

KaDraw v1.0 – Karlsruhe Graph Drawing User Guide

Henning Meyerhenke, Martin Nöllenburg and Christian Schulz

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Email: {meyerhenke, noellenburg, christian.schulz}@kit.edu

Abstract

This paper serves as a user guide to the graph drawing framework KaDraw (Karlsruhe Graph Drawing). We give a rough overview of the techniques used within the framework and describe the user interface as well as the file formats used.

Contents

1	Introduction	2
2	Graph Drawing Algorithms within KaDraw	2
3	Graph Format	3
3.1	Input File Format	3
3.2	Output File Formats	4
3.2.1	Coordinate File	4
3.2.2	PDF/PNG files	4
3.3	Troubleshooting	4
4	User Interface	5
4.1	KaFFPa	5
4.2	KaFFPaE / KaBaPE	6
4.3	Node Separators	7
4.4	Label Propagation	7
4.5	Graph Format Checker	7

1 Introduction

Drawing large networks (or graphs) with hundreds of thousands of nodes and edges has a variety of relevant applications. One of them can be interactive visualization, which helps humans working on graph data to gain insights about the properties of the data. If a very large high-end display is not available for such purpose, a hierarchical approach allows the user to select an appropriate zoom level [1]. Moreover, drawings of large graphs can also be used as a preprocessing step in high-performance applications [6].

One very promising class of layout algorithms in this context is based on the *stress* of a graph. Such algorithms can for instance be used for drawing graphs with fixed distances between vertex pairs, provided *a priori* in a distance matrix [3]. More recently, Gansner et al. [2] proposed a similar model that includes besides the stress an additional entropy term (hence its name *maxent-stress*). While still using shortest path distances, this model often results in more satisfactory layouts for large networks. The optimization problem can be cast as solving Laplacian linear systems successively. Since each right-hand side in this succession depends on the previous solution, many linear systems need to be solved until convergence.

Within KaDraw we present a novel multilevel algorithm, MulMent, to compute a graph layout with respect to maxent-stress. As opposed to previous work, we do not solve the linear systems of the maxent-stress metric with a typical numerical solver. Instead we use a simple local iterative scheme within a multilevel approach. To accelerate local optimization, we approximate long-range forces and parallelism.

2 Graph Drawing Algorithms within KaDraw

We now give a rough overview over the algorithms implemented in our framework. For details on the algorithms, we refer the interested reader to the corresponding paper [8]. A successful (meta)heuristic for graph drawing (and other optimization problems on large graphs) is the multilevel approach. We employ it for maxent-stress optimization as well. A multilevel algorithm computes a sequence of increasingly coarse but structurally related graphs as abstractions of the original graph. Starting from a layout of the coarsest graph, incremental refinement steps using the previous layout as a scaffold eventually produce a layout of the entire input graph, where the refinement steps are fast due to the good initial layouts.

We briefly sketch our algorithmic approach: The method for creating the graph hierarchy is based on fast graph clustering with controllable cluster sizes [7]. Each cluster computed on one hierarchy level is contracted into a new supervertex for the next level. After computing an initial layout on the coarsest hierarchy level, we improve the drawing on each finer level by iterating the maxent-equation to solve the maxent-stress system

$$x_u \leftarrow \frac{1}{\rho_u} \sum_{\{u,v\} \in E} w_{uv} \left(x_v + d_{uv} \frac{x_u - x_v}{\|x_u - x_v\|} \right) + \frac{\alpha}{\rho_u} \sum_{\{u,v\} \notin E} \frac{x_u - x_v}{\|x_u - x_v\|^2}, \quad (1)$$

where $\rho_u = \sum_{\{u,v\} \in E} w_{uv}$ and appropriately chosen weights and distances. Additionally, this process exploits the hierarchy and draws vertices that are densely connected with each other (i.e. which are in the same cluster) close to each other. Note that the local optimization algorithm presented above has a theoretical running time of $\mathcal{O}(n^2)$ per iteration. To speed this up, our algorithm uses approximations for the distances in the entropy term in Eq. (1). We do this by taking the cluster structure computed during coarsening into account, i.e. we use graphs and coordinates multiple levels beneath the current level under consideration to approximate the second summation in Eq. (1). For more details, we refer the reader to the paper [8].

