Implementation of Approximation Algorithms for the Multicast Congestion Problem

Qiang Lu[1,⋆] and Hu Zhang[2,⋆⋆]

[1] College of Civil Engineering and Architecture,
Zhejiang University, Hangzhou 310027, China
qlu66@zju.edu.cn

[2] Department of Computing and Software, McMaster University,
1280 Main Street West, Hamilton,
Ontario L8S 4K1, Canada
zhanghu@mcmaster.ca

Abstract. We implement the approximation algorithm for the multicast congestion problem in communication networks in [14] based on the fast approximation algorithm for packing problems in [13]. We use an approximate minimum Steiner tree solver as an oracle in our implementation. Furthermore, we design some heuristics for our implementation such that both the quality of solution and the running time are improved significantly, while the correctness of the solution is preserved. We also present brief analysis of these heuristics. Numerical results are reported for large scale instances. We show that our implementation results are much better than the results of a theoretically good algorithm in [10].

1 Introduction

We study the *multicast congestion problem* in communication networks. In a given communication network represented by an undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, each vertex v represents a processor, which is able to receive, duplicate and send data packets. A *multicast request* is a set $S \subseteq V$ of vertices (called *terminals*) which are to be connected such that they can receive copies of the same data packet from the source simultaneously. To fulfil a request

⋆ This work was done in part when this author was visiting the University of Kiel. Research supported in part by a scholar visiting program between the University of Kiel and Zhejiang University, by DGET, Zh0446, WAARD, G10444, grant of Zhejiang University.

⋆⋆ This work was done in part when this author was studying at the University of Kiel, Germany. Research supported in part by the DFG Graduiertenkolleg 357, Effiziente Algorithmen und Mehrskalenmethoden, by EU Thematic Network APPOL II, Approximation and Online Algorithms for Optimization Problems, IST-2001-32007, by EU Project CRESCCO, Critical Resource Sharing for Cooperation in Complex Systems, IST-2001-33135, by an MITACS grant of Canada, by the NSERC Discovery Grant DG 5-48923, and by the Canada Research Chair program.

© Springer-Verlag Berlin Heidelberg 2005

S, one subtree T in G is to be generated for spanning S, called an S-tree. In the multicast congestion problem in communication networks we are given a graph G and a set of multicast requests $S_1, \ldots, S_k \subseteq V$. A feasible solution is a set of k trees $T_1, \ldots, T_k$, where T_q connects the terminals in S_q, called an S_q-tree. The *congestion* of an edge in a solution is the number of S_q-trees which use the edge. The goal of the problem is to find a solution of S_q-trees for all $q = 1, \ldots, k$ that minimizes the maximum edge congestion.

If each request consists of only two terminals, the multicast congestion problem is reduced to the standard routing problem of finding integral paths with minimum congestion. In fact it is a generalization of the problem of finding edge disjoint shortest paths for source and destination pairs. This problem is $\mathcal{NP}$-hard [15] and hence the multicast congestion problem is also $\mathcal{NP}$-hard.

Another related problem is the *Steiner tree problem* in graphs. Given a graph $G = (V, E)$, a set $S \subseteq V$ of terminals and a non-negative length function (cost or weight) on the edges, a *Steiner tree* T is a subtree spanning all vertices in S. The vertices of T may be in $V \setminus S$. The goal of the Steiner tree problem in graphs is to find a minimum Steiner tree, i.e., a Steiner tree with minimum total edge length. Compared with the multicast congestion problem, in the Steiner tree problem there is only a single multicast and the objective function is different. However, the Steiner tree problem is proved $\mathcal{APX}$-hard [15, 1, 5]:

Proposition 1. *The Steiner tree problem in graphs is $\mathcal{NP}$-hard, even for unweighted graphs. Furthermore, there exists a constant $\bar{c} > 1$ such that there is no polynomial-time approximation algorithm for the Steiner tree problem in graphs with an approximation ratio less than $\bar{c}$, unless $\mathcal{P} = \mathcal{NP}$.*

The best known lower bound is $\bar{c} = 96/95 \approx 1.0105$ [8].

