Dr. R. Käppeli,
Dr. A. Schell, M. R. Herde,
C. Humbel, A. Mazzoleni,
N. A. Siegenheim

# Project 6

**Due date**: Monday 27 May 2024, 23:59 (midnight).

In this project, we will look at the graph partition problem in context of domain decomposition and popular HPC software libraries and tools widely used in computational science and engineering.
You may do this project in groups of students (max four or five). In fact, we prefer that you do so.

## 1 Graph Partitioning with Julia: Load balancing for HPC [50 points]

Partitioning a computational problem into sub-problems that can be solved efficiently in parallel is a recurring task in HPC. In order to be efficient, the sub-problems should be of near equal size, to ensure load balancing, and the required communication between the sub-problems should be minimized. We have already encountered this problem in a few instances during the course, mainly in the form of domain decomposition. However, the domains were mostly simple and rather square. In this sub-project, we consider more complex grids or meshes. We abstract domain decomposition as a graph partition problem.

The purpose of this sub-project is to familiarize yourself with some elementary graph partitioning algorithms and to implement them in the Julia language. We compare the resulting partitions with the METIS[1] graph partitioning software [5, 6, 7]. METIS was developed by Karypis and Kumar and is probably the most widely used graph partitioning software. 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.

We refer to Elsner [3] (see the course web page) for a comprehensive introduction to graph partitioning. For (much) more on graph partitioning, we refer to Bichot and Siarry [1]. Many other practical applications and the recent advances in the field of graph partitioning can be found in Buluç et al. [2] and references therein.

### 1.1 Graph partitioning: Generalities

Consider the mesh with 10 cells in Fig. 1a. Assuming that for the computation only cells sharing an edge have to exchange data[2], this mesh can be represented by the dependency graph shown in Fig. 1b. To decompose the mesh in two parts suitable for parallel processing, one tries to partition the mesh into two sub-meshes with the same number of cells and with a minimum number of edges between. This is equivalent to partition the graph in Fig. 1b into two complementary sub-graphs with equal number of vertices and a minimum number of edges between the two sub-graphs. In other words, we partition the graph in two equal parts by cutting the least possible number of edges.

For the simple example in Fig. 1, the solution is obvious. However, the solution is less obvious in general for larger meshes. In the following, we formalize the graph partitioning problem in a suitable manner for the scope of this sub-project and present three bisection algorithms, where bisection restricts the problem to the partition into two sub-graphs. Partitioning into a power of two partitions $p = 2^l$ can then be achieved recursively.

Let $\mathcal{G} = (V, E)$ be an undirected graph with $n$ vertices[3] $V = \{v_1, \dots, v_n\}$ connected by edges $E = \{e_{i,j} \mid \text{edge between } v_i \text{ and } v_j\}$. The goal is to bisect the graph $\mathcal{G}$ into two complementary sub-graphs

[1] `https://github.com/KarypisLab/METIS`
[2] This is a common situation in finite volume or discontinuous Galerkin methods.
[3] Depending on the context, vertices may also be called nodes or points.

$\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$, that is $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$, of near equal size, $|V_1| \approx |V_2|$, such that the number of edges between the parts,

$$\mathrm{cut}(V_1, V_2) = |\{e_{i,j} \in E \mid v_i \in V_1 \text{ and } v_j \in V_2\}|\,, \tag{1}$$

is minimized. The latter quantity is also known as the cut-size, edge-cut or cost of the partition.

We represent the dependency graph with the so-called adjacency matrix $A = (a_{ij})_{1\le i,j\le n}$ whose elements are defined by

$$a_{ij} = \begin{cases} 1 & \text{there is an edge } e_{i,j} \text{ between } v_i \text{ and } v_j, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Note that for an undirected graph the adjacency matrix is symmetric. The so-called degree of a vertex $\deg(v_i) = \sum_{j=1}^{n} a_{ij}$ is the number of edges that are connected to the vertex $v_i$. This is encoded in the degree matrix $D = \mathrm{diag}(d_1, \dots, d_n)$. With the adjacency and degree matrix, we can form the Laplacian matrix

$$L = D - A. \tag{3}$$

The latter will play an essential role in the spectral bisection algorithm presented below.

