

An Efficient Algorithm for Graph Bisection of Triangularizations

Gerold Jäger

Department of Computer Science
Washington University
Campus Box 1045, One Brookings Drive
St. Louis, Missouri 63130-4899, USA
jaegerg@cse.wustl.edu

Abstract

Graph bisection is an elementary problem in graph theory. We consider the best known experimental algorithms and introduce a new algorithm called Longest-Path-Algorithm. Applying this algorithm to the cluster tree generation of hierarchical matrices, arising for example in discretizations of partial equations, we show that this algorithm outperforms previous algorithms.

Mathematics Subject Classification: 05C85, 68Q25, 68W20

Keywords: Graph bisection, hierarchical matrices, triangularizations

1 Introduction

Let $G = (V, E)$ be an undirected and unweighted graph with $|V| = n$. Generalizing the standard definition for odd n we define a *bisection* as a partition (X, Y) of V with $|X| = \lceil \frac{n}{2} \rceil$. The *bisection width* is defined as the minimum number of edges between X and Y among all possible bisections (X, Y) and MINBISECTION is the NP-hard problem of finding a bisection with minimum bisection width.

Saran and Vazirani [9] developed a polynomial-time algorithm approximating the bisection width by a factor of $n/2$ and showed that their algorithm does not approximate it with a better factor. Feige, Krauthgamer, Nissim [2] improved the approximation factor to $\sqrt{n} \log n$. For some classes we can compute the bisection width in polynomial time. Papadimitriou, Sideri [8] gave such an algorithm for grid graphs. Boppana [1] gave an algorithm based on eigenvalue

computation and the ellipsoid algorithm, which is able to compute a lower bound for the bisection width and equals it for a class of random graphs.

We consider some elementary algorithms, as the Simple-Greedy-Algorithm [7], the Kernighan-Algorithm [7] and the Randomized-Black-Holes-Algorithm [3]. Although only few complexity results are known about them, they give good experimental results. In this paper, we introduce a new elementary bisection algorithm. We show that its complexity is equal or better than the complexity of the previous algorithms. In experiments with planar graphs it beats the known algorithms and finds optimal solutions. Our application is the following problem which is important in many topics, e.g. solving partial differential equations and integral equations:

Consider high-dimensional matrices, which are dense, but might have a large rank. As addition, multiplication and inversion of those matrices are expensive, they are represented as *hierarchical matrices* (\mathcal{H} -matrices). The operations applied to the hierarchical matrices give the exact results with a small error term, but with an almost linear complexity. The hierarchical matrices consist of some blocks of different size each having a small rank. The partition of the hierarchical matrices is described by cluster trees and can be viewed as a graph (for details see [4], [5], [6]). Finding a small bisection width of this graph enables us to efficiently compute the above operations. We apply the algorithms to the following typical example. We partition the matrices by triangularization, with refining at different places: uniform refinement, refinement at one side, refinement at all sides.

2 Previous Algorithms

For the algorithms we need the following definition:

Definition 1. Let $G = (V, E)$ be a graph and $X, Y \subseteq V$ with $X \cap Y = \emptyset$ and $X \cup Y = V$.

a) For $x \in X$ denote with $I(x)$ the inner costs, i.e. the number of edges $(x, z) \in E$ with $z \in X \setminus \{x\}$. Analogously we define $I(y)$ for $y \in Y$.

b) For $x \in X$ denote with $O(x)$ the outer costs, i.e. the number of edges $(x, z) \in E$ with $z \in Y$. Analogously we define $O(y)$ for $y \in Y$.

c) For $x \in X, y \in Y$ let $\omega(x, y) := \begin{cases} 1, & \text{if } (x, y) \in E \\ 0, & \text{otherwise} \end{cases}$.

d) For $x \in X, y \in Y$ let $S(x, y) := O(x) - I(x) + O(y) - I(y) - 2\omega(x, y)$.

