A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning

Kārlis Freivalds

University of Latvia
Institute of Mathematics and Computer Science
Raina blvd. 29, LV-1459, Riga, Latvia
Karlis.Freivalds@mii.lu.lv

Abstract. We propose a new method for finding the minimum ratio-cut of a graph. Ratio-cut is NP-hard problem for which the best previously known algorithm gives an $O(\log n)$-factor approximation by solving its dually related maximum concurrent flow problem. We formulate the minimum ratio-cut as a certain nondifferentiable optimization problem, and show that the global minimum of the optimization problem is equal to the minimum ratio-cut. Moreover, we provide strong symbolic computation based evidence that any strict local minimum gives an approximation by a factor of 2. We also give an efficient heuristic algorithm for finding a local minimum of the proposed optimization problem based on standard nondifferentiable optimization methods and evaluate its performance on several families of graphs. We achieve $O(n^{1.6})$ experimentally obtained running time on these graphs.

1 Introduction

Balanced cuts of a graph are hard computation problems important both in theory and practice. Ratio-cut is the most fundamental one since most of the others including minimum quotient cut, minimum bisection, multi-way cuts can be easily approximated using it [13, 20]. Also several other important approximation algorithms like crossing number and minimum cut linear arrangement are based on the ratio-cut [13]. Ratio-cut has many practical applications, most important being VLSI design, clustering and partitioning [23, 14, 1].

Since ratio-cut is a NP-hard problem [15] we must seek for approximation algorithms to solve it in practically reasonable time. Many purely heuristic algorithms were developed [23, 25, 21, 8] most of them relying on simulated annealing, spectral methods or iterative movement of nodes from one side of the partition to the other. A common idea exploited by several authors [25, 2, 9, 21, 22] to improve their quality is using multi-scale graph representation usually obtained by edge contraction. At first a partition at the coarsest scale is obtained and then refined to a more detailed one by one of mentioned algorithms. Although these algorithms may perform well in practice no optimality bounds are known for them.

The best previously known algorithm with proven optimality bounds finds an O(log n)-factor approximation, where n is the number of nodes in the graph. It is based on reduction of the ratio-cut problem to the multi-commodity flow problem, which can be solved with polynomial time linear programming methods. Unfortunately this method is not practical since the resulting linear program is of quadratic size of the number of nodes in the graph and cannot be solved efficiently. Then, approximation algorithms [16, 17, 12, 24] were discovered for the multi-commodity flow problem itself making this approach usable in practice. Several heuristic implementations [16, 11, 24] are based on this idea, some of them quite effective and practical. The most elaborate one [11] can deal with up to 100000-node graphs in reasonable time.

In this paper we propose a new way of finding the minimum ratio-cut of a graph. We construct a nondifferentiable optimization problem whose minimum solution equals the minimum ratio-cut value and use nonlinear programming methods to search for it. Since the problem is non-convex, we may find only a local minimum. However, we show that any strict local minimum gives us a factor of 2 approximate cut. For that purpose we introduce a notion of locally minimal ratio cut for which no subset of nodes taken from one side of the cut and moved to the other side decrease the cut value. We establish one-to-one correspondence between strictly locally minimal cuts and strict local minima of the proposed optimization problem.

The reduction of a NP-hard discrete problem to a continuous one is not a novel idea. For example the maximum clique problem of a graph can also be stated as an optimization problem [6] and numerical optimization methods for finding the optimum may be used. However for the maximum clique problem no optimality bounds of a local minimum are known. To show that our method is practical we present an efficient heuristic algorithm for finding a local minimum of the proposed problem, which is based on the standard methods of nondifferentiable optimization and analyze its performance on several families of graphs. With the proposed method we can find a good partition of a 200000-node graphs in less than one hour.

2 Problem Formulation

We are dealing with an undirected graph $G = (V, E)$, where V is its node set and E is its edge set. The nodes of the graph are identified by natural numbers from 1 to n. Each node i has a weight $d_i \geq 0$, and each edge (i, j) has a capacity $c_{ij} \geq 0$, satisfying the properties $c_{ij} = c_{ji}$, $c_{ii} = 0$. We define $c_{ij} = 0$ when there is no edge (i, j) in G. We assume that there are at least two nodes with non-zero weights. (A, A') denotes a cut that separates a set of nodes A from its complement $A' = V \backslash A$. The capacity of the cut $C(A, A')$ is the sum of edge capacities between A and A'. The ratio of the cut $R(A, A')$ is defined as follows

$$R(A, A') = \frac{C(A, A')}{\sum\limits_{i \in A} d_i \cdot \sum\limits_{i \in A'} d_i} \tag{1}$$

We will focus on finding the minimum ratio-cut i.e. the cut (A, A') with the minimum ratio over all nonempty A, A'.