As an example, let us consider again the mesh and corresponding dependency graph in Fig. 1. The adjacency and degree matrices are given by

$$A = \begin{pmatrix} 0&1&0&0&0&0&0&0&0&0\\ 1&0&1&0&0&0&0&1&0&0\\ 0&1&0&1&0&0&0&0&0&0\\ 0&0&1&0&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0\\ 0&0&0&1&0&0&1&0&1&0\\ 0&0&0&0&0&1&0&1&0&1\\ 0&1&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&1&0&0&0 \end{pmatrix}, \quad D = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 0&3&0&0&0&0&0&0&0&0\\ 0&0&2&0&0&0&0&0&0&0\\ 0&0&0&3&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&3&0&0&0&0\\ 0&0&0&0&0&0&3&0&0&0\\ 0&0&0&0&0&0&0&2&0&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

and the Laplacian matrix

$$L = D - A = \begin{pmatrix} 1&-1&0&0&0&0&0&0&0&0\\ -1&3&-1&0&0&0&0&-1&0&0\\ 0&-1&2&-1&0&0&0&0&0&0\\ 0&0&-1&3&-1&-1&0&0&0&0\\ 0&0&0&-1&1&0&0&0&0&0\\ 0&0&0&-1&0&3&-1&0&-1&0\\ 0&0&0&0&0&-1&3&-1&0&-1\\ 0&-1&0&0&0&0&-1&2&0&0\\ 0&0&0&0&0&-1&0&0&1&0\\ 0&0&0&0&0&0&-1&0&0&1 \end{pmatrix}.$$

This can easily be formalized to an arbitrary number of partitions $p$ and weighted vertices and edges. See Elsner [3] for details.

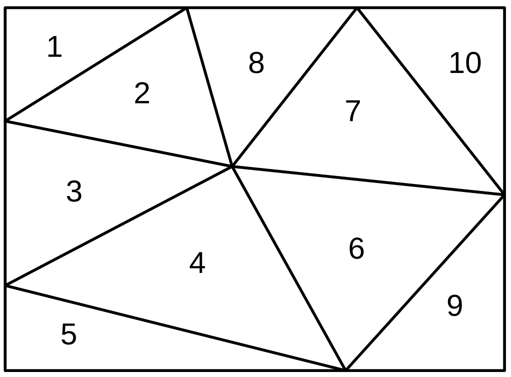
## 1.2 Graph partitioning: Simple algorithms

### 1.2.1 Partitioning with geometric information: Coordinate and inertial bisection

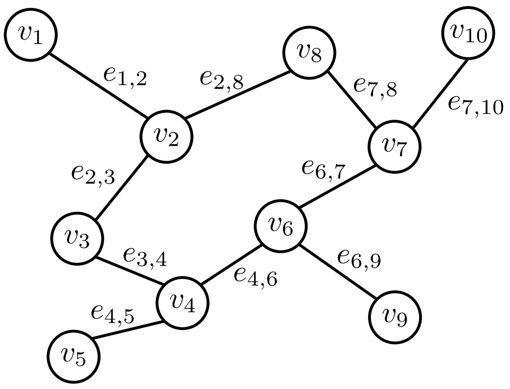
We start with two bisection algorithms that assume that the geometric layout of the graph is known. This is for instance the case in our application of domain decomposition of a given mesh.