Since the multicast congestion problem is $\mathcal{NP}$-hard, interests turn to approximation algorithms. In [20] a routing problem in the design of a certain class of VLSI circuits was studied as a special case of the multicast congestion problem. The goal is to reduce the maximum edge congestion of a two-dimensional rectilinear lattice with a specific set of a polynomial number of trees. By solving the relaxation of the integer linear program and applying randomized rounding, a randomized algorithm was proposed such that the congestion is bounded by $\mathcal{OPT} + O(\sqrt{\mathcal{OPT}} \ln(n^2/\varepsilon))$ with probability $1 - \varepsilon$ when $\mathcal{OPT}$ is sufficiently large, where $\mathcal{OPT}$ is the optimal value. Vempala and Vöcking [22] proposed an approximation algorithm for the multicast congestion problem. They applied a separation oracle and decomposed the fractional solution for each multicast into a set of paths. An $O(\ln n)$-approximate solution can be delivered in time $O(n^6\alpha^2 + n^7\alpha)$ by their algorithm, where α involves the number k and some other logarithmic factors. Carr and Vempala [6] proposed a randomized asymptotic algorithm for the multicast congestion problem with a constant approximation ratio. They analyzed the solution to the linear programming (LP) relaxation by the ellipsoid method, and showed that it is a convex combination of S_i-trees. By picking a tree with probability equal to its convex multiplier, they obtained a solution with congestions bounded by $2\exp(1)c \cdot \mathcal{OPT} + O(\ln n)$ with probability at least $1 - 1/n$, where $c > 1$ is the approximation ratio of the approximate minimum Steiner tree solver. The algorithm needs $\tilde{O}(n^7)$ time including k as a multiplication factor. Without awareness of above theoretical results, Chen et al. [7] studied this problem from practical point of view, which was called *multicast packing problem* in their paper. They showed some lower bounds for the problem and implemented some instances with small sizes by the branch-and-cut algorithm. More works on the multicast packing problem can be found in [18].

Baltz and Srivastav [3] studied the multicast congestion problem and proposed a formulation based on the ideas of Klein et al. [16] for the concurrent multicommodity flow problem with uniform capacities. The integer linear program has an exponential number of variables and they constructed a combinatorial LP-algorithm to obtain a polynomial number of S_q-trees for each multicast request S_q. Finally a randomized rounding technique in [19] was applied. The solution of their algorithm is bounded by

$$\begin{cases} (1+\varepsilon)c \cdot \mathcal{OPT} + (1+\varepsilon)(\exp(1)-1)\sqrt{c \cdot \mathcal{OPT} \ln m}, & \text{if } c \cdot \mathcal{OPT} \geq \ln m, \\ (1+\varepsilon)c \cdot \mathcal{OPT} + \dfrac{(1+\varepsilon)\exp(1)\ln m}{1+\ln(\ln m/(c \cdot \mathcal{OPT}))}, & \text{otherwise.} \end{cases} \tag{1}$$

In the case $c \cdot \mathcal{OPT} \geq \ln m$ the bound is in fact $(1+\varepsilon)\exp(1)c \cdot \mathcal{OPT}$ and otherwise it is $(1+\varepsilon)c \cdot \mathcal{OPT} + O(\ln m)$. The running time is $O(\beta n k^3 \varepsilon^{-9} \ln^3(m/\varepsilon) \cdot \min\{\ln m, \ln k\})$, where β is the running time of the approximate minimum Steiner tree solver. A randomized asymptotic approximation algorithm for the multicast congestion problem was presented in [14]. They applied the fast approximation algorithm for packing problems in [13] to solve the LP relaxation of the integer linear program in [3]. They showed that the block problem is the Steiner tree problem. The solution hence is bounded by (1) and the running time is improved to $O(m(\ln m + \varepsilon^{-2}\ln \varepsilon^{-1})(k\beta + m \ln\ln(m\varepsilon^{-1})))$. Baltz and Srivastav [4] further proposed an approximation algorithm for the multicast congestion problem based on the algorithm for packing problems in [10], which has the best known complexity $O(k(m+\beta)\varepsilon^{-2}\ln k \ln m)$. They also conducted some implementation with typical instances to explore the behaviour of the algorithms. It was reported that the algorithm in [10] is very impractical. In addition, they presented a heuristic based on an online algorithm in [2], which can find good solutions for their test instances within a few iterations.

In this paper we implement the algorithm in [14] with large scale instances. We design some heuristics to speed up the computation and to improve the quality of solution delivered in our implementation. We also present brief analysis of the heuristics. The numerical results show that the algorithm for packing problems [13] is reliable and practical. We also compare our results with those by the algorithm in [10] and the heuristic in [4]. Because other algorithms mentioned above are very impractical, we do not consider them for implementation.

The paper is organized as follows: In Section 2 the approximation algorithm for the multicast congestion problem in [14] is briefly reviewed. We analyze the technique to overcome the hardness of exponential number of variables in Section 3. Our heuristics are presented in Section 4. Finally, numerical results are reported in Section 5 with comparison with other approaches.

