# Appendix

alessio pellegrino (alessio.pellegrino@studio.unibo.it)

July 20, 2024

In this document, I will present some extra graphs as well as additional descriptions of the project, including where it can be found and how it can be executed. These graphs offer similar conclusions to those shown in the main report but add some extra points to the discussion.

**Project Structure and Execution** The project, which can also be found on GitHub[1], is composed of five main folders:

- **cuda**: Contains the CUDA implementation.
- **omp**: Contains the OMP implementation.
- **scripts**: Contains some Python scripts.
- **graphs**: Contains the graphs used for this report.
- **imgs**: Contains the images used for this report.

Both the CUDA and OMP code can be compiled by entering the corresponding folder and using the command *Make*. This will create a new folder called "output" that will contain the compiled executable. The executable can be used by calling it from the command line with the desired graph file as the input parameter.

Each graph file has the same structure: a CSV formatted file with the nodes of the graph in the first line, followed by the adjacency matrix representing the edges. In this case, the value 0 is used as a dummy value to denote the absence of an edge.

The program's output is a JSON-formatted string with properties for the node implementation ("nodes"), the edge implementation ("edges"), the input file name ("input_file"), and the number of used cores ("cores")[2]. Both the "nodes" and "edges" properties are JSON objects themselves with the following sub-properties: (i) "distances": the distance output of the algorithm, (ii) "predecessors": the predecessor output of the algorithm, (iii) "negative_cycles": the presence or absence of a negative cycle, (iv) "inf_time": time needed to compute

[1] `https://github.com/SeppiaBrilla/bellman-ford`
[2] OMP version only

the infinite value, (v) "init_time": time needed to compute the array initialization, (vi) "relaxation_time": time needed to compute the relaxation procedure, (vii) "negative_cycles_time": time needed to check for the presence of negative cycles.

In the scripts folder, there are two scripts: one used to check the correctness of a solution for each implementation, and one used to generate graphs, as well as a notebook script used to generate the graphs in this report.

The files "cuda_res_local.json", "cuda_res_slurm.json", "omp_res_local.json", and "omp_res_slurm.json" contain the results presented in this report and can be replicated by running the "run.sh" script. The run.sh script compiles the project and runs the two Python scripts: "run_cuda.py", "run_omp.py" and "run_weak.py".

**Local Results** The biggest difference between the Slurm and the local results is that, in the local version, the node implementation is faster, while the opposite is true for the Slurm runs. This may be caused by a large number of factors, including different hardware specifications or the driver version used.

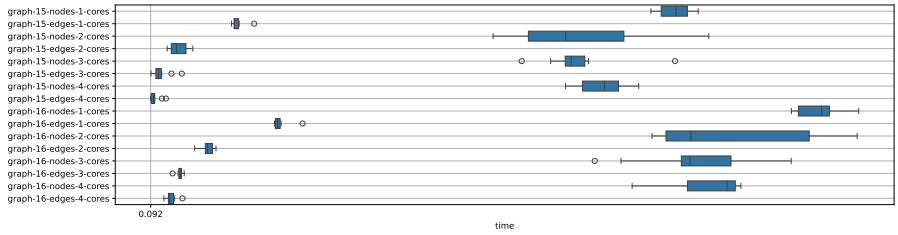


Figure 1: Variation of the execution time for a given combination of input, version, and number of cores (local version).

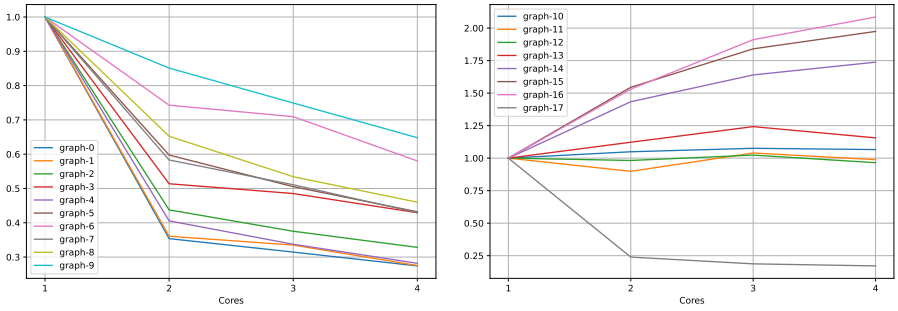


Figure 2: Speedup on the edge version (local version).

