Università della Svizzera italiana | Faculty of Informatics | Institute of Computing CI

Numerical Computing, Fall Semester 2022
Lecturer: Prof. O. Schenk
Assistants: E. Vecchi, L. Gaedke-Merzhäuser

Project 3 – Graph Partitioning
Due date: Wednesday, 9 November 2022, 11:59 PM

Graph Partitioning

In computational science and high-performance computing, the graph partition problem is defined on data represented in the form of a graph $\mathcal{G} = (V, E)$ with V vertices and E edges, such that it is possible to partition G into smaller components with specific properties. For instance, a k-way partition divides the vertex set into k smaller components. In general, good solutions for the graph partitioning problem require that cuts are small and partitions have equal size. The problem arises, for instance, when assigning work to a parallel computer. In order to achieve efficiency, the workload (partition size) should be balanced evenly among the processors while communication between them (edge cut) should be kept to a minimum. Other important application domains of graph partitioning, and a detailed overview of advances in the field can be found at [2].
Recently, the graph partition problem has gained importance due to its application for clustering and detection of cliques in social, pathological, and biological networks. Since graph partitioning is an NP-hard problem, practical solutions are based on heuristics. There are two broad categories of methods, local and global. Well known local methods are the Kernighan–Lin algorithm, and Fiduccia–Mattheyses algorithm, which were the first effective 2-way cuts by local search strategies. Their major drawback is the arbitrary initial partitioning of the vertex set, which can affect the final solution quality. Global approaches rely on properties of the entire graph and not on an arbitrary initial partition. The most common example is spectral partitioning, where a partition is derived from the spectrum of the graph Laplacian matrix. In cases where the nodes of the graph are associated with a coordinate list, inertial bisection is also a frequent choice of partitioning method.

Spectral Bisection

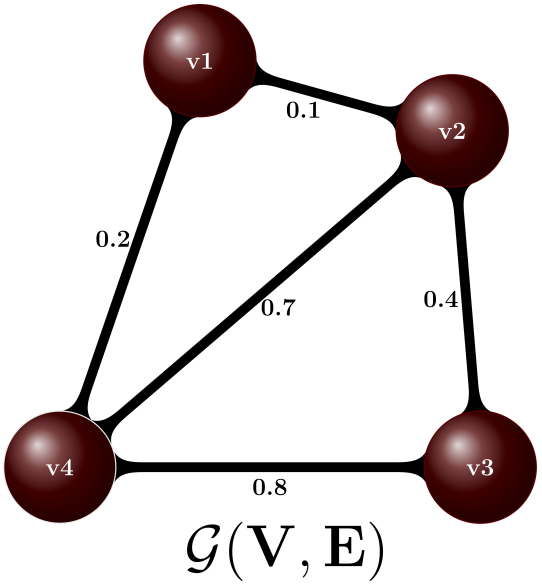
The spectral method was initially considered the standard to solve graph partitioning problems. It utilizes the spectral graph theorem of linear algebra, that enables the decomposition of a real symmetric matrix into eigenvalues, within an orthonormal basis of eigenvectors. For an undirected graph $\mathcal{G}(V, E)$, with vertex set $V = \{v_1, \cdots, v_n\}$ and two complementary subsets V_1, V_2, such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, we consider an indicator vector $\mathbf{x} \in \mathbb{R}^n$ such that

$$x_i = \begin{cases} 1\,, & i \in V_1, \\ 0\,, & i \in V_2. \end{cases}$$

Some of the n nodes of the graph are connected via m edges, each of which has a positive weight associated with it, thus

$$w_{ij} = \begin{cases} > 0, & \text{if } v_i \text{ connected to } v_j, \\ = 0 & \text{otherwise.} \end{cases}$$