2.1 Simple-Greedy-Algorithm

The following elementary algorithm starts with a random bisection and swaps two vertices of different sides of the bisection obtaining a better bisection, until there is no improvement by this method. As the inner costs become outer costs and the outer costs become inner costs by swapping two vertices x, y , we get an improvement, if and only if $S(x, y) > 0$.

Algorithm 1. (Simple-Greedy)

Input Graph $G = (V, E)$ with $|V| = n$.

- 1 Choose a bisection (X, Y) , uniformly at random among all possible bisections.
- 2 Choose $x \in X, y \in Y$ with $S(x, y) > 0$.
- 3 Swap vertices x and y .
- 4 Repeat steps 2 and 3, until there are no $x \in X, y \in Y$ with $S(x, y) > 0$.

Output Bisection (X, Y)

As the bisection width is smaller than n^2 , the steps 2 and 3 can be executed at most n^2 times. Since in each execution of step 2 the value $S(x, y)$ is computed at most $\frac{n}{2} \cdot \frac{n}{2}$ times, it follows:

Remark 1. The Simple-Greedy-Algorithm needs $O(n^4)$ steps.

2.2 Kernighan-Lin-Algorithm

The following algorithm of Kernighan, Lin [7] is a generalization of the Simple-Greedy-Algorithm. We do some swaps, even if there is no improvement after the swaps, but there might be a later improvement.

Algorithm 2. (Kernighan, Lin)

Input Graph $G = (V, E)$ with $|V| = n$.

- 1 Choose a bisection (X, Y) , uniformly at random among all possible bisections.
- 2 Copy (X, Y) to (X', Y') .
- 3 Choose $x \in X', y \in Y'$ with maximum $S(x, y)$ (it might be $S(x, y) \leq 0$).
- 4 Swap vertices x and y .

- 5 $X' := X' \setminus \{y\}, Y' := Y' \setminus \{x\}$.
- 6 Repeat steps 3 to 5, until $X' := \emptyset, Y' := \emptyset$.
- 7 Choose the bisection (X, Y) as the bisection with minimum bisection width among all bisections received after step 4.
- 8 Repeat steps 2 to 7, until we cannot get any improvement by these steps.

Output Bisection (X, Y)

Instead of one execution of the steps 2 and 3 in the Simple-Greedy-Algorithm, we execute $\frac{n}{2}$ times the steps 3 to 5 of the Kernighan-Lin-Algorithm. So we obtain:

Remark 2. The Kernighan-Lin-Algorithm needs $O(n^5)$ steps.

2.3 Randomized-Black-Holes-Algorithm

The following algorithm of Ferencz et al. [3] starts with two empty sets X, Y , called the black holes, and alternately adds to both sets one vertex, until we have a bisection.

Algorithm 3. (Randomized-Black-Holes)

Input Graph $G = (V, E)$ with $|V| = n$.

- 1 $X := \emptyset, Y := \emptyset$.
- 2 Choose uniformly at random an edge between $V \setminus \{X \cup Y\}$ and X and add the corresponding vertex in $V \setminus \{X \cup Y\}$ to X . If there is no such edge, choose uniformly at random a vertex among all vertices in $V \setminus \{X \cup Y\}$ and add it to X .
- 3 Do step 2 for Y .
- 4 Repeat steps 2 and 3, until (X, Y) is a bisection.

Output Bisection (X, Y)

Since in each execution of step 2 or 3 we have to test at most n^2 edges and the steps 2 or 3 are executed n times, it follows:

Remark 3. The Randomized-Black-Holes-Algorithm needs $O(n^3)$ steps.

3 Longest-Path-Algorithm

For planar graphs, it is reasonable to partition the vertices into two classes, which are separated by only one line. We should get such a separation, if we start from two vertices with large distance.

So we choose one arbitrary vertex z . By repeatedly determining the neighborhoods of z , we find another vertex x , as far as possible from z . After repeating this process with x , we obtain y . We start with $\{x\}$ and $\{y\}$ and iteratively add the neighborhoods to the previous sets, until we have a bisection.

Algorithm 4. (Longest-Path)

Input Graph $G = (V, E)$ with $|V| = n$.