2 Approximation Algorithm

Let $\mathcal{T}_q$ be the set of all S_q-trees for any $q \in \{1, \ldots, k\}$. Here the cardinality of $\mathcal{T}_q$ may be exponentially large. Define by $x_q(T)$ a variable indicating whether the tree $T \in \mathcal{T}_q$ is chosen in a solution for the multicast request S_q. Based on the idea in [3, 4], the following integer linear program can be formulated:

$$\begin{array}{lll} \min \lambda & & \\ \text{s.t.} \sum_{q=1}^{k} \sum_{T \in \mathcal{T}_q \ \& \ e_i \in T} x_q(T) \le \lambda, & \text{for all } i \in \{1, \ldots, m\}; & \\ \quad \sum_{T \in \mathcal{T}_q} x_q(T) = 1, & \text{for all } q \in \{1, \ldots, k\}; & \\ \quad x_q(T) \in \{0, 1\}, & \text{for all } q \text{ and all } T \in \mathcal{T}_q, & \end{array} \tag{2}$$

where λ is the maximum congestion. The first set of constraints show that the congestion on any edge is bounded by λ, and the second set of constraints indicate that exact one Steiner tree is chosen for one request. As usual, the strategy is to first solve the LP relaxation of (2) and then round the fractional solution to a feasible solution.

We define a vector $x_q = (x_q(T_1), x_q(T_2), \ldots)^T$ for all $T_1, T_2, \ldots \in \mathcal{T}_q$ representing the vector of indicator variables corresponding to all Steiner trees for the q-th request. Denote by a vector $x = (x_1^T, \ldots, x_k^T)^T$ the vector of all indicator variables. Furthermore, a vector function $f(x) = (f_1(x), \ldots, f_m(x))^T$ is used, where $f_i(x) = \sum_{q=1}^{k} \sum_{T \in \mathcal{T}_q \ \& \ e_i \in T} x_q(T)$ represents the congestion on edge e_i, for $i \in \{1, \ldots, m\}$. In addition, we define by $B = B_1 \times \ldots \times B_k$ where $B_q = \{(x_q(T))^T | T \in \mathcal{T}_q, \sum_{T \in \mathcal{T}_q} x_q(T) = 1, x_q(T) \ge 0\}$, for $q \in \{1, \ldots, k\}$. It is obvious that $x_q \in B_q$ and $x \in B$. In this way the LP relaxation of (2) is formulated as the following *packing problem* (the linear case of the *min-max resource sharing problems* [12, 24, 13]): $\min\{\lambda | f(x) \le \lambda, x \in B\}$. Thus we are able to use the approximation algorithm for packing problems [13] to solve the LP relaxation of (2).

The computational bottleneck lies on the exponential number of variables $x_q(T)$ in (2). The algorithm for packing problems in [13] is employed in [14] with a column generation technique implicitly applied. We briefly describe the algorithm as follows. The algorithm is an iterative method. In each iteration (coordination step) there are three steps. In the first step a *price vector* w is calculated according to current iterate x. Then an approximate block solver is called as an oracle to generate an approximate solution $\hat{x}$ corresponding to the price vector w in the second step. In the third step the iterate is moved to $(1 - \tau)x + \tau\hat{x}$ with an appropriate step length $\tau \in (0, 1)$. The coordination step stops when any one of two stopping rules holds with respect to an relative error tolerance σ, which indicates that the resulting iterate is a $c(1 + \sigma)$-approximate solution. Scaling phase strategy is applied to reduce the coordination complexity. In the first phase $\sigma = 1$ is set. When a coordination step stops, current phase finishes and σ is halved to start a new phase, until $\sigma \le \varepsilon$. Finally the delivered solution fulfils $\lambda(x) \le c(1 + \varepsilon)\lambda^*$, where λ^* is the optimum value of the LP relaxation of (2) (See [13, 14]).

The block problem is exactly the Steiner tree problem in graphs and the edge length function is the price vector w [14]. So k minimum Steiner trees are computed corresponding to the k requests $S_1, \ldots, S_k$ with respect to the length function in current iteration. In the iterative procedure lengths on the edges with large congestions increase while edges with small congestions have decreasing lengths. In this way the edges with large congestions are punished and have low probability to be selected in the generated Steiner trees. The best known algorithm for the Steiner tree problem has an approximation ratio $c = 1 + (\ln 3)/2 \approx 1.550$ [21] but the complexity is large. So in our implementation, we use a 2-approximate minimum Steiner tree solver ($\mathcal{MSTS}$) as the block solver, and its time complexity is $O(m + n \ln n)$ [17, 9]. We call this algorithm $\mathcal{MC}$ and its details can be found in [13, 14]. Then the following result holds [13, 14]:

