.86	0.85	1	1.05	1.03	1.07	1.09
100	10	0.49	0.77	0.78	0.8	0.59	0.86	0.81	0.8	1.12	1.09	1.07	1.12
100	20	0.67	0.74	0.77	0.7	0.62	0.64	0.75	0.61	1.02	1.01	1	1
100	20	0.79	0.93	0.91	0.98	0.7	0.81	0.9	0.77	1	1	0.99	0.97
100	30	0.65	0.56	0.4	0.29	0.66	0.53	0.43	0.38	0.99	0.96	0.99	0.99
100	30	0.63	0.63	0.66	0.66	0.78	0.78	0.91	0.91	0.95	0.95	0.95	0.95

Table 3: (Part 1) Edge hit ratio of GASANT to NGSA for detecting the first satisfactory multicast tree. (Part 2) Average Overhead Analysis of ACO Module.

	Part 1			Part 2
Network Size	mp=0.1	mp=0.2	mp=0.3	Total Consumed Bandwidth (Kbps)
20	0.04	0.12	0.26	0.12
40	0.14	0.28	0.01	0.28
60	0.08	0.16	0.67	0.45
80	0.23	0.24	0.07	0.62
100	0.48	0.33	0.09	0.88

5.3 Analysis of Reduction of Search Process

Part 1 of Table 3 illustrates edge hit frequency ratio of GASANT to NGSA during the detection of the first satisfactory multicast tree. Edge hit frequency is the number of times that the edge is visited by the multicast tree discovery algorithm during trunk creation or limb appending phases. As can be seen, edge hit ratio of GASANT is considerably smaller than NGSA's. This implies that GASANT visits the edges of the network graph sufficiently not excessively and redundantly. This redundancy can be a symptom of high try and error nature of NGSA before achieving a satisfactory multicast tree. Whereas, GASANT has a global view of the network specific properties. Thus, with a smaller effort, it gains a solution even with higher QoS. Therefore, GASANT discovers a satisfactory solution with a smaller search process. GASANT owes its success to non-stoping efforts of diligent ants of the proactive ACO Module.

5.4 Overhead Analysis of ACO Module

Part 2 of Table 3 illustates the total bandwidth consumed by control messages of ACO module as a function of network size. As can be seen, by increasing network size the amount of overhead increases almost linearly. This implies that the proactive ACO module is scalable as the network size grows.

6 Conclusion

In this paper, we have proposed GASANT to solve the NP-Complete problem of finding a QoS constrained least-cost multicast tree for a given communication network. To this end, we have combined ACO, GA and SA together to utilize the full advantages of them. Here, ACO is responsible for both improved initial population which are fed into GA, and also reduced search process. By improved initial population, we mean that links constructing initial population are more likely to appear in optimal multicast tree. By reduced search process, we mean that the GA visits the network edges for a small number of times rather than moving back and forth on the same and previously-visited edges for many times. For fleeing from standing still around a local optimal solution and also extending search space SA has been deployed. Experiments show that ants in ACO provides a valuable approach of exploring the network and collecting information about states of the links in an efficient and deliberate way. In this way, GA is fed by a better initial population in its first step. Also, it considers the links that are highly recommended by ants to be found more likely in the final multicast tree and consequently reaches to a satisfactory solution sooner. The experiments in this paper ensure the aforementioned claims, and prove that GASANT outperforms its close counterpart NGSA.

Bibliography

[1] W. Zhengying, S. Bingxin, Z. Erdun, Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm Computer Communications, Volume 24, Issues 7-8, April 2001.

[2] F.K. Hwang, D.S. Richards, P. Winter, The Steiner Tree Problem, Annals of Discrete Mathematics, vol. 53, 1992.

[3] R. M. Karp, Reducibility Among Combinatorial Problems, Complexity of Computer Computations, 1972.

[4] V. P. Kompella, J. Pasquale, G. C. Polyzos, Multicast routing for multimedia communication IEEE/ACM Transactions on Networking. 1(3) (1993) 286-292.

[5] M. Parsa, Q. Zhu, J.J. Garcia-Luna-Aceves, An iterative algorithm for delay-constrained minimum-cost multicasting, IEEE/ACM Transactions on Networking 6 (4) 1998.