- 1 Choose uniformly at random a vertex z .
- 2 $Z := \{z\}$.
- 3 List all neighbors of vertices from Z and add it to Z .
- 4 Repeat step 3, until $Z = V$.
- 5 Choose one of the vertices, added in the last execution of step 3, and denote it with x .
- 6 Repeat steps 2 to 5 with x instead of z . The resulting vertex is denoted with y .
- 7 $X := \{x\}, Y := \{y\}$.
- 8 List all neighbors of vertices from X and add each neighbor to X , except for $|X| \geq \lceil \frac{n}{2} \rceil$.
- 9 Repeat step 8, if $|X| < \lceil \frac{n}{2} \rceil$.
- 10 List all neighbors of vertices from Y and add each neighbor to Y , except for $|Y| \geq \lfloor \frac{n}{2} \rfloor$.
- 11 Repeat step 10, if $|Y| < \lfloor \frac{n}{2} \rfloor$.
- 12 Add the remaining vertices in an arbitrary way, so that $|X| = \lceil \frac{n}{2} \rceil$ and $|Y| = \lfloor \frac{n}{2} \rfloor$.

Output Bisection (X, Y)

As in the steps 3,8 and 10 for every added vertex we have to test at most n^2 edges, we obtain the same complexity as for the Randomized-Black-Holes-Algorithm:

Remark 4. The Longest-Path-Algorithm needs $O(n^3)$ steps.

As the Simple-Greedy-Algorithm should reduce the length of the line separating the two sides of the bisection, we execute this algorithm after the Longest-Path-Algorithm. In this case, for the Simple-Greedy-Algorithm it is sufficient to test only the vertices, which have at least one neighbor at the other side of the bisection.

4 Experimental Results

We have tested the Simple-Greedy-Algorithm (SG), the Kernighan-Lin-Algorithm (KL), the Randomized-Black-Holes-Algorithm (RBH), the Longest-Path-Algorithm (LP) and the Longest-Path-Algorithm followed by the Simple-Greedy-Algorithm (LP+ SG) for the examples described in the introduction. We compare their bisection widths with the optimal ones and additionally compare their execution times. Finally we graphically present one typical example of all three graph classes. We color all areas at the one side of the bisection with red and all areas at the other side of the bisection with green. Areas between the sides of the bisection are not colored. For the Longest-Path-Algorithm, we additionally show, where are the starting vertices x and y . In the tables let n be the number of vertices and m the number of edges of the graph.

4.1 Uniform refinement

From the structure of the graph, the optimal bisection width of step k of refinement can easily be shown to be $2^{k+1} + 1$.

The Kernighan-Lin-Algorithm and the Randomized-Black-Holes-Algorithm produce better results than the Simple-Greedy-Algorithm. Although they need much more execution time, they give considerably worse results than the two versions of the Longest-Path-Algorithm. The Simple-Greedy-Algorithm after the Longest-Path-Algorithm leads only to a small improvement. With the Longest-Path-Algorithm followed by the Simple-Greedy-Algorithm, we find an optimal solution or a solution, only larger by 1, in comparison to the optimal one.

4.2 Refinement at one side

For these graphs, the optimal bisection width of step k of refinement is $2k + 3$. The Longest-Path-Algorithm followed by the Simple-Greedy-Algorithm finds an optimal solution in all cases. The other results are similar to the case of uniform refinement.

4.3 Refinement at all sides

As there is no trivial formula for the optimal bisection width, we calculate it separately for every step of refinement.

The Simple-Greedy-Algorithm after the Longest-Path-Algorithm gives a much smaller bisection width. Except for step $k = 7$ of refinement, we find an optimal or almost optimal solution. The case $k = 7$ gives such a bad result, as there are two green areas in the red area, and so there is not only one line separating the two areas. For regular structures like rectangular grid graphs or the graph of uniform refinement of section 4.1 this case cannot appear.

5 Conclusions

In most cases the Longest-Path-Algorithm followed by the Simple-Greedy-Algorithm finds bisections with minimum bisection width. These bisection widths are considerably smaller than those that can be obtained from previous algorithms. The execution times as well as the complexity of the algorithm is better or equal to previous algorithms.