Theorem 1. *For a given relative accuracy* $\varepsilon \in (0, 1)$*, Algorithm* $\mathcal{MC}$ *delivers a solution* x *such that* $\lambda(x) \leq c(1 + \varepsilon)\lambda^*$ *in* $N = O(m(\ln m + \varepsilon^{-2} \ln \varepsilon^{-1}))$ *iterations. The overall complexity of Algorithm* $\mathcal{MC}$ *is* $O(m(\ln m + \varepsilon^{-2} \ln \varepsilon^{-1})(k\beta + m \ln \ln(m\varepsilon^{-1})))$*, where* β *is the complexity of the approximate minimum Steiner tree solver.*

3 The Number of Variables

In the LP relaxation of (2), there can be an exponential number of variables. However, with the algorithm in [13, 14], a column generation technique is automatically applied and totally the trees generated by the algorithm is a polynomial size subset of $\mathcal{T} = \cup_{q=1}^{k} \mathcal{T}_q$.

If a Steiner tree $T_{q_j} \in \mathcal{T}_q$ is chosen for a request S_q, the corresponding indicator variable is set to $x_{q_j} = 1$. In the fractional sense, it represents the probability to choose the corresponding Steiner tree T_{q_j}. For any tree $T_{q_j} \in \mathcal{T}_q$ for a request S_q, if it is not generated by $\mathcal{MSTS}$ in any iteration of Algorithm $\mathcal{MC}$, then the corresponding indicator variable $x_{q_j} = 0$, which shows that it will never be chosen. Because in each iteration, there are k Steiner trees generated for the k requests, respectively, we conclude that there are only polynomially many trees generated in Algorithm $\mathcal{MC}$ according to Theorem 1:

Theorem 2. *When Algorithm* $\mathcal{MC}$ *halts, there are only* $O(km(\ln m + \varepsilon^{-2} \ln \varepsilon^{-1}))$ *non-zero indicator variables of the vector* x *and only the same number of Steiner trees generated.*

In our implementation, we maintain a vector x with a size $k(N + 1)$, where N is the actual number of iterations. We also maintain a set $\mathcal{T}$ of Steiner trees generated in the algorithm. Notice that here $\mathcal{T}$ is not the set of all feasible Steiner trees. At the beginning the set $\mathcal{T}$ is empty and all components of x are zeros. In the initialization step, k Steiner trees are generated. Then the first k components of x are all ones and the corresponding generated k Steiner trees $T_1, \ldots, T_k$ are included in $\mathcal{T}$. In the j-th iteration, for the q-th request a Steiner tree T_{jk+q} is generated. No matter whether it is identical to any previously generated tree, we just consider it as a new one and include it in the tree set $\mathcal{T}$. Meanwhile, we set the corresponding components $\hat{x}_{jk+q} = 1$. Therefore after the j-th iteration there are totally $(j+1)k$ nonzero indicator variables (nonzero probability to select the corresponding trees in $\mathcal{T}$). Finally, there are $|\mathcal{T}| = (N+1)k$ non-zero indicator variables.

However, in practice it is not easy to estimate the exact value of N in advance as there is only an upper bound $O(m(\ln m + \varepsilon^{-2}\ln \varepsilon^{-1}))$ for N. In our implementation, we set $N = 100$. If it is insufficient we will double it, until the value of N suffices. In fact according to our implementation results the setting $N = 100$ is enough as for all of our test instances there are only $O(k)$ Steiner trees generated (See Section 5).

4 Heuristics

4.1 Choose the Step Length

In Algorithm $\mathcal{MC}$ the step length τ is set as $t\theta\nu/(2m(w^T f(x) + w^T f(\hat{x})))$ as in [13, 14], where t and θ are parameters for computing the price vector, and $\nu = (w^T f(x) - w^T f(\hat{x}))/(w^T f(x) + w^T f(\hat{x}))$ is a parameter for stopping rules. In the last coordination steps of $\mathcal{MC}$, we have that $t = O(\varepsilon)$ and $\nu = O(\varepsilon)$ according to the scaling phase and the stopping rules, respectively. Assuming that $\theta/(w^T f + w^T \hat{f}) = O(1)$, we notice that $\tau = O(\varepsilon^2/m)$ is very small. It means that the contribution of the block solution is very tiny and the iterate moves to the desired neighbourhood of the optimum very slowly, which results in a large number of iterations (though the bound in Theorem 1 still holds). In fact in our implementation we find that even at the beginning of the iterative procedure the value of τ defined in [13, 14] is too small. In [11, 13] it is mentioned that any $\tau \in (0, 1)$ can be employed as the step length. We test several feasible settings of τ such as $\tau = 1 - t\theta\nu/(2m(w^T f + w^T \hat{f}))$, $\tau = 1 - \nu$ and $\tau = \nu$. Experimental results show that $\tau = \nu$ is the best among them. With this heuristic, the number of iterations is reduced significantly (see Section 5).