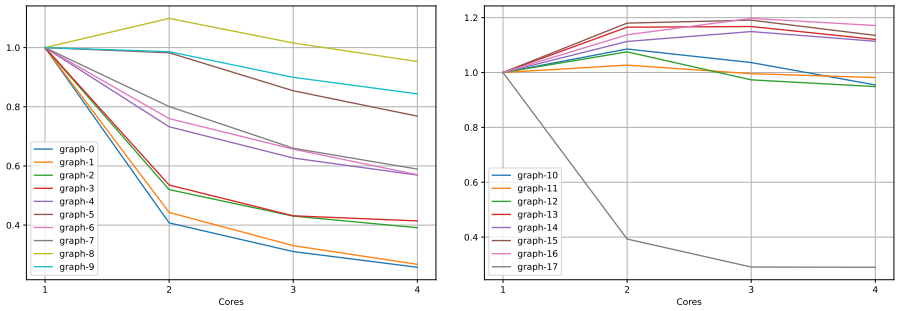


Figure 3: Speedup on the node version (local version).

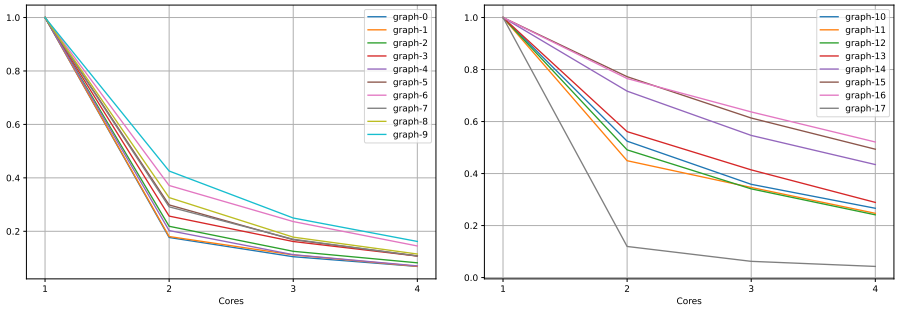


Figure 4: Strong efficiency on the edge version (local version).

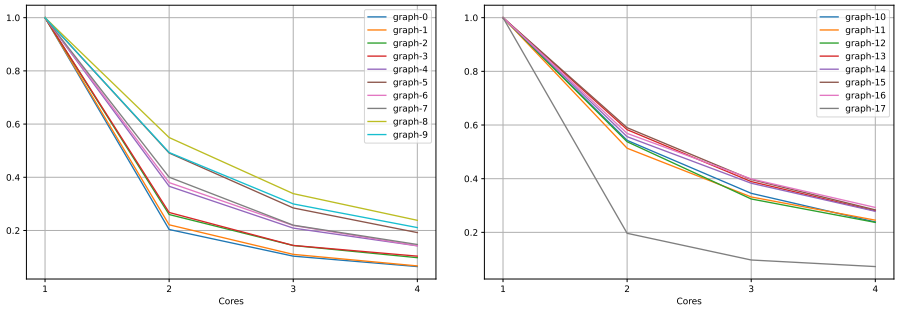


Figure 5: Strong efficiency on the node version (local version).

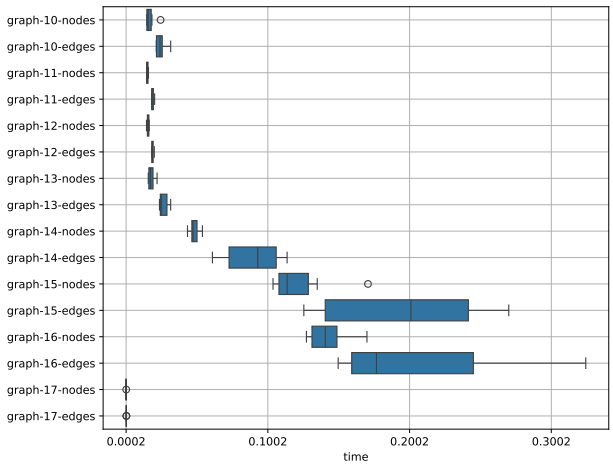


Figure 6: Variation of the execution time for a given combination of input and version on a CUDA device (local version).

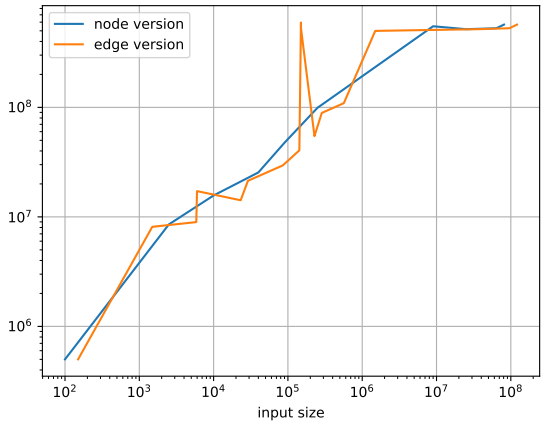


Figure 7: Throughput of the two different versions of the Bellman-Ford algorithm (local version).