Acknowledgement

I would like to thank Wolfgang Hackbusch and Anand Srivastav for introducing me to the partitioning problem of hierarchical matrices as well as for the idea of the Longest-Path-Algorithm and Lars Grasedyck for some hints for implementing and graphically presenting the graphs.

The research was funded in part by the United States National Science Foundation grant IIS-053525.

References

- [1] R.B. Boppana, Eigenvalues and Graph Bisection: An Average-Case Analysis, IEEE Symposium on Foundations of Computer Science (FOCS) (1987), 280-285.

- [2] U. Feige, R. Krauthgamer and K. Nissim, Approximating the Minimum Bisection Size, Annual ACM Symposium on Theory of Computing (STOC) (2000), 530-536.
- [3] A. Ferencz, R. Szweczyk, J. Weinstein and J. Wilkening, Graph Bisection, Final Report, University of California at Berkeley (1999).
- [4] L. Grasedyck, Theorie und Anwendungen Hierarchischer Matrizen, Ph. D. Thesis, University of Kiel (2001).
- [5] W. Hackbusch, A Sparse Matrix Arithmetic Based on \mathcal{H} -Matrices. Part I: Introduction to \mathcal{H} -Matrices, Computing, 62(2) (1999), 89-108.
- [6] W. Hackbusch and B.N. Khoromskij, A Sparse \mathcal{H} Arithmetic. Part II: Application to Multi-Dimensional Problems, Computing, 64 (2000), 21-47.
- [7] B.W. Kernighan and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs, The Bell System Technical Journal, 49(2) (1970), 291-307.
- [8] C.H. Papadimitriou and M. Sideri, The Bisection Width of Grid Graphs, Mathematical Systems Theory, 29 (1996), 97-110.
- [9] H. Saran and V.V. Vazirani, Finding k Cuts within Twice the Optimal, SIAM J. Comput., 24(1) (1995), 101-108.

Received: October 19, 2006

| | | SG | KL | RBH | LP | LP+SG | OPT |
|----------------------------|-----------------|----------|-----------|----------|----------|----------|-----|
| $n = 9$ $m = 16$ | Bisection width | 5 | 5 | 7 | 6 | 5 | 5 |
| | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 25$ $m = 56$ | Bisection width | 12 | 9 | 9 | 9 | 9 | 9 |
| | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 81$ $m = 208$ | Bisection width | 55 | 37 | 31 | 20 | 18 | 17 |
| | Execution time | 00:00:00 | 00:00:02 | 00:00:01 | 00:00:00 | 00:00:00 | – |
| $n = 289$ $m = 800$ | Bisection width | 98 | 88 | 69 | 35 | 33 | 33 |
| | Execution time | 00:00:00 | 00:02:33 | 00:00:37 | 00:00:00 | 00:00:00 | – |
| $n = 1089$ $m = 3136$ | Bisection width | 442 | 207 | 105 | 69 | 66 | 65 |
| | Execution time | 00:00:05 | 05:44:04 | 00:33:46 | 00:00:02 | 00:00:06 | – |
| $n = 4225$ $m = 12416$ | Bisection width | 1890 | 1501 | 261 | 131 | 129 | 129 |
| | Execution time | 00:01:23 | 240:27:19 | 33:20:59 | 00:00:32 | 00:01:35 | – |
| $n = 16641$ $m = 49408$ | Bisection width | 7750 | | | 265 | 258 | 257 |
| | Execution time | 00:22:53 | | | 00:08:35 | 00:25:11 | – |