4.2 Remove the Scaling Phase

In our implementation we set $\varepsilon = 10^{-5}$. In this way we are able to estimate the number of scaling phases $N_s = -\log \varepsilon = 5 \log 10 \approx 16.61$. Therefore in the total computation there should be 17 scaling phases. In fact we find that in many cases in our implementation there is only one iteration in each scaling phase. Thus, the number of scaling phases dominates the overall number of iterations and there are only $O(1)$ iterations in a scaling phase.

We notice that in [13] the algorithm without scaling phase is also mentioned and the corresponding coordination complexity is $O(mc^2(\ln m + \varepsilon^{-2} + \varepsilon^{-3}\ln c))$. In our implementation $c = 2$ is a constant so the complexity does not increase much. In practice with this strategy the algorithm could run faster, especially when there are only very few iterations in each scaling phase. Therefore we use this approach and the number of iterations is reduced.

4.3 Add Only One Steiner Tree in Each Iteration

Algorithm $\mathcal{MC}$ calls the block solver $\mathcal{MSTS}$ k times independently for the k requests in each iteration. We now consider Example 1 which leads to hardness for finding an optimum solution. The instance is as follows: In the graph G, $|V| = 4$ and $|E| = 5$. The edges are $(1, 2)$, $(1, 3)$, $(2, 3)$, $(1, 4)$ and $(2, 4)$ (see Figure 1(a)). There are 3 identical requests $S_q = \{1, 2\}$ for $q = 1, 2, 3$. In general we can also study the graphs with $|V| = p$, $|E| = 2p - 3$, with edge set $E = \{(1, 2), (1, i), (i, 2)|i = 3, \dots, p\}$ and identical requests $S_q = \{1, 2\}$ for $q = 1, \dots, p - 1$ for $p \in N$ and $p \geq 4$.

In the initialization step of Algorithm $\mathcal{MC}$, each edge is assigned an identical length 1/5. For all requests, the minimum Steiner trees T_q, $q = 1, 2, 3$ are all the path containing only edge $(1, 2)$, with a total length 1/5. After T_1 is generated for the first request S_1, Algorithm $\mathcal{MC}$ is not aware of the change of the congestion on edge $(1, 2)$, and still assign the identical trees T_2 and T_3 to requests S_2 and S_3. After the initialization congestions of edges are all zero except for edge $(1, 2)$, which has a congestion 3 (see Figure 1(b)). In the first iteration, the edge lengths changes and the length on edge $(1, 2)$ is the maximum, and other edges have very small lengths. Therefore Algorithm $\mathcal{MC}$ will choose the path $\{(1, 3), (3, 2)\}$ as T_4 for S_1. With the same arguments, other requests are also assigned the path $\{(1, 3), (3, 2)\}$ as their corresponding minimum Steiner trees (see Figure 1(c)). In the second iteration all requests are assigned the path $\{(1, 4), (4, 1)\}$ (see Figure 1(d)) and in the third iteration the solution returns back to the case in Figure 1(b). This procedure continues and in each iteration only one path is used for all requests, which leads to a wrong solution with always a maximum congestion 3. It is also verified by our implementation.

This problem does not result from Algorithm $\mathcal{MC}$ itself but from the data structure (the indices of the vertices and edges). An intuitive approach is to re-index the nodes (and hence edges) after each Steiner tree is generated. However, this approach causes large computational cost of re-indexing. The strategy we apply here is to establish a permutation of the requests. In each iteration only one request is chosen according to the permutation, and a Steiner tree is

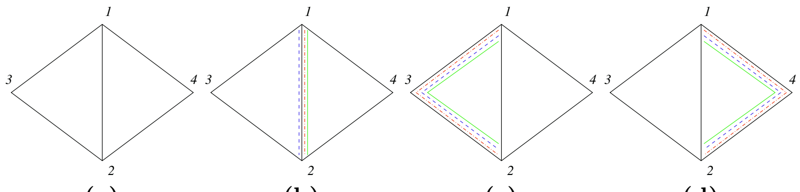

Fig. 1. Examples 1

generated by $\mathcal{MSTS}$ for the chosen request. This method is applied in [23] to solve the packing problems with block structured variables but with a standard (not weak) approximate block solver. The bound on the number of iterations is $O(k \ln m(\ln \min\{k, m\} + \varepsilon^{-2}))$, where k is the number of blocks of variables. There is also a randomized algorithm for such problems [11] with a number of iterations bounded by $O(k \ln m(\ln k + \varepsilon^{-2}))$. In our problem there are also k blocks of indicator variables corresponding to the k requests. But there are only weak block solvers with the approximation ratio $c > 1$. So we apply this method as a heuristic in our implementation. Furthermore, in our implementation we find it is not necessary to construct and maintain the permutation. We can just choose the requests according to their indices. In this way the optimum solution can be attained in only 3 iterations for Example 1 such that the three requests are realized by the three disjoint paths between vertex 1 and 2.