A bisection is computed using the eigenvector associated with the second smallest eigenvalue of the graph Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n \times n}$, which encodes the degree $d_i = \sum_{j=1}^{n} w_{ij}$, i.e.; the number of connected edges, of each vertex along its diagonal, and the negative weights $-w_{ij}$. For an undirected and connected graph $G(V, E)$, as illustrated in Figure 1, with n vertices and m edges, the graph Laplacian can be realized in terms of the adjacency matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ and the degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ as follows. The graph Laplacian is a symmetric positive semidefinite



$$\mathcal{G}(\mathbf{V}, \mathbf{E})$$

$$\mathbf{W} := \sum_{i \in A, j \in B} w_{ij} = \begin{bmatrix} 0 & 0.1 & 0 & 0.2 \\ 0.1 & 0 & 0.4 & 0.7 \\ 0 & 0.4 & 0 & 0.8 \\ 0.2 & 0.7 & 0.8 & 0 \end{bmatrix}, \quad \mathbf{D} := \sum_{j=1}^{n} w_{ij} = \begin{bmatrix} 0.3 & 0 & 0 & 0 \\ 0 & 1.2 & 0 & 0 \\ 0 & 0 & 1.2 & 0 \\ 0 & 0 & 0 & 1.7 \end{bmatrix}$$

$$\mathbf{L} := \mathbf{D} - \mathbf{W} = \begin{bmatrix} 0.3 & -0.1 & 0 & -0.2 \\ -0.1 & 1.2 & -0.4 & -0.7 \\ 0 & -0.4 & 1.2 & -0.8 \\ -0.2 & -0.7 & -0.8 & 1.7 \end{bmatrix}.$$

Figure 1: A weighted, undirected and connected graph $G(V, E)$, with 4 vertices and 5 edges, with its degree $\mathbf{D}$, adjacency $\mathbf{W}$, and graph Laplacian $\mathbf{L}$ matrices.

matrix, with its smallest eigenvalue being $\lambda_1 = 0$, and the associated eigenvector $\mathbf{w}_1 = c\mathbf{1}$ [1]. The eigenvector $\mathbf{w}_2$ being the constant one associated with the second smallest eigenvalue λ_2 is Fiedler's celebrated eigenvector, and is crucial for spectral graph partitioning. Each node v_i of the graph is associated with one entry in $\mathbf{w}_2$. Thresholding the values of $\mathbf{w}_2$ around 0 results in two roughly balanced (equal sized) partitions, with minimum edgecut, while thresholding around the median value of $\mathbf{w}_2$ produces two strictly balanced partitions. The procedure to compute a bisection using spectral partitioning is summarized in Algorithm 1. For a much more detailed description of the method we refer to [4].

Inertial Bisection

Inertial bisection is a very different approach, relying on the geometric layout of the graph, i.e. geometric coordinates attached to the vertices of the graph. The main idea of the method is to find a hyperlane running though the "center of gravity of the points". In 2 dimensions (2D) such a line ℓ is chosen such that the sum of squares of the distance of the nodes to the line is minimized. It is defined by a point $\overline{P} = (\overline{x}, \overline{y})$ and a vector $\mathbf{u} = (u_1, u_2)$ with $\|u\|_2 = \sqrt{u_1^2 + u_2^2}$

[1] Please note that we are now using the variable $\mathbf{w}_i$ to refer to an eigenvector as opposed to the previous $\mathbf{v}_i$ to not confuse it with a vertex v_i of a graph.

Algorithm 1 Spectral bisection

Require: $\mathcal{G}(V, E)$,
Ensure: A bisection of $\mathcal{G}$
1: **function** SPECTRALBISECTION(graph $\mathcal{G}(V, E)$)
2: Form the graph Laplacian matrix $\mathbf{L}$
3: Calculate the second smallest eigenvalue λ_2 and its associated eigenvector $\mathbf{w}_2$.
4: Set 0 or the median of all components of $\mathbf{w}_2$ as threshold.
5: Choose $V_1 := \{v_i \in V | w_i < thres\}, V_2 := \{v_i \in V | w_i \geq thres\}$.
6: **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
7: **end function**

such that $\ell = \{\overline{P} + \alpha \mathbf{u} \,|\, \alpha \in \mathbb{R}\}$ (Figure 2). Let each node v_i have the coordinates (x_i, y_i). We then choose