Definition 1. *A cut (A, A') is locally minimal if for all non-empty $U \subset A, U \neq A$: $R(A, A') \leq R(A \backslash U, A' \cup U)$ and for all non-empty $U' \subset A', U' \neq A'$: $R(A, A') \leq R(A \cup U', A' \backslash U')$. Similarly we call a ratio-cut strictly locally minimal if strong inequalities hold in this definition.*

3 Ratio-Cut as an Optimization Problem

We can assign a variable $x_i \in \mathbb{R}$ to each node i and consider the following optimization problem over x from $\mathbb{R}^n$.

$$\min F(x) = \sum_{i,j \in V} c_{ij} |x_i - x_j| \tag{2}$$

$$\text{subject to } H(x) = \sum_{i,j \in V} d_i d_j |x_i - x_j| = 1, \tag{3}$$

$$\sum_{i \in V} x_i = 0. \tag{4}$$

This optimization problem is equivalent to the ratio-cut problem in the sense described below.

Definition 2. *A characteristic vector x^A for a cut (A, A') is defined such that its components*

$$x_i^A = \begin{cases} a, \; i \in A \\ b, \; i \in A' \end{cases} , \textit{where} \tag{5}$$

$$a = -\frac{1}{2 \sum_{i \in A} d_i \cdot \sum_{i \in A'} d_i} \cdot \frac{|A'|}{|V|}, b = \frac{1}{2 \sum_{i \in A} d_i \cdot \sum_{i \in A'} d_i} \cdot \frac{|A|}{|V|}.$$

It is straightforward from this definition that for a cut (A, A') x^A satisfies the constraints (3, 4), and $F(x^A) = R(A, A')$.

Definition 3. *For some feasible x and some $p \in \mathbb{R}$ we call the cut (P, P') positional, if $P = \{i | x_i \leq p\}$, $P' = V \backslash P$, and both P and P' are non-empty. The ratio of this cut $R_p(x) = R(P, P')$. For a fixed x we can speak of minimum positional cut $R_{\min}(x)$ over all possible positional cuts obtained for different p:*

$$R_{\min}(x) = \min_{p \in \mathbb{R}} R_p(x).$$

Theorem 1. *For each feasible x^* of (2, 3, 4) $F(x^*) \geq R_{\min}(x^*)$. Also $F(x^*) > R_{\min}(x^*)$ if there are at least two positional cuts with different values.*

Proof. Here we only sketch the main steps of the proof, for details omitted due to space limitation refer to [7]. Let us partition all nodes into three sets U_1, U_2, U_3 as follows.

$$y_1^* = \min_i x_i^*, \quad y_2^* = \min_{i|x_i^*>y_1} x_i^*,$$

$$U_1 = \{i|x_i^* = y_1^*\}, \quad U_2 = \{i|x_i^* = y_2^*\}, \quad U_3 = \{i|x_i^* > y_2^*\}$$

If U_3 is empty there is exactly one positional cut, x^* is in the form of characteristic vector and $F(x^*) = R_{\min}(x^*)$ what concludes this proof, else define

$$y_3^* = \min_{i|x_i^*>y_2} x_i^*.$$

We create a sub-problem of (2, 3, 4) by reducing the original one to a new variable $y = (y_1, y_2, y_3)$. Next, we further restrict it with $y_1 \le y_2 \le y_3$ and can drop the absolute value signs getting:

$$\min F_2(y) = 2 \sum_{i\in U_1, j\in U_2} c_{ij}(y_2 - y_1) + \sum_{i\in U_1, j\in U_3} c_{ij}(y_3 + l_j - y_1) + \sum_{i\in U_2, j\in U_3} c_{ij}(y_3 + l_j - y_2) + K, \quad (6)$$

$$\text{subject to } H_2(y) = 2 \sum_{i\in U_1, j\in U_2} d_i d_j(y_2 - y_1) + \sum_{i\in U_1, j\in U_3} d_i d_j(y_3 + l_j - y_1) + \sum_{i\in U_2, j\in U_3} d_i d_j(y_3 + l_j - y_2) + P, \quad (7)$$

$$|U_1|y_1 + |U_2|y_2 + |U_2|y_2 = 0, \quad (8)$$

$$y_1 \le y_2 \le y_3, \quad (9)$$

where P and K are appropriately calculated constants.