#### Coordinate bisection

Coordinate bisection simply consists of finding the hyperplane orthogonal to a coordinate axis that divides the graph vertices in two (almost) equal parts with the smallest edge-cut. This is achieved by computing the median $\overline{x}_i$ over each coordinate axis $x_i$ (e.i., $x_1 \equiv x$ and $x_2 \equiv y$), that is $\overline{x}_i$ separates all vertices of the graph in two with half having $x_i$ coordinates less and half larger than $\overline{x}_i$. Then one computes the edge-cut for each of the coordinate axes and bisects along the one with the smallest edge-cut. This algorithm is summarized in Algorithm 1 for two dimensions.



(a)



(b)

Figure 1: Mesh with 10 (simplex/triangle) cells (left) and corresponding dependency graph (right).

Coordinate bisection is a very simple bisection algorithm. However, the quality of the resulting partition may depend strongly on the coordinate systems (e.g., a simple coordinate axes rotation may lead to very different results). We refer to Elsner [3] for further information.

**Algorithm 1** Coordinate bisection.

```
Require: $\mathcal{G}(V,E), P_i = (x_i, y_i)$, $i = 1, \dots, n$ the coordinates of the vertices
Ensure: A bisection of $\mathcal{G}$
1: function InertialBisection(graph $\mathcal{G}(V,E), P_i$)
2:     For each coordinate axis, $x$ and $y$, find the median value $\overline{x}$ and $\overline{y}$ that divides the vertices in two equal parts.
3:     Partition the vertices of the graph around median coordinate line having the smallest edge-cut.
4:     return $V_1, V_2$                                   ▷ bisection of $\mathcal{G}$
5: end function
```

### Inertial bisection

Inertial bisection improves upon the axes dependence of coordinate bisection by choosing the dividing hyperplane orthogonal along a direction that runs through the center of mass of the vertices. In two dimensions, such a line $L$ is chosen such that the sum of squares of the distance of the vertices to the line is minimized. It is defined by a point $\overline{P} = (\overline{x}, \overline{y})$ and a vector $\mathbf{u} = [u_1, u_2]^T$ with $\|\mathbf{u}\|_2 = \sqrt{u_1^2 + u_2^2}$ such that $L = \{\overline{P} + \alpha \mathbf{u} \,|\, \alpha \in \mathbb{R}\}$ (see Fig. 2). We set

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \overline{y} = \frac{1}{n}\sum_{j=1}^{n} y_j, \tag{4}$$

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

$$\begin{aligned}\sum_{i=1}^{n} d_i^2 &= \sum_{i=1}^{n} (x_i - \overline{x})^2 + (y_i - \overline{y})^2 - (u_1(x_i - \overline{x}) + u_2(y_i - \overline{y}))^2 \\ &= u_2^2 \sum_{i=1}^{n} (x_i - \overline{x})^2 + u_1^2 \sum_{i=1}^{n} (y_i - \overline{y})^2 + 2u_2 u_1 \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y}) \\ &= \mathbf{u}^T \begin{bmatrix} S_{yy} & S_{xy} \\ S_{xy} & S_{xx} \end{bmatrix} \mathbf{u} = \mathbf{u}^T M \mathbf{u}, \end{aligned} \tag{5}$$

where $S_{xx}, S_{xy}, S_{yy}$ are the sums as defined in the previous line. The resulting matrix $M$ is symmetric, thus the minimum of Eq. (5) 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 Elsner [3] and references therein for a thorough overview of the inertial bisection algorithm.

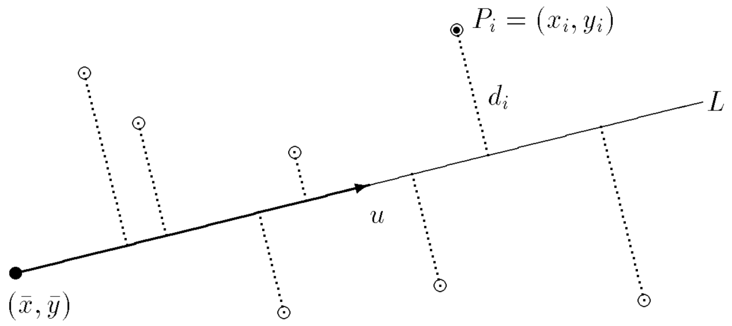


Figure 2: Illustration of inertial bisection in 2D (from Elsner [3]).