Shared-memory Parallelization. Our local optimization algorithms are shared-memory parallelized using OpenMP. An iteration of the local optimizer works as follows: Since new coordinates of the vertices in the same iteration can be computed independently, we use multiple threads to do so. Parallelism is also used analogously when working on different levels for the distance approximations in the entropy term.

3 Graph Format

3.1 Input File Format

The graph format used by our partitioning programs is the same as used by Metis [5], Chaco [4] and the graph format that has been used during the 10th DIMACS Implementation Challenge on Graph Clustering and Partitioning. The input graph has to be undirected, without self-loops and without parallel edges.

To give a description of the graph format, we follow the description of the Metis 4.0 user guide very closely. A graph $G = (V, E)$ with n vertices and m edges is stored in a plain text file that contains $n + 1$ lines (excluding comment lines). The first line contains information about the size and the type of the graph, while the remaining n lines contain information for each vertex of G . Any line that starts with % is a comment line and is skipped.

The first line in the file contains either two integers, $n\ m$, or three integers, $n\ m\ f$. The first two integers are the number of vertices n and the number of undirected edges of the graph, respectively. Note that in determining the number of edges m , an edge between any pair of vertices v and u is counted *only once* and not twice, i.e. we do not count the edge (v, u) from (u, v) separately. The third integer f is used to specify whether or not the graph has weights associated with its vertices, its edges or both. If the graph is unweighted then this parameter can be omitted. It should be set to 1 if the graph has edge weights, 10 if the graph has node weights and 11 if the graph has edge and node weights.

The remaining n lines of the file store information about the actual structure of the graph. In particular, the i th line (again excluding comment lines) contains information about the i th vertex. Depending on the value of f , the information stored in each line is somewhat different. In the most general form (when $f = 11$, i.e. we have node and edge weights) each line has the following structure:

$$c\ v_1\ w_1\ v_2\ w_2\ \dots\ v_k\ w_k$$

where c is the vertex weight associated with this vertex, v_1, \dots, v_k are the vertices adjacent to this vertex, and w_1, \dots, w_k are the weights of the edges. Note that the vertices are numbered starting from 1 (not from 0). Furthermore, the vertex-weights must be integers greater or equal to 0, whereas the edge-weights must be strictly greater than 0.

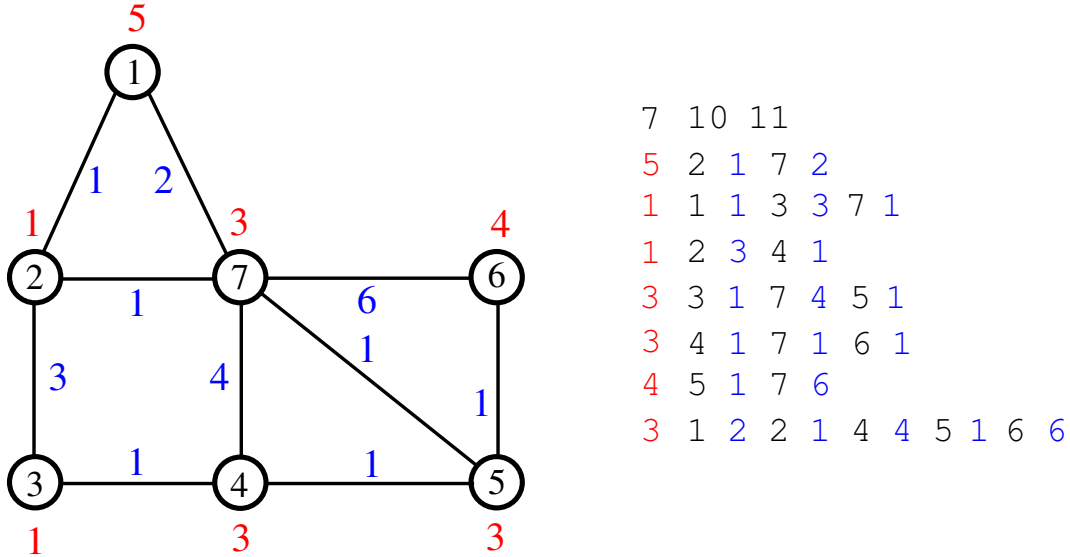


Figure 1: An example graph and its representation in the graph format. The IDs of the vertices are drawn within the circle, the vertex weight is shown next to the circle (red) and the edge weight is plotted next to the edge (blue).