We have obtained a locally equivalent linear program in the sense that for y^* the constraints (7, 8, 9) are satisfied and $F(x^*) = F_2(y^*)$. Also, if we can find a better solution for $(6 - 9)$ we can substitute the result back to the original problem giving a better feasible solution for it. From the linear programming theory it is known that we can find the optimal solution by examining the vertices of the polytope defined by the constraints – in our case that means when one of the inequalities in (9) is satisfied as equality. Let us examine both cases.

In case $y_1 = y_2$ after some calculations we get

$$F_2(y_1, y_1, y_3) = (1 - L)R(U_1 \cup U_2, U_3) + B \quad (10)$$

for appropriate constants L and B. And similarly in case $y_2 = y_3$ we get

$$F_2(y_1, y_2, y_2) = (1 - L)R(U_1, U_2 \cup U_3) + B. \quad (11)$$

We chose the solution y with $y_1 = y_2$ if $R(U_1 \cup U_2, U_3) < R(U_1, U_2 \cup U_3)$, otherwise choose the solution with $y_2 = y_3$. It is evident that if both these ratio costs are non-equal we get a strictly smaller function value. We substitute the solution back into the original problem obtaining a new x. x is a feasible solution of (2, 3, 4) with a smaller or equal function value and the set U_2 merged to U_1 or U_3. We repeat the described process until U_3 is empty. Let us analyze the resulting x and sets U_1 and U_2. We have $F(x) = R(U_1, U_2)$, where (U_1, U_2) is some positional cut of x^* (in fact the minimal one), hence $F(x^*) = F(x) = R_{\min}(x^*)$. If we had some non-equal cuts compared during the process, we have a strict decrease in the function and hence the second statement of the theorem holds. □

Definition 4. *A feasible x^* is a local minimum of (2, 3, 4) if there exists $\varepsilon > 0$ such that $F(x^*) \leq F(x)$ for each x satisfying (3, 4) and $||x - x^*|| < \varepsilon$.*

Definition 5. *A feasible x^* is a strict local minimum of (2, 3, 4) if there exists $\varepsilon > 0$ such that $F(x^*) < F(x)$ for each $x \neq x^*$ satisfying (3, 4) and $||x - x^*|| < \varepsilon$.*

Theorem 2. *Each strict local minimum x^* of (2, 3, 4) is in the form $x^* = x^A$ for some cut (A, A').*

Proof. We need to prove that in a strict local minimum the expression (5) holds for some a and b, the correct values for a and b are guaranteed by the constraints (3) and (4). Assume to the contrary that there are more than two distinct values for the components of x^*. We form the reduced problem like in Theorem 1 obtaining equations $(6-9)$. We are able to do that since U_3 is non-empty due to our assumption. From the linear programming theory a strict local minimum can only be on the vertex of the polytope defined by the constraints $(7 - 9)$, however in our case y^* cannot be on the vertex since the constraints where equality holds at y^* are less then the number of variables. Consequently, x^* cannot be a strict local minimum for the original problem. Hence our assumption is false and the theorem is proven. □

There is one-to-one correspondence between strictly locally minimal cuts and strict local minimums of (2, 3, 4) as stated in the following theorem.

Theorem 3. *x is a strict local minimum of (2, 3, 4) if and only if x is a characteristic vector of some strictly locally minimal cut (A, A').*

The proof is rather technical and is omitted due to space constraints. See [7] for details.

We shall note that also for each non-strictly minimal ratio cut its characteristic vector x^A gives a local minimum of the function, however the converse is not true. There exist non-strict local minima of (2, 3, 4) with the function value not equal to any locally minimal cut value.