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i, \tag{1}$$

as the center of mass lies on the line ℓ. Finally, we need to compute $\mathbf{u}$ in order to minimize the sum of distances

$$\begin{aligned}\sum_{i=1}^{n} d_i^2 &= \sum_{i=1}^{n} (x_i - \overline{x})^2 + (y_i - \overline{y})^2 - (u_1(x_i - \overline{x}) + u_2(y_i - \overline{y}))^2 \\ &= \underbrace{\left(\sum_{i=1}^{n}(x_i - \overline{x})^2\right)}_{S_{xx}} u_1^2 + \underbrace{\left(\sum_{i=1}^{n}(y_i - \overline{y})^2\right)}_{S_{yy}} u_2^2 + 2\underbrace{\left(\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})\right)}_{S_{xy}} u_1 u_2 \\ &= \mathbf{u}^T \begin{bmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{bmatrix} \mathbf{u} = \mathbf{u}^T \mathbf{M} \mathbf{u}, \end{aligned} \tag{2}$$

where S_{xx}, S_{xy}, S_{yy} are the sums as defined in the previous line. The resulting matrix M is symmetric[2], thus the minimum of (2) is achieved by choosing $\mathbf{u}$ to be the normalized eigenvector corresponding to the smallest eigenvalue of M. The procedure of bisecting a graph using inertial partitioning is summarized in Algorithm 2. We refer again to [4] and references therein for a thorough overview of the inertial method.

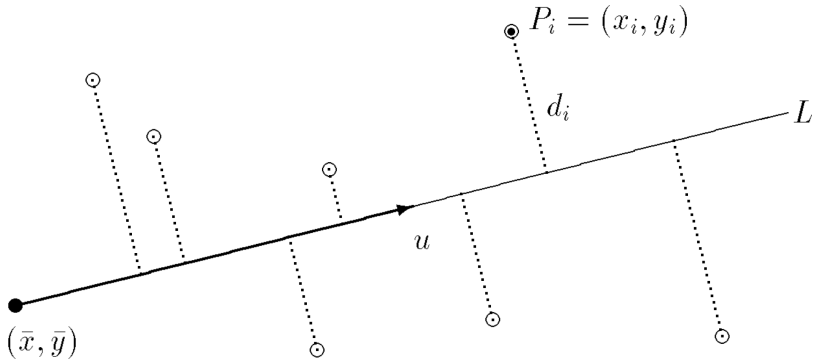


Figure 2: Illustration of inertial bisection in 2D [4].

[2]We can obtain matrix M by rewriting the quadratic form $Q(u_1, u_2)$ in matrix notation.

Algorithm 2 Inertial bisection.

Require: $\mathcal{G}(V, E), P_i = (x_i, y_i)$, $i = 1, \ldots, n$ the coordinates of the nodes
Ensure: A bisection of $\mathcal{G}$
1: **function** INERTIALBISECTION(graph $\mathcal{G}(V, E), P_i$)
2: Calculate the center of mass of the points ▷ acc. to (1)
3: Compute the eigenvector associated with the smallest eigenvalue of $\mathbf{M}$ ▷ $\mathbf{M}$ acc. to (2)
4: Partition the nodes of the graph around line L.
5: **return** V_1, V_2 ▷ bisection of $\mathcal{G}$
6: **end function**

The last task of the inertial partitioning routine (line 4), i.e partitioning nodes associated with geometric coordinates around a direction normal to the partitioning plane, is already implemented in the toolbox provided to you in function `partition.m`. The eigenvalue computations, both for spectral and inertial bisection should be performed with the in-Matlab function `eig`. Type `help eig`, or `help eigs` in your command line for more information.

Partitioning metrics

In what follows we will use the number of cut edges between partitions to determine the quality of the partitioning result. The size of an edge separator, or edgecut, which partitions the graph into a vertex subset and its complement $V_1 \cup V_2 = V$ is defined as follows:

$$\text{cut}(V_1, V_2) = \sum_{i \in V_1, j \in V_2} w_{ij}. \tag{3}$$

The calculation of cut edges is implemented in the function `cutsize.m` of our toolbox. The cardinality of each subset (i.e the number of nodes in each partition) is given by the 2-norm of indicator vector $\mathbf{x}$

$$\mathbf{x}^T \mathbf{x} = \|\mathbf{x}\|_2^2 = |V_1|. \tag{4}$$

Good scalability of parallel distributed memory solvers requires equally sized (balanced) cardinalities, so that the waiting time of the individual threads is minimized. For a k-way partition the load imbalance b_k is defined as the ratio of the highest partition cardinality over the average partition cardinality,

$$b^k = \max_{|V_i^k|} \frac{|V_i^k|}{|\widetilde{V}^k|}, \tag{5}$$

where $\widetilde{V}$ is the average partition weight. The load imbalance $b^k = 1 + b_r^k \geq 1$, where $b_r^k \geq 0$ characterizes the deviation from obtaining an evenly balanced partitioning. The optimal value for b_r^k is therefore zero, and it implies that the k partitions contain exactly the same number of nodes. In most applications it suffices to ensure that b_r^k is bounded from above by a desired threshold.