It is interesting that when this heuristic is employed, not only the quality of the solution but also the running time are improved. In fact for many instances with symmetric topology structure, such a problem due to data structure can happen without our heuristic or the re-indexing approach.

4.4 Punish the Edges with Large Congestions

We use a 2-approximate minimum Steiner tree solver ($c = 2$) here as the block solver in our implementation. We notice that with the above heuristics we can only obtain a solution bounded by (1) as indicated in our implementation results (see Section 5). In fact our implementation shows that as soon as the solution fulfils (1) for $c = 2$, the algorithm halts immediately. In order to obtain a better approximate solution still with $\mathcal{MSTS}$, we could modify the stopping rules to force the algorithm to continue running with more iterations. However, here we use another heuristic without changing the stopping rules in order to avoid more running time.

The price vector is used as edge length in our algorithm for the Steiner tree problem. It is obvious that a large congestion leads to a large length on the edge. Thus we can add extra punishment to edges with large congestions to balance the edge congestions over the whole graph. We apply the following strategy:

First we define an edge e_i *high-congested* if its congestion f_i fulfils the following inequality:

$$\lambda - f_i \leq r(\lambda - \hat{\lambda}). \tag{3}$$

Here λ is the maximum congestion in current iteration, $\hat{\lambda}$ is the average congestion defined as the sum of congestions over all edges divided by the number of edges with nonzero congestions, and r is a ratio depending on the quality of the current solution defined as follows:

$$r = \sqrt{1 - (\lambda_0 - \lambda)^2/\lambda_0^2}, \tag{4}$$

where $\lambda_0 \geq \lambda$ is the maximum congestion of the initial solution. According to (4), $r \in (0, 1]$. In addition, (4) is an ellipse function. At the beginning $\lambda \approx \lambda_0$ so $r \approx 1$. With the maximum congestion being reduced, the value of r also decreases. Furthermore, according to the property of the ellipse function, at the beginning of the iterative procedure the value of r decreases slowly. When the congestions are well distributed, r reduces quickly. This formulation guarantees that at the beginning of the iterative procedure there is a large portion of high-congested edges while later there is only a small portion.

Next we re-assign length function to all edges in the graph. For any edge not high-congested, we keep its length as computed by the method in [13, 14]. For a high-congested edge, we set its length as its current congestion. Afterwards we normalize all edge lengths such that the sum of lengths of all edges is exactly one. Our implementation shows that this technique can not only improve the quality of solution but also speed up the convergence (with less iterations).

5 Experimental Results

Our test instances are two-dimensional rectilinear lattices (grid graphs with certain rectangular holes). These instances typically arise in VLSI logic chip design problems and the holes represent big arrays on the chips. These instances are regarded hard for path- or tree-packing problems. The instances have the following sizes:

Example 1. $n = 2079$ and $m = 4059$; $k = 50$ to 2000.

Example 2. $n = 500$ and $m = 940$; $k = 50$ to 300.

Example 3. $n = 4604$ and $m = 9058$; $k = 50$ to 500.

Example 4. $n = 1277$ and $m = 2464$; $k = 50$ to 500.

We first demonstrate the influence of the heuristics mentioned in Section 4 by a hard instance. The instance belongs to Instance 3 with 4604 vertices, 9058 edges and 100 requests. The sizes of requests varies and the smallest request has 5 vertices. We test our algorithm without or with heuristics and the results are shown in Table 1.

We refer Algorithm 1 the original Algorithm $\mathcal{MC}$ without any heuristics. Algorithm 2 is referred to Algorithm $\mathcal{MC}$ with the heuristic to add only one Steiner tree in each iteration. For Algorithm 3, we refer the algorithm similar to Algorithm 2 but with step length $\tau = \nu$. Algorithm 4 is similar to Algorithm 3 but with extra punishment to high-congested edges. It is worth noting that in Algorithm 1, the block solver $\mathcal{MSTS}$ is called k times in each iteration, while

Table 1. Numerical results of Algorithm $\mathcal{MC}$ without and with heuristics

	Alg. 1	Alg. 2	Alg. 3	Alg. 4
Initial Congestion	17	17	17	17
Final Congestion	17	13	6	4
Number of Calls	–	44	85	90

Table 2. Numerical results of Instance 1 compared with Garg-Könemann's algorithm and Baltz-Srivastav's heuristic