Theorem 4. *The minimum ratio-cut R is equal to the optimum solution of (2, 3, 4).*

Proof. From Theorem 1 $F(x) \geq R_{\min}(x) \geq R$. For the characteristic vector x^A of the minimum ratio cut $F(x^A) = R$. The claim follows immediately. □

Theorem 5. *Each locally minimal cut (A, A') is not grater than two times the minimum ratio cut.*

Proof. Let us denote the optimal cut (B, B'). We form four sets $A \cap B$, $A \cap B'$, $A' \cap B$, $A' \cap B'$ shown in Fig. 1. From Definition 1 the ratio of each of the cuts $C_1 = (A \cap B, V - A \cap B)$, $C_2 = (A \cap B', V - A \cap B')$, $C_3 = (A' \cap B', V - A' \cap B')$, $C_4 = (A' \cap B, V - A' \cap B)$ is at least as large as ratio of $C_{12} = (A, A')$ and $C_{23} = (B, B')$. We form a full graph by taking each of the sets $A \cap B$, $A \cap B'$, $A' \cap B$, $A' \cap B'$ as the nodes of the graph and assign edge capacities as the sum of all edge capacities in the original graph between the corresponding sets. We obtain

$$R_1 = \frac{c_1 + c_4 + c_6}{d_1(d_2 + d_3 + d_4)}, R_2 = \frac{c_1 + c_2 + c_5}{d_2(d_1 + d_3 + d_4)},$$
$$R_3 = \frac{c_2 + c_3 + c_6}{d_3(d_1 + d_2 + d_4)}, R_4 = \frac{c_3 + c_4 + c_5}{d_4(d_1 + d_2 + d_3)},$$
$$R_{12} = \frac{c_2 + c_4 + c_5 + c_6}{(d_1 + d_2)(d_3 + d_4)}, R_{23} = \frac{c_1 + c_3 + c_5 + c_6}{(d_2 + d_3)(d_1 + d_4)}.$$
$$R_1 \geq R_{12}, R_2 \geq R_{12}, R_3 \geq R_{12}, R_4 \geq R_{12}.$$

And the statement of the theorem to be proven translates to $\frac{R_{12}}{R_{23}} \leq 2$. We verified this using symbolical computation in Mathematica 4.0. See [7] for details. □

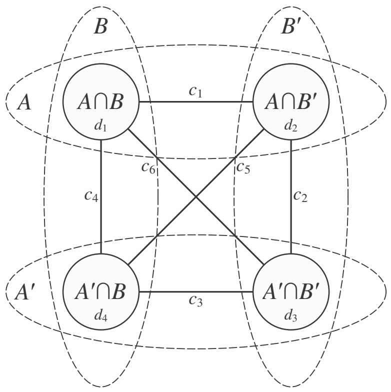

Fig. 1. Illustration to the proof of Theorem 5

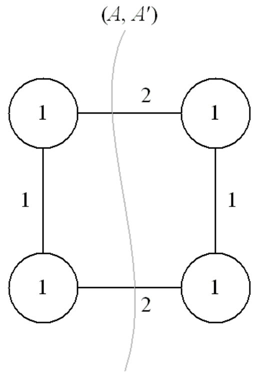

Fig. 2. A graph achieving the bound of Theorem 5

Corollary 1. *Each strict local minimum of (2, 3, 4) is not grater than two times the minimum ratio cut.*

The proof is straightforward from Theorem 3 and Theorem 5.

The bound of Theorem 5 is tight; the graph with 4 nodes shown in Fig. 2 achieves this bound. One can make larger examples easily by substituting any connected graph with sufficiently high edge capacities in place of the nodes of the given graph.

4 The Algorithm

In this section we present a heuristic algorithm based on standard nondifferentiable optimization methods for finding a local minimum of (2, 3). Finding a minimum of a nondifferentiable function is one of well- explored nonlinear programming topics [5, 18]. One of the possibilities is to approximate the nondifferentiable function with a smooth one and apply one of the well-known algorithms to find its local minimum [5, 3]. Often a better approach is to handle it directly. Indeed, in our case we obtain a very simple and fast algorithm.

Most of the optimization theory deals with convex problems for which algorithms with proven convergence can be developed. Many of these methods also work for non-convex functions finding a local minimum. However, then the convergence cannot be shown or can be shown only in a local neighborhood of some local minimum what is not satisfactory in our case. The very basic algorithm of nondifferentiable optimization is a *subgradient* algorithm [5, 18]. We will adopt it for solving our problem. Since we apply it to a non-convex problem, it should be considered mostly as a heuristic, however practice shows that it actually converges to a local minimum of our problem.