Software tools

We will use one external partioning software tool for this project. METIS[3] [6], probably the most widely used graph partitioning software, was developed by Karypis and Kumar and is probably the most widely used graph partitioning software. It consists of a set of serial programs for partitioning graphs, partitioning finite element meshes, and producing fill reducing orderings for sparse matrices. It is based on local techniques that iteratively improve starting solutions

[3] http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

by swapping nodes between the partitions that lie in the neighborhood of the cut. Kernighan and Lin [7] were the first to propose a local search method in this fashion. Their partition refinement strategy, which swaps pairs of vertices that result in the largest decrease in the size of the edgecut, was later accelerated by Fiduccia and Mattheyses [5]. METIS is based on this method, and subsequent improvements of it. Finally, in terms of computational efficiency, numerical experiments have demonstrated that graphs with several millions of vertices can be partitioned to 256 parts in a few seconds on current generation workstations and laptops using METIS. We will interface Julia and METIS through the Julia wrapper `Metis.jl` which allow us to use the functions `METIS_PartGraphRecursive` and `METIS_PartGraphKway` with the same arguments and argument order described in the Metis manual.

The assignment

You can find the graph partitioning toolbox `Tools` on the Moodle webpage. This toolbox contains Julia code for the creation of several graph and graph partitioning methods, e.g., coordinate bisection, and has routines to generate recursive multiway partitions and visualize the partitioning results.

Set up the environment

In order to use `Metis.jl` and the other packages, you can install the packages from the Julia package manager active session, which you can access by pressing the key `]` in the REPL:

```
julia> ]
(Project) pkg> add DelimitedFiles MAT Arpack LinearAlgebra Metis Random SparseArrays
  Statistics Graphs SGtSNEpi GLMakie Colors CairoMakie PrettyTables
```

For more information about packages management and environement, please refer to the Pkg.jl documentation. After installing the required packages, please remember to import them:

```
julia> using DelimitedFiles, MAT, Arpack, LinearAlgebra
julia> using Random, SparseArrays, Statistics
julia> using Metis, Graphs, SGtSNEpi, GLMakie
julia> using Colors, CairoMakie, PrettyTables
```

The file `add_paths.jl` contains everything which is needed to run the provided code. When you start working on the mini-project, please include it as:

```
include("path/to/add_paths.jl");
```

and you are ready to go. Remember to include this file each time you make updates to the provided files (e.g., `part_spectral.jl` or `part_inertial.jl`) and you want to re-run `bench_bisection.jl`.

1. Implement various graph partitioning algorithms [50 points]

- Open the Julia file `bench_bisection.jl` and familiarize yourself with the code.
- Implement **spectral graph bisection** based on the entries of the Fiedler eigenvector. Use the incomplete Julia file `part_spectral.jl` for your solution. Justify the threshold you chose.
- Implement **inertial graph bisection**. For a graph with 2D coordinates, this inertial bisection constructs a line such that half the nodes are on one side of the line, and half are on the other. Use the incomplete Julia file `part_inertial.jl` for your solution.
- Report the bisection edgecut for all toy meshes that are loaded in the script `bench_bisection.jl`. Use Table 1 to report these results. Comment on your results.

Table 1: Bisection results

Mesh	Coordinate	Metis 5.0.2	Spectral	Inertial
grid(12, 100)	12			
grid(100, 12)	12			
grid(100, 12, pi/4)				
gridt(50)				
gridt(40)				
Smallmesh				
Tapir				
Eppstein				

2. Recursively bisecting meshes [20 points]

We will now partition 2D graphs emerging from structural engineering matrices of NASA, available in the SuiteSparse Matrix Collection (SSMC) [3]. A robust algorithm that allows us to create a 2^l partition, with l being an integer, is presented in Algorithm 3. It uses the recursive function `Recursion` that takes as inputs the part C' of the graph to partition, the number of parts p' into which we will partition C', and the index (an integer) of the first part of the partition of C' into the final partitioning result $\mathcal{G}_p$. In practize, only strongly balanced bisection routines are utilized within this algorithm [1].