# req.(# term.)	G-K	B-S	Alg. 3	Alg. 4
50(4)	2.5(5000)	2(50)	4(111)	2(67)
100(4)	4.4(10000)	3(100)	7(207)	3(180)
150(4)	6.1(15000)	4(300)	9(314)	5(131)
200(4)	8.0(20000)	5(400)	11(594)	6(260)
300(4)	11.5(30000)	7(900)	15(826)	8(492)
500(4)	19.9(50000)	12(1000)	23(1389)	13(977)
1000(4)	36.5(100000)	21(69000)	48(2786)	24(2955)
2000(4)	76.1(200000)	44(4000)	96(5563)	54(3878)
500(≥ 2)	69.1(50000)	32(4500)	39(1381)	37(501)
1000(≥ 2)	100.5(100000)	65(3000)	78(2933)	72(1004)

in Algorithm 2, 3 and 4 $\mathcal{MSTS}$ is called only once in each iteration. In order to compare the running time fairly, we count the number of calls to $\mathcal{MSTS}$ as the measurement of running time. In fact according to our implementation, the running time of $\mathcal{MSTS}$ dominates the overall running time. From Table 1 it is obvious that the heuristics improve the quality of solution much. Since the value of τ is too small in Algorithm 1, the iterate does not move after long time and we manually terminate the program.

In [4] Instance 1 was implemented to test their heuristic based on an online algorithm in [2] and a well-known approximation algorithm for packing problems in [10] based on an approximation algorithm for the fractional multicommodity flow problem. Here, we also use the same instances to test our Algorithm 3 and 4. The results are shown in Table 2. In the first column of Table 2 the number of requests and the number of terminals per request are given. The solution delivered by the algorithms and heuristics are presented in other columns, together with the number of calls to $\mathcal{MSTS}$ in brackets. The results of Garg and Könemann's algorithm are only for the LP relaxation.

It is clear that Algorithm 4 is superior to Algorithm 3 in the examples of regular requests (with 4 terminals per request). Furthermore, it is worth noting that our Algorithm 4 delivers better solutions than the algorithm by Garg and Könemann [10] with much less number of calls to $\mathcal{MSTS}$. In fact the fractional solutions of Algorithm 3 are also better than those of the algorithm by Garg and Könemann. Our results are not as good as those of the heuristic proposed in [4] for these instances. However, there is no performance guarantee of their heuristic, while our solutions are always bounded by (1). A possible reason of this case is that we use a 2-approximate block solver, which leads to a low accuracy. We believe that a better approximate minimum Steiner tree solver and some more strict stopping rules can result in better performance of our algorithm.

We also test our Algorithm 4 by Instances 2, 3 and 4, which are not implemented in [4]. The results are listed in Table 3. Our algorithm can always generate satisfactory solutions for these hard instances in short running times.

Table 3. Numerical results of Instance 2, 3 and 4

Inst.	# req.(# term.)	Alg. 4	Inst.	# req.(# term.)	Alg. 4
2	$50(\geq 10)$	7(37)	2	$150(\geq 30)$	25(116)
2	$50(\geq 5)$	6(41)	2	$200(\geq 10)$	34(404)
2	$100(\geq 5)$	12(83)	2	$200(\geq 30)$	32(148)
2	$100(\geq 10)$	14(77)	2	$300(\geq 10)$	38(559)
2	$150(\geq 10)$	19(131)	2	$300(\geq 30)$	47(231)
3	$50(\geq 5)$	2(146)	3	$200(\geq 20)$	13(320)
3	$50(\geq 20)$	4(105)	3	$300(\geq 5)$	8(565)
3	$100(\geq 5)$	4(90)	3	$300(\geq 20)$	19(285)
3	$100(\geq 20)$	7(186)	3	$500(\geq 5)$	13(962)
3	$200(\geq 5)$	6(210)	3	$500(\geq 20)$	30(483)
4	$50(\geq 5)$	3(103)	4	$200(\geq 20)$	24(165)
4	$50(\geq 20)$	7(36)	4	$300(\geq 5)$	14(560)
4	$100(\geq 5)$	6(83)	4	$300(\geq 20)$	34(280)
4	$100(\geq 20)$	13(86)	4	$500(\geq 5)$	24(475)
4	$200(\geq 5)$	10(173)	4	$500(\geq 20)$	56(390)

For any request of all these instances, the corresponding $\mathcal{MSTS}$ is called at most 3 times.

6 Conclusion

We have implemented the approximation algorithm for the multicast congestion problem in communication networks in [14] based on [13] with some heuristics to improve the quality of solution and reduce the running time. The numerical results for hard instances are reported and are compared with the results of the approximation algorithm in [10] and a heuristic in [4]. It shows that the algorithm in [13] is practical and efficient for packing problems with a provably good approximation ratio.