The algorithm is iterative one. The iteration of the algorithm consists of going in the negative direction of a subgradient of the function by a fixed step and then performing a projection onto the constraint (3). The constraint (4) was introduced for technical reasons required in proofs and may be not considered in the algorithm. An appropriate subgradient q of F can be calculated with the following equation for its components

$$q_i = \sum_{j \in V} \mathrm{sign}(x_i - x_j),$$

Choosing the right step size is crucial for the convergence speed. Our heuristic observation is that it should be proportional to the node spread. We choose the step equal to `stepFactor`$\cdot (x_{\max} - x_{\min})$, where `stepFactor` is some parameter. Initially `stepFactor` $= 1/14$. Its update during the algorithm is be discussed later. The projection is performed by going in the direction of a subgradient of the constraint till the constraint is reached. For the constraint H we can write its subgradient r in a different form to allow its faster evaluation

$$r_i = d_i \sum_{j \in V} d_j \mathrm{sign}(x_i - x_j) = d_i \left(\sum_{j : x_j < x_i} d_j - \sum_{j : x_j > x_i} d_j \right).$$

If we sort the x_i values and consider them in increasing order, then the needed sums can be updated incrementally leading to linear time evaluation (not counting the sorting). To perform the projection we need to calculate the step length towards the constraint. To simplify the calculations we will assume that the ordering of x_i will not change during the projection. Then the constraint function H becomes linear and the desired step can be easily calculated from the linear equation defined as the point of value 1 on the ray defined by the subgradient from the current point. In the case of unit node weights our assumption is fulfilled. To see this we must observe that the step in the function will always lead to a point with $H < 1$. Then, if we have unit node weights, $r_i \leq r_k$ for all i and k satisfying $x_i \leq x_k$, so after the projection the distance between them can only increase thus keeping the same ordering. For the case of arbitrary node weights such algorithm gives a usable approximation of the projection step length.

The whole algorithm starts with a random initialization. We assign a random position for each node such that $H < 1$ and then perform a projection to obtain a feasible starting position. We experimented also with several other initializations, but obtained significant improvements only for tree graphs. Since the optimal cut for trees can be found in linear time, we can construct a starting position that reveals it and the algorithm will only perform a few iterations to confirm that the solution does not improve. Hence to get a comprehensive picture of the algorithm behavior we decided to consider random initialization only.

After the initialization we perform iterations until convergence. To tell when the process is converged we track the minimum positional cut values obtained at each iteration. The same values will also tell us how to change the step size parameter `stepFactor`. If the new cut value is lower than the previous then we are making progress and we should continue with the same step size. If the value is higher than the previous then the step size is too large. If the cut does not change then we have a clue that the process has converged. Of course we do not hurry to make decisions only from one iteration. Instead we wait for a certain number of iterations controlled by the constants `MAX_OSCILLATIONS` and `MAX_EQUAL` before making the decision. Such delay also improves the convergence speed by allowing to iterate longer with a larger step size

To determine the positional cut value at each iteration proceed as follows. We consider the sorted sequence of nodes, calculate the positional cut in each interval between two consecutive nodes and take the minimum one. We can do it incrementally on the sorted sequence in time $\mathrm{O}(n + m)$ provided the sorting, where m is the number of edges in the graph.

As suggested in one of the exercises in [5] the performance of this algorithm can be improved by taking the previous directions into account. We add the previous direction to the current reduced be some constant `REDUCTION_FACTOR` between 0 and 1. It models a heavy ball motion in the presence of a force in the direction of the subgradient. In our experiments such modification with `REDUCTION_FACTOR` $= 0.95$ performed substantially better.

All the steps described before can be implemented to run in time $\mathrm{O}(n + m)$. Adding the time needed for sorting the nodes one iteration takes time $\mathrm{O}(m$

+ n log n). The number of iterations is hard to estimate so we will provide experimental data in the next sections. The constants `MAX_OSCILLATIONS` and `MAX_EQUAL` have the most impact on the iteration count and also on the quality of the obtained cut. So we must select them carefully. After some experimentation we chose `MAX_EQUAL` = 200 and `MAX_OSCILLATIONS` = 30.

```
algorithm ratio-cut
  calculate a random feasible initial position
  acum = 0,oscillationCounter = 0,equalityCounter = 0,stepFactor = 1/14
  while (equalityCounter < MAX EQUAL)
    x = x + acum
    d = direction(x)
    x = x + d
    acum = acum * REDUCTION FACTOR + d
    if(minimum positional cut value has increased in this iteration)
      equalityCounter = 0
      oscillationCounter ++
    else if (minimum positional cut value has decreased in this iteration)
      equalityCounter = 0
    else equalityCounter ++
    if (oscillationCounter > MAX OSCILLATIONS)
      stepFactor /= 1.3
      oscillationCounter = 0
  endwhile
end
function direction (x)
  d = subgradient(x)
  step = (xmax - xmin) * stepFactor
  x1 = x + d*step
  x2 = projection of x1 on the constraint
  return x2-x
end
```