Algorithm 3 Recursive bisection.

Require: $\mathcal{G}(V, E)$,
Ensure: A p-way partition of $\mathcal{G}$
1: $\mathcal{G}_p = \{C_1, \ldots, C_p\}$
2: $p = 2^l$ ▷ p is a power of 2
3: **function** RECURSIVEBISECTION(graph $\mathcal{G}(V, E)$, number of parts p)
4: **function** RECURSION(C', p', index)
5: **if** p' is even **then** ▷ If p' is even, a bisection is possible
6: $p' \leftarrow \frac{p'}{2}$
7: $(C'_1, C'_2) \leftarrow$ `bisection`(C')
8: RECURSION(C'_1, p',index)
9: RECURSION(C'_2, p',index+k')
10: **else** ▷ No more bisection possible, C' is in a partition of $\mathcal{G}$
11: $C_{index} \leftarrow C'$
12: **end if**
13: **end function**
14: RECURSION$(C, p, 1)$
15: **return** $\mathcal{G}_p$ ▷ $\mathcal{G}_p$ is a partition of $\mathcal{G}$ in $p = 2^l$ parts
16: **end function**

This algorithm is implemented in the file `recursive_bisection.jl` of the toolbox. Utilize the script `bench_recusrive` to recursively bisect the finite element meshes loaded in 8 and 16 subgraphs. Use your inertial and spectral partitioning implementations, as well as the coordinate partitioning and the METIS bisection routine. Summarize your results in 2 and comment about these results. Finally, visualize the results for $p = 16$ for the case "crack". An example for spectral recursive partitioning is illustrated in Figure 3.

Table 2: Edge-cut results for recursive bi-partitioning.

Case	Spectral	Metis 5.1.0	Coordinate	Inertial
mesh3e1				
airfoil1				
3elt				
barth4				
crack				