3.2 Output File Formats

3.2.1 Coordinate File

The output format of coordinates is basically a text file. This file contains n lines. In each line the 2D-coordinate of the corresponding vertex is given, i.e. line i contains the coordinate of vertex i .

3.2.2 PDF/PNG files

If the output is a node separator then the same format as used for a partition is used. However, in this case the nodes of the separator get the block ID k where as the other nodes maintain their original block id.

3.3 Troubleshooting

KaDraw should not crash! If KaDraw crashes it is mostly due to the following reasons: the provided graph contains self-loops or parallel edges, there exists a forward edge but the backward edge is missing or the forward and backward edges have different weights, or the number of vertices or edges specified does not match the number of vertices or edges provided in the file. Please use the *graphcheck* tool provided in our graph partitioning package to verify whether your graph has the right input format. If our graphcheck tool tells you that the graph that you provided has the correct format and KaDraw crashes anyway, please write us an email.

4 User Interface

KaHIP contains the following programs: kaffpa, kaffpaE, partition_to_vertex_separator, graphchecker. To compile these programs you need to have Argtable, g++, OpenMPI and scons installed (we use argtable-2.10, g++-4.8.0, OpenMPI-1.4.1 and scons-1.2). Once you have that you can execute *compile.sh* in the main folder of the release. When the process is finished the binaries can be found in the folder *deploy*. We now explain the parameters of each of the programs briefly.

4.1 KaFFPa

Description: This is the multilevel graph partitioning program.

Usage:

```
kaffpa file --k=<int> [--help] [--seed=<int>] [--preconfiguration=variant] [--imbalance=<double>]
      [--time_limit=<double>] [--enforce_balance] [--input_partition=<string>]
      [--balance_edges] [--output_filename=<string>]
```

Options:

file	Path to graph file that you want to partition.
--k=<int>	Number of blocks to partition the graph into.
--help	Print help.
--seed=<int>	Seed to use for the random number generator.
--preconfiguration=variant	Use a preconfiguration. (Default: eco) [strong eco fast fastsocial ecosocial strongsocial]. Strong should be used if quality is paramount, eco if you need a good tradeoff between partition quality and running time, and fast if partitioning speed is in your focus. Configurations with a social in their name should be used for social networks and web graphs.
--imbalance=<double>	Desired balance. Default: 3 (%).
--time_limit=<double>	Time limit in seconds s. The default value is set to 0s, i.e. one partitioner call will be made. If you set a time limit t , kaffpa will repeatedly call the multilevel method until the time limit is reached and return the best solution found.
--enforce_balance	Use this option only on graphs without vertex weights. If this option is enabled, kaffpa guarantees that the output partition is feasible, i.e. fulfills the specified balance constraint.
--balance_edges	Use this option to balance the edges among the blocks as well as the nodes. In this case node weights are set to $c(v) + \deg_{\omega}(v)$ for a node $v \in V$ and the balance constraint is adapted accordingly.
--input_partition=<string>	You can specify an input partition. If you do so, KaFFPa will try to improve it.
--output_filename=<string>	Specify the output filename (default tmppartition\$k).

4.2 KaFFPaE / KaBaPE

Description: This is the distributed evolutionary algorithm to tackle the graph partitioning problem. It includes also the perfectly balance case $\epsilon = 0$.

Usage:

```
mpirun -n P kaffpaE file --k=<int> [--help] [--seed=<int>] [--preconfiguration=variant] [--imbalance=<double>]
                    [--time_limit=<double>] [--mh_enable_quickstart] [--mh_optimize_communication_volume]
                    [--mh_enable_kabapE] [--mh_enable_tabu_search] [--kabaE_internal_bal=<double>]
                    [--input_partition=<string>] [--balance_edges] [--output_filename=<string> ]
```

Options:

P	Number of processes to use.
file	Path to graph file that you want to partition.
--k=<int>	Number of blocks to partition the graph into.
--help	Print help.
--seed=<int>	Seed to use for the random number generator.
--preconfiguration=variant	Use a preconfiguration. (Default: strong) [strong eco fast fastsocial ecosocial strongsocial]. Strong should be used if quality is paramount, eco if you need a good tradeoff between partition quality and running time, and fast if partitioning speed is in your focus. Configurations with a social in their name should be used for social networks and web graphs.
--imbalance=<double>	Desired balance. Default: 3 (%).
--time_limit=<double>	Time limit in seconds s. The default value is set to 0s, i.e. one partitioner call will be made by each PE. In order to use combine operations you <i>have to</i> set a time limit $t > 0$. kaffpaE will return the best solution after the time limit is reached. A time limit $t = 0$ means that the algorithm will only create the initial population.
--mh_enable_quickstart	Enables the quickstart option. In this case each PE creates a few partitions and these partitions are distributed among the PEs.
--mh_optimize_communication_volume	Modifies the fitness function of the evolutionary algorithm so that communication volume is optimized.
--mh_enable_kabapE	Enables the combine operator of <i>KaBaPE</i> .
--mh_enable_tabu_search	Enables our version of combine operation by block matching.
--kabaE_internal_bal=<double>	The internal balance parameter for KaBaPE (Default: 0.01) (1 %)
--balance_edges	Use this option to balance the edges among the blocks as well as the nodes. In this case node weights are set to $c(v) + \deg_{\omega}(v)$ for a node $v \in V$ and the balance constraint is adapted accordingly.
--input_partition=<string>	You can specify an input partition. If you do so, KaFFPaE will try to improve it.
--output_filename=<string>	Specify the output filename (default tmppartition\$k).

4.3 Node Separators

Description: This is the program that computes a k -way vertex separator given a k -way partition of the graph.

Usage:

```
partition_to_vertex_separator file --k=<int> --input_partition=<string> [--help] [--seed=<int>] [--output_filename]
```

Options:

file	Path to the graph file.
--k=<int>	Number of blocks the graph is partitioned in by using the input partition.
--input_partition=<string>	Input partition to compute the vertex separator from.
--help	Print help.
--seed=<int>	Seed to use for the random number generator.
--output_filename=<string>	Specify the output filename (default tmpseparator).

4.4 Label Propagation

Description: This is the program that a size-constrained label propagation clustering.

Usage:

```
label_propagation file [--cluster_upperbound=<int>] [--label_propagation_iterations] [--help] [--seed=<int>]  
[--output_filename]
```

Options:

file	Path to the graph file.
--cluster_upperbound=<int>	Set a size-constraint on the size of a cluster. For example, specifying a value of 10 means that each cluster can have at most 10 vertices (or a weight of 10 if the graph is weighted). By default there is no size-constraint, so that each node can potentially be in on cluster.
--label_propagation_iterations=<int>	Set the number of label propagation iterations. The default value is 10.
--help	Print help.
--seed=<int>	Seed to use for the random number generator.
--output_filename=<string>	Specify the output filename (default tmpclustering).

4.5 Graph Format Checker

Description: This program checks if the graph specified in a given file is valid.

Usage:

```
graphchecker file
```

Options:

file	Path to the graph file.
------	-------------------------

References

- [1] J. Abello, F. van Ham, and Neeraj K. Ask-graphview: A large scale graph visualization system. *IEEE Trans. Visualization and Computer Graphics*, 12(5):669–676, 2006.
- [2] E. R. Gansner, Y. Hu, and S. C. North. A maxent-stress model for graph layout. *IEEE Trans. Visualization and Computer Graphics*, 19(6):927–940, 2013.
- [3] E. R. Gansner, Y. Koren, and S. North. Graph drawing by stress majorization. In *Graph Drawing (GD’04)*, volume 3383 of *LNCS*, pages 239–250. Springer, 2005.
- [4] B. Hendrickson. Chaco: Software for Partitioning Graphs. <http://www.cs.sandia.gov/~bahendr/chaco.html>.
- [5] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.
- [6] S. Kirmani and P. Raghavan. Scalable parallel graph partitioning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC ’13, pages 51:1–51:10, New York, NY, USA, 2013. ACM.
- [7] H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning Complex Networks via Size-constrained Clustering. In *13th Int. Symp. on Exp. Algorithms*, LNCS. Springer, 2014.
- [8] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. Drawing large graphs by multilevel maxent-stress optimization. Technical report.