5 The Data

We evaluated the proposed algorithm on three families of graphs: random cubic graphs, random geometric graphs and random trees. We considered only graphs with unit weight nodes and edges.

Random cubic graphs are potentially hard for ratio-cut algorithms since in [13] it was shown that there is actually a O(log n) gap between the minimum ratio-cut and the maximum concurrent flow on these graphs. We generate them using the algorithm provided in [19].

Random geometric graphs are standard test suite for balanced cut problems used in several papers [11, 25]. To generate a geometric graph we place the nodes of the graph randomly in the unit square. Then we include an edge between each two nodes that are within distance δ in the graph, where δ is the minimum value such that the resulting graph is connected.

Tree graphs are seemingly easy graphs because their optimal ratio-cut can be calculated in linear time. Also it is not hard to show that only one local minimum exists for the corresponding optimization problem. Nevertheless, it is an interesting family since our experiments indicate a slow convergence of our algorithm on these graphs. Also we can compare our result with the optimal one. We generate random trees with the classical algorithm where each tree is produced with the same probability. This algorithm produces long and skinny trees, which are particularly difficult for our algorithm.

6 Experimental Results

We implemented the algorithm in C++ and evaluated its performance on a computer equipped with a Pentium III 800 MHz processor and 256 Mbytes of RAM. For each graph family we measured the running time in seconds, the number of iterations and the quality of the produced cuts. Since we did not know the exact cut values for random and geometric graphs, we evaluated how much the ratio-cut value decreases when we continue the algorithm for the same number of iterations as performed before termination. Measuring the decrease of the ratio-cut value we can estimate how far the result is from the optimal.

The algorithm was run on series of graphs of exponentially increasing size from 100 up to 204800 nodes. Ten graphs were generated for each size and the results were averaged. The average node degree for all graph families is constant. Although we cannot specify the degree explicitly for geometric graphs, due to their nature it was about 10 on all instances. The experimental results are given in figures 3 – 8. For each graph family tables show the running time, and how much the the the ratio-cut value improves when the iteration count is doubled. Let us discuss the results separately for each graph family.

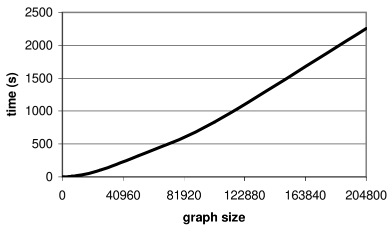

Fig. 3. Running time for cubic graphs

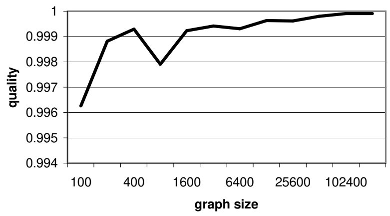

Fig. 4. Quality for cubic graphs

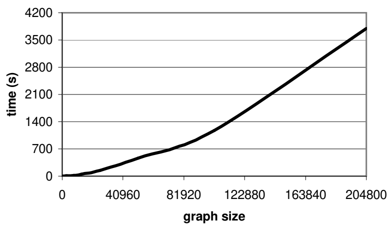

Fig. 5. Running time for geometric graphs

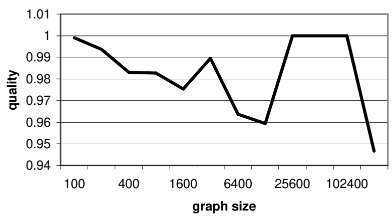

Fig. 6. Quality for geometric graphs

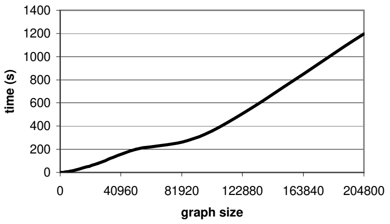

Fig. 7. Running time for tree graphs

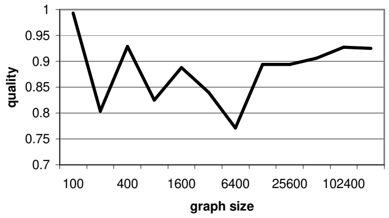

Fig. 8. Quality for tree graphs

6.1 Cubic Graphs

Although these graphs were suggested as hard, the algorithm performed very well on them. It took on average about 35 minutes to partition the 204800 node graphs. The running time dependence from the graph size is shown in Fig. 3. When we approximated the running time with a function in a form $O(n^p)$ we get the asymptotical running time about $O(n^{1.6})$ on these graphs.