**Algorithm 2** Inertial bisection.

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

### 1.2.2 Partitioning without geometric information: Spectral bisection

Spectral bisection was initially considered the standard for solving graph partitioning problems. Unlike the previously introduced bisection algorithms, it does not use any geometric information associated with the graph to partition. A bisection is computed using the eigenvector associated with the second smallest eigenvalue of the Laplacian matrix $L$ of the graph. The graph Laplacian $L$ is a symmetric positive semi-definite matrix[4], with its smallest eigenvalue being $\lambda^{(1)} = 0$, and the associated eigenvector $\mathbf{u}^{(1)} = c\mathbf{1}$ being the constant one. The eigenvector $\mathbf{u}^{(2)}$ associated with the second smallest eigenvalue $\lambda^{(2)}$ 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 $\mathbf{u}^{(2)}$. Thresholding the values of $\mathbf{u}^{(2)}$ around 0 results in two roughly balanced (equal sized) partitions, with minimum edge-cut, while thresholding around the median value of $\mathbf{u}^{(2)}$ produces two strictly balanced partitions. The procedure to compute a bisection using spectral partitioning is summarized in Algorithm 3. For a much more detailed description of the algorithm, we again refer to Elsner [3].

**Algorithm 3** 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 $\lambda^{(2)}$ and its associated eigenvector $\mathbf{u}^{(2)}$.
4:     Set 0 or the median of all components of $\mathbf{u}^{(2)}$ as threshold $\epsilon$.
5:     Choose$V_1 := \{v_i \in V | u_i < \epsilon\}$, $V_2 := \{\{v_i \in V | u_i \geq \epsilon\}$.
6:     return $V_1, V_2$                                                          ▷ bisection of $\mathcal{G}$
7: end function
```

## 1.3 Graph partitioning: Recursive bisection

A simple and robust algorithm to create a $p = 2^l$ partition, with $l$ being an integer, is recursive bisection and it is presented in Algorithm 4. 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 $\mathcal{G}_p$.

[4]According to the spectral theorem of linear algebra the Laplacian matrix $L$ therefore possesses an orthogonal basis of eigenvectors $\mathbf{u}^{(i)}$ and corresponding real eigenvalues $\lambda^{(1)}$.

**Algorithm 4** 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+$p'$)
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
```

## 1.4 Graph partitioning with Julia

The goal of this sub-project is to familiarize yourself with various algorithms for graph partitioning applied to domain decomposition. We have chosen the Julia language as our tool due to its high-level, dynamic capabilities, which are increasingly popular in HPC for some specific tasks. For more information about the Julia programming language, including package management and environment setup, please refer to the Julia documentation.

### 1.4.1 Environment setup

The following commands log in to Euler and set up the Julia environment:

```
[user@local]$ ssh -Y euler.ethz.ch
[user@eu-login-43]$ module load gcc julia eth_proxy

The following have been reloaded with a version change:
  1) gcc/4.8.5 => gcc/11.4.0

[user@eu-login-43]$ export
↪  JULIA_DEPOT_PATH="$HOME/.julia:/cluster/work/math/hpclab/julia:$JULIA_DEPOT_PATH"
[user@eu-login-43]$ srun --time=4:00:00 --mem-per-cpu=8G --x11 --pty bash
srun: job 58201149 queued and waiting for resources
srun: job 58201149 has been allocated resources
[user@eu-a2p-284]$ julia
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.10.2 (2024-03-01)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

julia> 1 + 1
2

julia>
```

The first command enables X11 forwarding, the second loads the necessary modules, the third sets the Julia depot path (where we have already installed the necessary packages), the fourth submits an interactive job to the queue (1 CPU, 8 GB of memory and X11 forwarding enabled for 4 hours) and the final command launches a Julia interactive session (often referred to as "REPL" for Read-Eval-Print-Loop). For the duration of the project, we recommend adding the second and third commands to your `.bashrc` configuration script (located in `$HOME/.bashrc`).