[6] R. Widyono, The design and evaluation of routing algorithms for realtime channels, Technical Reports TR-94-024, Tenet Group, Department of EECS, University of California at Berkeley, 1994.

[7] A.G. Waters, A new heuristic for ATM multicast routing, 2nd IFIP Workshop on Performance Modeling and Evaluation of ATM networks, 1994.

[8] Q. Sun, H. Langendorfer, An efficient delay-constrained multicast routing algorithm, Journal of High-Speed Networks 7(1) 1998 43-55.

[9] H.F. Salama, D.S. Reeves, Y. Viniotis, Evaluation of multicast routing algorithms for real-time communication on high-speed networks, IEEE Journal on Selected Areas in Communications 15(3) (1997) 332-345.

[10] F. Xiang, L. Junzhou, W. Jieyi, G. Guanqun, QoS routing based on genetic algorithm, Computer Communications 22(15-16) (1999) 1392-1399.

[11] A.T. Haghighat, K. Faez, M. Dehghan, A. Mowlaei, Y. Ghahremani, GA-Based Heuristic Algorithms for QoS Based Multicast Routing, knowledge-based systems 16 (2003) 305-312.

[12] C.P. Ravikumar, R. Bajpai, Source-based delay-bounded multicasting in multimedia networks, Computer Communications 21 (1998) 126-132.

[13] Z. Wang, B. Shi, E. Zhao, Bandwidth-delay-constrainted least-cost multicast routing based on heuristic genetic algorithm, Computer Communications 24 (2001) 685-692.

[14] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, 2003.

[15] Q. Zhang, Y.W. Leung, An orthogonal genetic algorithm for multimedia multicast routing, IEEE Trans. Evol. Comput. 3 (1) (1999) 53-62.

[16] K. Vijayalakshmi, and S. Radhakrishnan, Dynamic Routing to Multiple Destinations in IP Networks Using Hybrid Genetic Algorithm (DRHGA), , International Journal of Information Technology. 4(1) 2008.

[17] Vijayalakshmi ,S. Radhakrishnan, Artificial immune based hybrid GA for QoS based multicast routing in large scale networks (AISMR), Computer communications, 2008.

[18] N. Shimamoto, A. Hiramatsu, K. Yamasaki, A dynamic routing control based on a GA, In proceedings of the IEEE International Conference on Neural Network (1993) pp. 1123-1128.

[19] J.J. Wu, R.H. Hwang, H.I. Lu, Multicast routing with multiple QoS constraints in ATM networks, Information Sciences 124 (2000) 29-57.

[20] Li Zhang, Lian-bo Cai, Meng Li, Fa-hui Wang, A method for least-cost QoS least-Cost multicast routing based on genetic simulated annealing algorithm, Computer Communications, 31 (2008) 3984-3994.

[21] Sushil J. Louis Gregory J. E. Rawlins, Predicting Convergence Time for Genetic Algorithms, Technical Report, Indiana University, 1992.

[22] R. R. Hill, Ch. Hiremath, Improving genetic algorithm convergence using problem structure and domain knowledge in multidimensional knapsack problems, International Journal of Operational Research 2005 - Vol. 1, No.1/2 pp. 145 - 159.

[23] G. Di Caro, M. Dorigo, AntNet: Distributed Stigmergetic Control for Communications Networks, Journal of Artificial Intelligence Research 9 (1998) 317-365.

[24] Özgür Yeniay, Penalty Function Methods for Constrained Optimization with Genetic Algorithms, Journal of Mathematical and Computation Applications, 10(1) (2005) pp. 45-56.

[25] J. D. Schaffer, A. Morishima, An Adaptive Crossover Distribution Mechanism for Genetic Algorithms, ICGA 1987.

[26] C. Guoliang, W. Xufa, Z. Zhenquan, Genetic Algorithm and its Application, People's Posts and Telecommunications Press, 1996.

[27] M. Farooq, Implementation of AntNet in OMNeT++, http://www.omnetpp.org/component/content/article/9-software/3559, November 2009.

[28] B.M.Waxman, Routing of multipoint connections. IEEE J. Select. Areas Commun. 6(9) (1998) 1617-1622.

[29] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science, New Series, Vol. 220, No. 4598 (1983) pp. 671-680.