The algorithm seems to find a very close to optimal cut since after doubling the iteration count the quality increased only by less than 0.4%. Even more, the quality improved for the larger graphs approaching 1 (see Fig. 4). Such behavior is not surprising since it is not hard to prove that the ratio of a cut (A, A') of a random cubic graph is $\frac{3}{n-1}$ on average independently of the sizes of A and A' (a similar proof for general random graphs is presented in [23]). Then it is unlikely that the minimum ratio cut will be much different from this average value.

6.2 Geometric Graphs

The algorithm performed very well on these graphs both in terms of speed and quality. It took on average about 1 hour to partition the 204800 node graphs. The running time dependence from the graph size is shown in Fig. 5. The asymptotical running time behavior on these graphs was about $O(n^{1.64})$. After doubling the iteration count the quality increased by less than 5% (see Fig. 6). Also visually the cuts seemed the best possible. Fig. 9 shows a typical cut of a 10000-node graph.

6.3 Tree Graphs

The running time for trees was better than for other families. The largest graphs were partitioned in about 20 minutes. The running time dependence from the graph size is shown in Fig. 7. The asymptotical running time on these graphs was about $O(n^{1.45})$. However the quality was poor. As shown in Fig. 10 the obtained cuts were far from the optimal and the quality decreased with increasing graph size. Also doubling the iteration count showed 10% to 20% quality improvement (see Fig. 8).

When we explored further the reason of the poor behavior we found out that convergence is much slower than for other graph families and the stopping criterion does not work correctly in this case. When we allowed the algorithm to run for a sufficiently long time, it always found the optimum solution. However we did not find a robust stopping criterion that correctly works with tree graphs and does not increase running time much for other graph families. As already mentioned a smarter initialization can be used to improve the quality of the partition if such tree or tree-like graphs are common for some application.

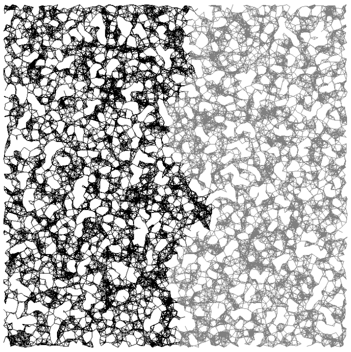

Fig. 9. The obtained cut for a 1000-node geometric graph

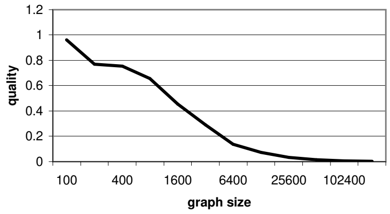

Fig. 10. Optimality for tree graphs

7 Conclusions and Open Problems

We have proposed a nondifferentiable optimization based method for solving the ratio-cut problem and presented a heuristic algorithm implementing it. We have shown that any strict local minimum is 2-optimal. The presented algorithm, however, in certain cases can find a non-strict minimum, but we can easily transform the obtained x vector into the characteristic form. Then the algorithm can be run again from this starting position and this process can be iterated until the result does not change giving a locally minimal cut, which by Theorem 5 is 2-optimal.

The obtained algorithm is simple and fast and uses amount of memory that is proportional to the size of the graph. Its running time and quality are verified experimentally. Its practical running time is about $O(n^{1.6})$ on our test data. The algorithm produce high quality cuts on random cubic and geometric graphs. On trees and other very sparse graphs the quality can be significantly improved by choosing a better starting position than a random one. We evaluated the algorithm on artificially generated data. As a further work it would be important to evaluate its performance on real-life problems. Although the algorithm performed well on most graphs, anyway it is heuristic. It is an open question whether we can find a local minimum of (2, 3, 4) in polynomial time?