Table 1: Triangularization graphs with uniform refinement

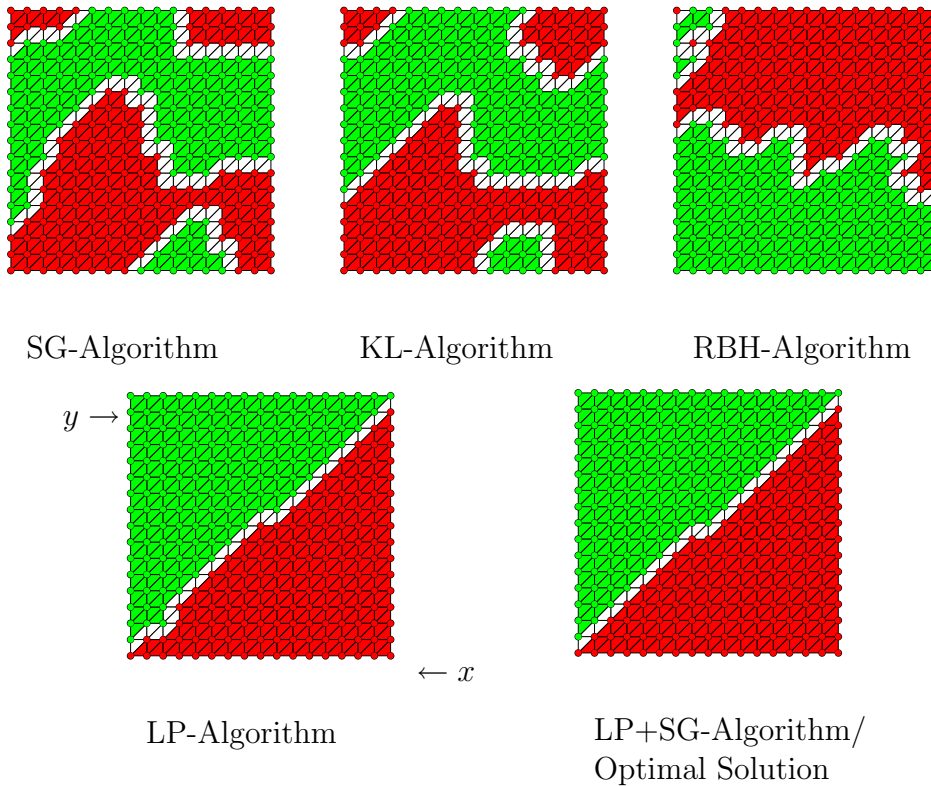


Figure 1: Example for uniform refinement: $n = 289, m = 800$

| | | SG | KL | RBH | LP | LP+SG | OPT |
|-------------|-----------------|----------|----------|----------|----------|----------|-----|
| $n = 9$ | Bisection width | 5 | 5 | 7 | 6 | 5 | 5 |
| $m = 16$ | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 18$ | Bisection width | 7 | 7 | 11 | 7 | 7 | 7 |
| $m = 39$ | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 35$ | Bisection width | 18 | 11 | 9 | 9 | 9 | 9 |
| $m = 84$ | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 68$ | Bisection width | 11 | 11 | 27 | 15 | 11 | 11 |
| $m = 173$ | Execution time | 00:00:00 | 00:00:03 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 133$ | Bisection width | 33 | 31 | 39 | 17 | 13 | 13 |
| $m = 350$ | Execution time | 00:00:00 | 00:00:21 | 00:00:03 | 00:00:00 | 00:00:00 | – |
| $n = 262$ | Bisection width | 81 | 109 | 69 | 34 | 15 | 15 |
| $m = 703$ | Execution time | 00:00:00 | 00:02:09 | 00:00:27 | 00:00:00 | 00:00:00 | – |
| $n = 519$ | Bisection width | 173 | 149 | 49 | 50 | 17 | 17 |
| $m = 1408$ | Execution time | 00:00:02 | 00:21:33 | 00:03:27 | 00:00:00 | 00:00:01 | – |
| $n = 1032$ | Bisection width | 444 | 315 | 152 | 39 | 19 | 19 |
| $m = 2817$ | Execution time | 00:00:05 | 02:50:37 | 00:27:15 | 00:00:02 | 00:00:05 | – |
| $n = 2057$ | Bisection width | 546 | | 87 | 43 | 21 | 21 |
| $m = 5634$ | Execution time | 00:00:22 | | 03:37:28 | 00:00:07 | 00:00:22 | – |
| $n = 4106$ | Bisection width | 722 | | | 68 | 23 | 23 |
| $m = 11267$ | Execution time | 00:01:37 | | | 00:00:29 | 00:01:27 | – |
| $n = 8203$ | Bisection width | 2409 | | | 74 | 25 | 25 |
| $m = 22532$ | Execution time | 00:06:17 | | | 00:01:58 | 00:05:56 | – |
| $n = 16396$ | Bisection width | 5821 | | | 76 | 27 | 27 |
| $m = 45061$ | Execution time | 00:21:49 | | | 00:07:54 | 00:23:08 | – |