Before starting to work on the sub-project tasks, you need to activate the provided project package. Go into the directory `graph_part` (containing the `Project.toml` and `Manifest.toml`). In there, start an interactive Julia session and enter the package manager `Pkg` by pressing the `]` key to activate the package:

```
julia> ]
(@v1.10) pkg> activate .
```

Once the environment set up, you may run the `main.jl` script in the REPL

```
julia> include("main.jl")
```

This will create several PDFs of the mesh/graph in `airfoil1.mat` displayed in Fig. 3 and report the partitions' edge-cut[5]. You can display the PDFs with

```
[user@eu-a2p-284]$ display airfoil1.pdf # airfoil1_coordinate.pdf
                                        # or airfoil1_metis.pdf
```

To run code in a file non-interactively, you can

```
[user@eu-a2p-284]$ julia --project=. ./main.jl
```

in the provided Julia project directory.

Alternatively, you can also install Julia and the packages on your own machine. First, you need to install Julia. You can either use an appropriate package manager, pre-compiled binaries or compile directly from source. The last option is probably more involved. Please follow the instructions available at https://julialang.org/downloads/. Once Julia is installed, launch an interactive Julia REPL session and either activate and instantiate the project environment[6]

```
julia> ]
(@v1.10) pkg> activate "/path/to/julia/project/"
(@v1.10) pkg> instantiate
```

or manually install the packages into your `MyOwnProject` with the Julia package manager

```
julia> ]
(@v1.10) pkg> activate "MyOwnProject"
(@v1.10) pkg> add Arpack CairoMakie Colors Graphs LinearAlgebra MAT Metis
↪   PrettyTables Random SGtSNEpi SparseArrays Statistics
```

### 1.4.2 The sub-project tasks

Complete the following tasks

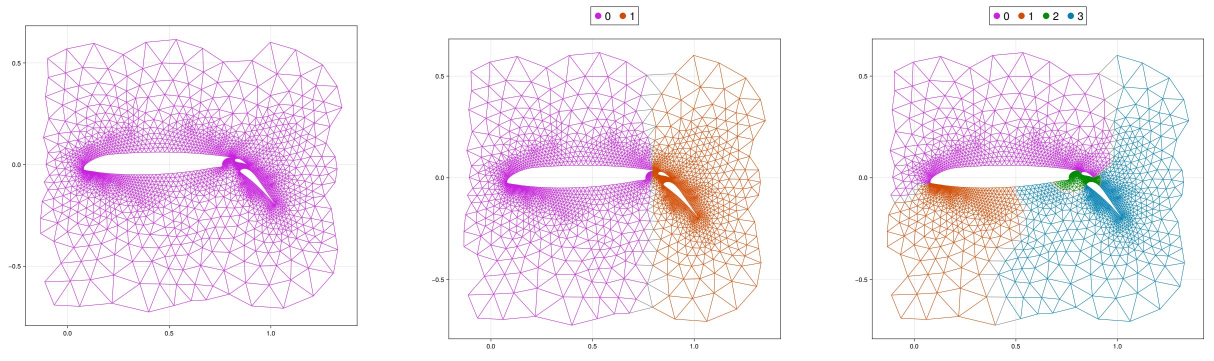
(1) Familiarize yourself with the provided skeleton code.

(2) Implement graph partitioning algorithms **[30 points]**.

- Implement the inertial bisection algorithm. Use the provided skeleton code `part_inertial.jl`. **Hint**: In the skeleton code, we provide an implementation of coordinate bisection.
- Implement the spectral bisection algorithm. Use the provided skeleton code `part_spectral.jl`.