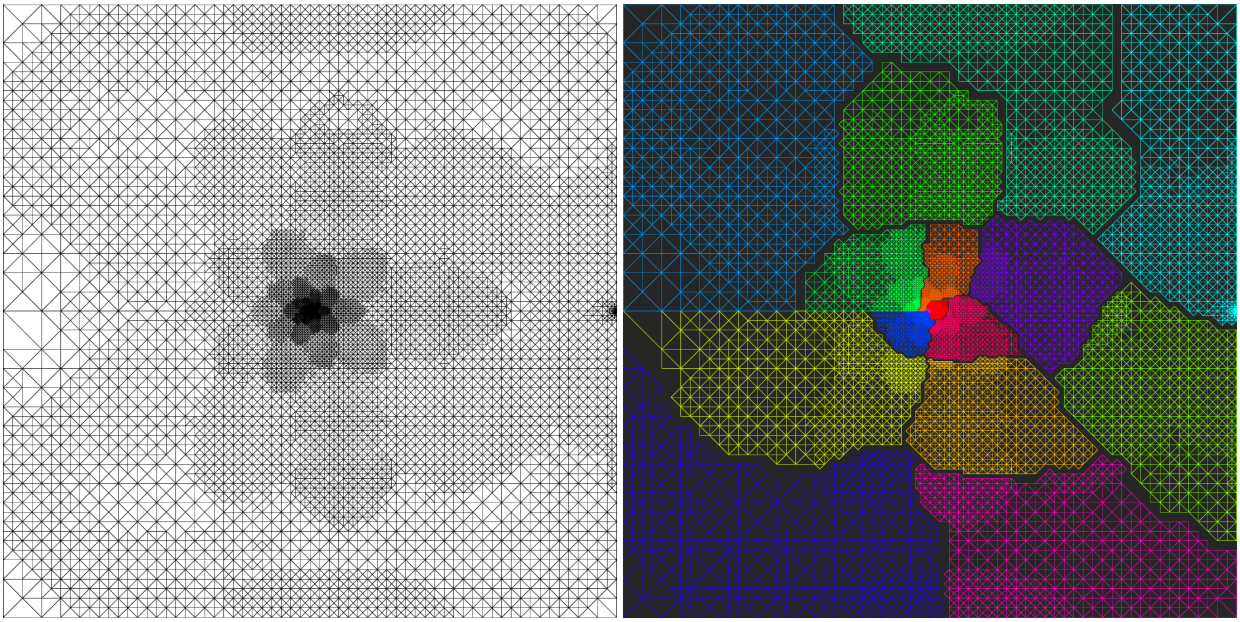


Figure 3: Left: The finite element mesh "crack" with 10240 nodes and 30380 edges. Right: 16-way recursive bisection of the mesh using Metis 5.1.0. Number of cut edges: 1186.

3. Comparing recursive bisection to direct k-way partitioning [15 points]

Recursive bisection is highly dependent on the decisions made during the early stages of the process, and also suffers from the lack of global information. Thus, it may result in suboptimal partitions [8]. This necessitated the development of robust methods for direct k-way partitioning.

Besides recursive bi-partitioning, METIS also employs a multilevel k-way partitioning algorithm. The graph $\mathcal{G} = (V, E)$ is initially coarsened down to a small number of vertices, a k-way partitioning of this much smaller graph is computed, and then this partitioning is projected back towards the original finer graph by successively refining the partitioning at each intermediate level [6]. We will compare the quality of the cut resulting from the application of recursive bipartitioning and direct multiway partitioning as implemented in Metis 5.1.0. Our test cases will be the commanche_dual and the skirt example.

Use the incomplete file `bench_metis.jl` for your implementation. Compare the cut obtained from Metis 5.1.0 after applying recursive bisection and direct multiway partitioning for the graphs in question. Consult the Metis manual to familiarize yourself with the way the Metis recursive and direct multiway partitioning functionalities should be invoked. Summarize your results in Table 3 for 16 and 32 partitions. Comment on your results.

Table 3: Comparing the number of cut edges for recursive bisection and direct multiway partitioning in Metis 5.1.0.

Partitions	Helicopter	Skirt
16-recursive bisection		
16-way direct bisection		
32-recursive bisection		
32-way direct bisection		

Quality of the code and of the report [15 Points]

The highest possible score of each project is 85 points and up to 15 additional points can be awarded based on the quality of your report and code (maximum possible grade: 100 points). Your report should be a coherent document, structured according to the template provided on iCorsi. If there are theoretical questions, provide a complete and detailed answer. All figures must have a caption and must be of sufficient quality (include them either as .eps or .pdf). If you made a particular choice in your implementation that might be out of the ordinary, clearly explain it in the report. The code you submit must be self-contained and executable, and must include the set-up for all the results that you obtained and listed in your report. It has to be readable and, if particularly complicated, well-commented.

Additional notes and submission details

Summarize your results and experiments for all exercises by writing an extended LaTeX report, by using the template provided on iCorsi (`https://www.icorsi.ch/course/view.php?id=14666`). Submit your gzipped archive file (tar, zip, etc.) – named `project_number_lastname_firstname.zip/tgz` – **on iCorsi** (see *[NC 2022] Project 3 - Submission Graph Partitioning*) **before the deadline**. Submission by email or through any other channel will not be considered. Late submissions will not be graded and will result in a score of 0 points for that project. You are allowed to discuss all questions with anyone you like, but: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently. Please remember that plagiarism will result in a harsh penalization for all involved parties.

In-class assistance

If you experience difficulties in solving the problems above, please join us in class either on Tuesdays (16:30-18:15) or on Wednesdays (14:30-16:15) in room C1.03.

References

[1] C.E. Bichot and P. Siarry. *Graph Partitioning*. ISTE. Wiley, 2013.

[2] Aydin Buluc, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. *Recent Advances in Graph Partitioning*, pages 117–158. Springer International Publishing, Cham, 2016.

[3] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[4] U. Elsner. Graph Partitioning: A Survey. Technical report, Technische Universität Chemnitz, Germany, 97-27, 1997.

[5] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, Piscataway, NJ, USA, June 1982. IEEE Press.

[6] George Karypis and Vipin Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing '96, page 35–es, USA, 1996. IEEE Computer Society.

[7] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, Feb 1970.

[8] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, September 1997.