Table 2: Triangularization graphs with refinement at one side

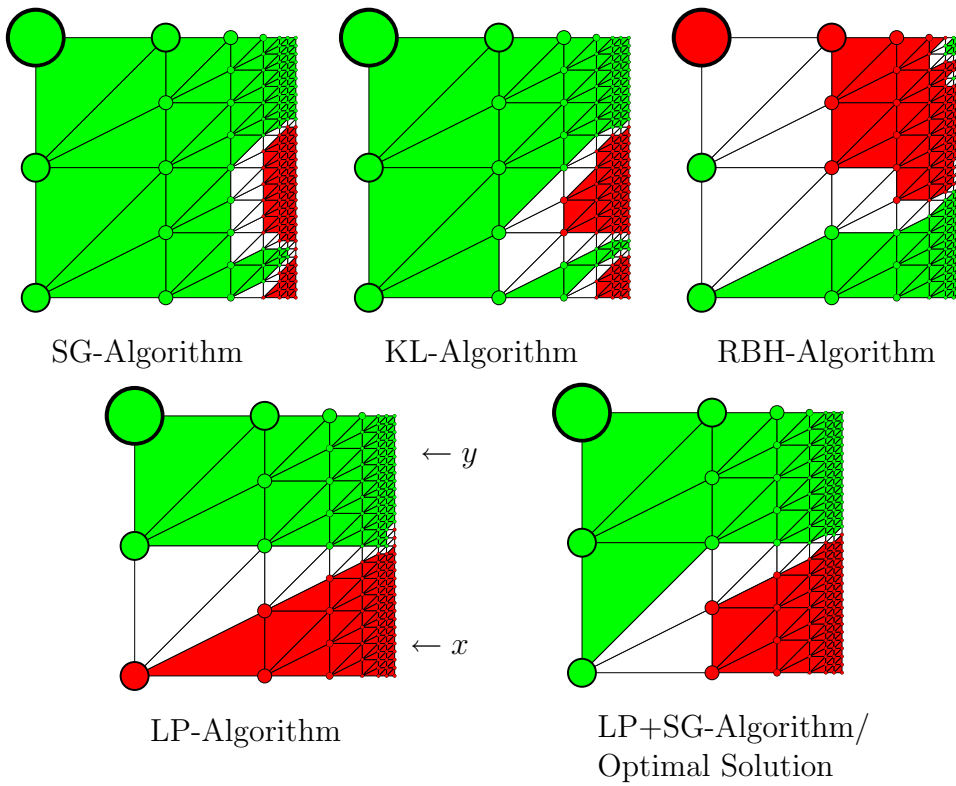


Figure 2: Example for refinement at one side: $n = 133, m = 350$

| | | SG | KL | RBH | LP | LP+SG | OPT |
|----------------------------|-----------------|----------|----------|----------|----------|----------|-----|
| $n = 9$ $m = 16$ | Bisection width | 5 | 5 | 7 | 6 | 5 | 5 |
| | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 25$ $m = 56$ | Bisection width | 12 | 9 | 9 | 9 | 9 | 9 |
| | Execution time | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | 00:00:00 | – |
| $n = 73$ $m = 184$ | Bisection width | 21 | 21 | 31 | 16 | 16 | 15 |
| | Execution time | 00:00:00 | 00:00:02 | 00:00:01 | 00:00:00 | 00:00:00 | – |
| $n = 185$ $m = 488$ | Bisection width | 64 | 59 | 75 | 38 | 19 | 19 |
| | Execution time | 00:00:00 | 00:00:44 | 00:00:09 | 00:00:00 | 00:00:00 | – |
| $n = 425$ $m = 1144$ | Bisection width | 134 | 103 | 73 | 64 | 25 | 25 |