[5] You may also see a warning from the `Makie.jl` package. This is unfortunate, but can be safely ignored.

[6] This might give a warning if your installed Julia version is different from the one on Euler. One solution is to manually install the packages or to use the Julia installer and version multiplexer `juliaup` (https://github.com/JuliaLang/juliaup).



(a) (b) (c)

Figure 3: The mesh of an airfoil: original (left), partitioned in two with coordinate bisection (middle) and four with METIS (right).

- Report the bisection edge-cut for all toy meshes that are loaded in the function `bench_bisection.jl` by running the script `main_bench.jl`. Qualitatively discuss your results briefly. Do you observe any surprising, or sub-optimal, partitions? For illustration, please also include figures of the partitions (i.e., similar to Fig. 3).

For the eigenvalue computations, use the `eigs` function from the `Arpack` package. Please visit Arpack.jl[7] for more information.

(3) Implement the recursive bisection algorithm **[20 points]**.

- Implement the recursive bisection algorithm in `recursive_bisection.jl`.
- Report the bisection edge-cut for all toy meshes that are loaded in the function `bench_recursive.jl` by running the script `main_bench.jl`. Qualitatively discuss your results briefly. Do you observe any surprising, or sub-optimal, partitions? For illustration, please also include figures of the partitions (i.e., similar to Fig. 3).

## 2 HPC software frameworks [50 points]

This sub-project will be uploaded within the week and you will be notified by email.

## Additional notes and submission details

Submit **all the source code files** together with your used build files (e.g., `Makefile`(s)) and other scripts (e.g., batch job scripts) in an archive file (tar or zip) and summarize your results and observations for all sections by writing a detailed LaTeX report. Use the LaTeX template from the webpage and upload the report as a PDF to Moodle.

- Your submission should be a tar or zip archive, formatted like `project_number_lastname_firstname.zip/tgz`. It must contain:
  - All the source codes of your solutions.
  - Build files and scripts. If you have modified the provided build files or scripts, make sure they still build the sources an run correctly. We will use them to grade your submission.
  - `project_number_lastname_firstname.pdf`, your report with your name.
  - Follow the provided guidelines for the report.
- Submit your archive file through Moodle.

Please follow these instructions and naming conventions. Failure to comply results in additional work for the TAs, which makes the TAs sad...

[7] https://arpack.julialinearalgebra.org/latest/

## References

[1] Charles-Edmond Bichot and Patrick Siarry, editors. *Graph Partitioning.* Wiley, feb 2013. doi: 10.1002/9781118601181.

[2] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent Advances in Graph Partitioning. Lecture Notes in Computer Science, pages 117–158. Springer International Publishing, Cham, 2016. ISBN 9783319494876. doi: 10.1007/978-3-319-49487-6_4. URL https://doi.org/10.1007/978-3-319-49487-6_4.

[3] Ulrich Elsner. Graph partitioning - a survey. Technical report, Technische Universität Chemnitz, September 2005. URL https://nbn-resolving.org/urn:nbn:de:swb:ch1-200501047.

[4] G. Karypis and V. Kumar. Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs. In *Supercomputing '96:Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, pages 35–35, January 1996. doi: 10.1109/SUPERC.1996.183537.

[5] G. Karypis and V. Kumar. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, pages 28–28, November 1998. doi: 10.1109/SC.1998.10018. URL https://ieeexplore.ieee.org/document/1437315.

[6] George Karypis and Vipin Kumar. Multilevel$k$-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, January 1998. ISSN 0743-7315. doi: 10.1006/jpdc.1997.1404. URL https://www.sciencedirect.com/science/article/pii/S0743731597914040.

[7] George Karypis and Vipin Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, January 1998. ISSN 1064-8275. doi: 10.1137/S1064827595287997. URL https://epubs.siam.org/doi/abs/10.1137/S1064827595287997.