8 Acknowledgements

The author would like to thank Paulis Kikusts and Krists Boitmanis for valuable discussion and help in preparation of the paper.

References

1. C. J. Alpert and A. Kahng, Recent directions in netlist partitioning: a survey, tech. report, Computer Science Department, University of California at Los Angeles, 1995.
2. C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. Proc. Design Automation Conf, pp. 530 − 533, 1997.
3. R. Baldick, A. B. Kahng, A. Kennings and I. L. Markov, Function Smoothing with Applications to VLSI Layout, Proc. Asia and South Pacific Design Automation Conf., Jan. 1999.
4. J. W. Berry and M. K. Goldberg, Path Optimization for Graph Partitioning Problems. Technical report TR: 95-34, DIMACS, 1995.
5. D. Bertsekas, Nonlinear Programming, Athena Scientific, 1999.
6. I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. Handbook of Combinatorial Optimization, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.
7. K. Freivalds. A Nondifferentiable Optimization Approach to Ratio-Cut Partitioning. Manuscript 2003. Available at *http://www.gradetools.com/karlisf/papers.html*.
8. L. Hagen and A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1074 − 1085.
9. T. Hamada, C. K. Cheng, P. M. Chau. An efficient multilevel placement technique using hierarchical partitioning. IEEE Trans. on Circuits and Systems, vol. 39(6) pp. 432 − 439, June 1992.
10. P. Klein, S. Plotkin, C. Stein, E. Tardos, Faster approximation algorithms for unit capacity concurrent flow problems with applications to routing and sparsest cuts, SIAM J. Computing, 3(23) (1994), pp. 466 − 488.
11. K. Lang and S. Rao. Finding Near-Optimal Cuts: An Empirical Evaluation. 4th. ACM-SIAM Symposium on Discrete Algorithms, pp. 212 − 221, 1993.
12. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, S. Tragoudas, Fast approximation algorithms for multicommodity flow problems. Journal of Computer and System Sciences, 50(2), pp. 228 − 243, April 1995.
13. T. Leighton, S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM, vol 46 , No. 6 (Nov. 1999), pp. 787 − 832.
14. T. Lengauer. Combinatorial Algorithms for Integrated Circuit Layout. Stuttgart, John Wiley & Sons 1994.
15. D. W. Matula, F. Shahrokhi, Sparsest Cuts and Bottlenecks in Graphs. Journal of Disc. Applied Math., vol. 27 (1990), pp. 113 − 123.
16. F. Shahrokhi, D. W. Matula, On Solving Large Maximum Concurrent Flow Problems, Proceedings of ACM 1987 National Conference, pp. 205 − 209.
17. F. Shahroki, D. W. Matula, The maximum concurrent flow problem. Journal of the ACM, vol. 37, pp. 318 − 334, 1990.
18. N. Z. Shor, Methods of minimization of nondifferentiable functions and their applications. Kiev, "Naukova Dumka" 1979. (in Russian).
19. A. Steger and N. C. Wormald, Generating random regular graphs quickly. Combinatorics, Probab. and Comput. vol. 8 (1999), pp. 377 − 396.
20. M. Wang, S. K. Lim, J. Cong, M. Sarrafzadeh. Multi-way partitioning using bipartition heuristics. Proc. Asia and South Pacific Design Automation Conf., pp. 667 − 672, 2000.
21. Y. C. Wei, C. K. Cheng, An Improved Two-way Partitioning Algorithm with Stable Performance, IEEE Trans. on Computer-Aided Design, 1990, pp. 1502 − 1511.

22. Y. C. Wei, C. K. Cheng, A two-level two-way partitioning algorithm, Proc. Int'l. Conf. Computer- Aided Design, pp. 516 − 519, 1990.
23. Y. C. Wei, C. K. Cheng, Ratio Cut Partitioning for Hierarchical Designs. IEEE Trans. on Computer- Aided Design, vol. 10, pp. 911− 921, July 1991.
24. C. W. Yeh, C. K. Cheng, and T. T. Y. Lin. A probabilistic multicommodity-flow solution to circuit clustering problems. IEEE International Conference on Computer-Aided Design, pp. 428 − 431, 1992.
25. C. W. Yeh, C. K. Cheng, T. T. Y. Lin, An Experimental Evaluation of Partitioning Algorithms, IEEE International ASIC Conference, P14-1.1 − P14-1.4. 1991.