| | Execution time | 00:00:01 | 00:15:58 | 00:01:58 | 00:00:00 | 00:00:01 | – |
| $n = 921$ $m = 2504$ | Bisection width | 238 | 275 | 105 | 118 | 32 | 32 |
| | Execution time | 00:00:04 | 02:44:46 | 00:20:07 | 00:00:01 | 00:00:04 | – |
| $n = 1929$ $m = 5272$ | Bisection width | 432 | | | 224 | 97 | 34 |
| | Execution time | 00:00:22 | | | 00:00:07 | 00:00:20 | – |
| $n = 3961$ $m = 10856$ | Bisection width | 902 | | | 418 | 44 | 43 |
| | Execution time | 00:01:28 | | | 00:00:27 | 00:01:23 | – |
| $n = 8041$ $m = 22072$ | Bisection width | 2720 | | | 808 | 47 | 45 |
| | Execution time | 00:05:25 | | | 00:01:53 | 00:05:46 | – |
| $n = 16217$ $m = 44552$ | Bisection width | 5026 | | | 1580 | 53 | 53 |
| | Execution time | 00:22:24 | | | 00:07:45 | 00:23:29 | – |

Table 3: Triangularization graphs with refinement at all sides

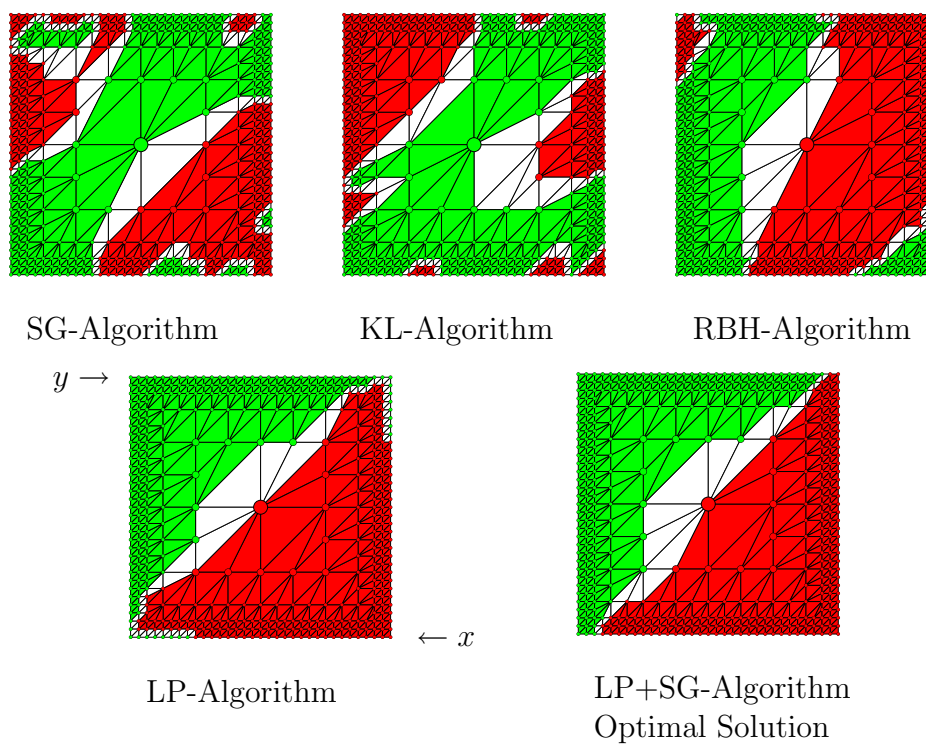


Figure 3: Example for refinement at all sides: $n = 425, m = 1144$