There could be some interesting techniques to further improve the experimental performance of the algorithm. A possible method is to use a better approximate minimum Steiner tree solver (e.g. the algorithm in [21]), though the running time will be significantly increased. Another technique is to use the line search for the step length to reduce the number of iterations. However, the running time in each iteration increases so the improvement of the overall running time could be not significant. More heuristics and techniques are to be designed and implemented in our further work.

Acknowledgment

The authors thank Andreas Baltz for providing the data of testing instances. We also thank Klaus Jansen for his helpful discussion.

References

1. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and hardness of approximation problems, *Journal of the ACM*, 45 (1998), 501-555.
2. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin and O. Waarts, On-line routing of virtual circuits with applications to load balancing and machine scheduling, *Journal of the Association for Computing Machinery*, 44(3) (1997), 486-504.
3. A. Baltz and A. Srivastav, Fast approximation of multicast congestion, *manuscript* (2001).
4. A. Baltz and A. Srivastav, Fast approximation of minimum multicast congestion - implementation versus theory, *Proceedings of the 5th Conference on Algorithms and Complexity*, CIAC 2003.
5. M. Bern and P. Plassmann, The Steiner problem with edge lengths 1 and 2, *Information Professing Letters*, 32 (1989), 171-176.
6. R. Carr and S. Vempala, Randomized meta-rounding, *Proceedings of the 32nd ACM Symposium on the Theory of Computing*, STOC 2000, 58-62.
7. S. Chen, O. Günlük and B. Yener, The multicast packing problem, *IEEE/ACM Transactions on Networking*, 8 (3) (2000), 311-318.
8. M. Chlebík and J. Chlebíková, Approximation hardness of the Steiner tree problem, *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, SWAT 2002, LNCS 2368, 170-179.
9. R. Floren, A note on "A faster approximation algorithm for the Steiner problem in graphs", *Information Processing Letters*, 38 (1991), 177-178.
10. N. Garg and J. Könemann, Fast and simpler algorithms for multicommodity flow and other fractional packing problems, *Proceedings of the 39th IEEE Annual Symposium on Foundations of Computer Science*, FOCS 1998, 300-309.
11. M. D. Grigoriadis and L. G. Khachiyan, Fast approximation schemes for convex programs with many blocks and coupling constraints, *SIAM Journal on Optimization*, 4 (1994), 86-107.
12. M. D. Grigoriadis and L. G. Khachiyan, Coordination complexity of parallel price-directive decomposition, *Mathematics of Operations Research*, 2 (1996), 321-340.
13. K. Jansen and H. Zhang, Approximation algorithms for general packing problems with modified logarithmic potential function, *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science*, TCS 2002, 255-266.
14. K. Jansen and H. Zhang, An approximation algorithm for the multicast congestion problem via minimum Steiner trees, *Proceedings of the 3rd International Workshop on Approximation and Randomized Algorithms in Communication Networks*, ARACNE 2002, 77-90.
15. R. M. Karp, Reducibility among combinatorial problems, *in R. E. Miller and J. W. Thatcher (Eds.), Complexity of Computer Computations*, Plenum Press, NY, (1972), 85-103.
16. P. Klein, S. Plotkin, C. Stein and E. Tardos, Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts, *SIAM Journal on Computing*, 23 (1994), 466-487.
17. K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, *Information Processing Letters*, 27 (1988), 125-128.
18. C. A. S. Oliveira and P. M. Pardolos, A survey of combinatorial optimization problems in multicast routing, *Computers and Operations Research*, 32 (2005), 1953-1981.

19. P. Raghavan, Probabilistic construction of deterministic algorithms: Approximating packing integer programs, *Journal of Computer and System Science*, 37 (1988), 130-143.
20. P. Raghavan and C. Thompson, Randomized rounding: a technique for provably good algorithms and algorithmic proofs, *Combinatorica*, 7 (1987), 365-374.
21. G. Robins and A. Zelikovsky, Improved Steiner tree approximation in graphs, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2000, 770-779.
22. S. Vempala and B. Vöcking, Approximating multicast congestion, *Proceedings of the tenth International Symposium on Algorithms and Computation*, ISAAC 1999, LNCS 1741, 367-372.
23. J. Villavicencio and M. D. Grigoriadis, Approximate structured optimization by cyclic block-coordinate descent, *Applied Mathematics and Parallel Computing, H. Fisher et al. (Eds), Physica Verlag*, (1996), 359-371.
24. J. Villavicencio and M. D. Grigoriadis, Approximate Lagrangian decomposition with a modified Karmarkar logarithmic potential, *Network Optimization, P. Pardalos, D. W. Hearn and W. W. Hager (Eds.), Lecture Notes in Economics and Mathematical Systems 450, Springer-Verlag, Berlin*, (1997), 471-485.