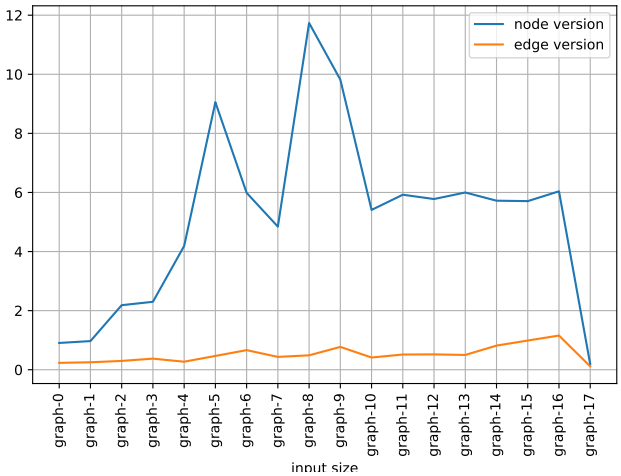


Figure 8: Speedup of the two versions of the algorithm compared to the sequential version on the CPU (local version).

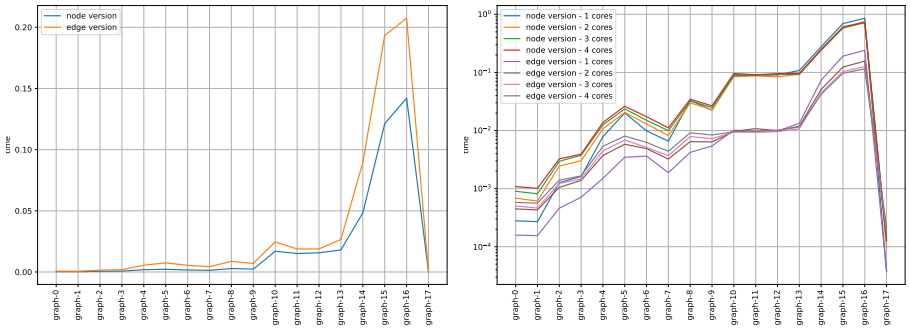


Figure 9: Times on CUDA (left) and OMP (right) (local version).

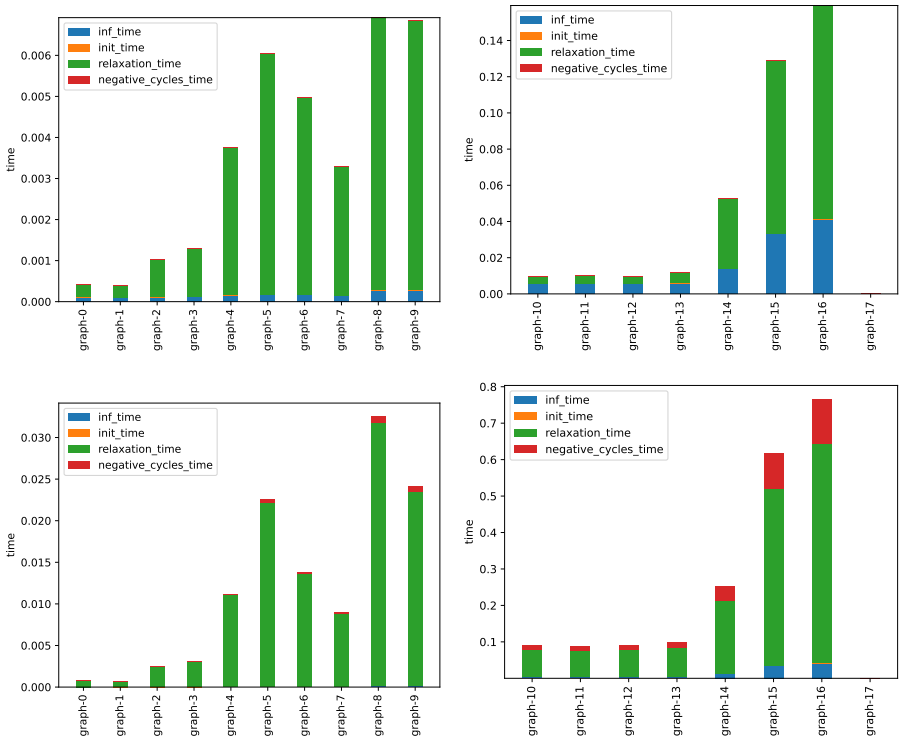


Figure 10: Average time taken by each section of the algorithm on the edge (top) and nodes (bottom) versions on OMP (local version).

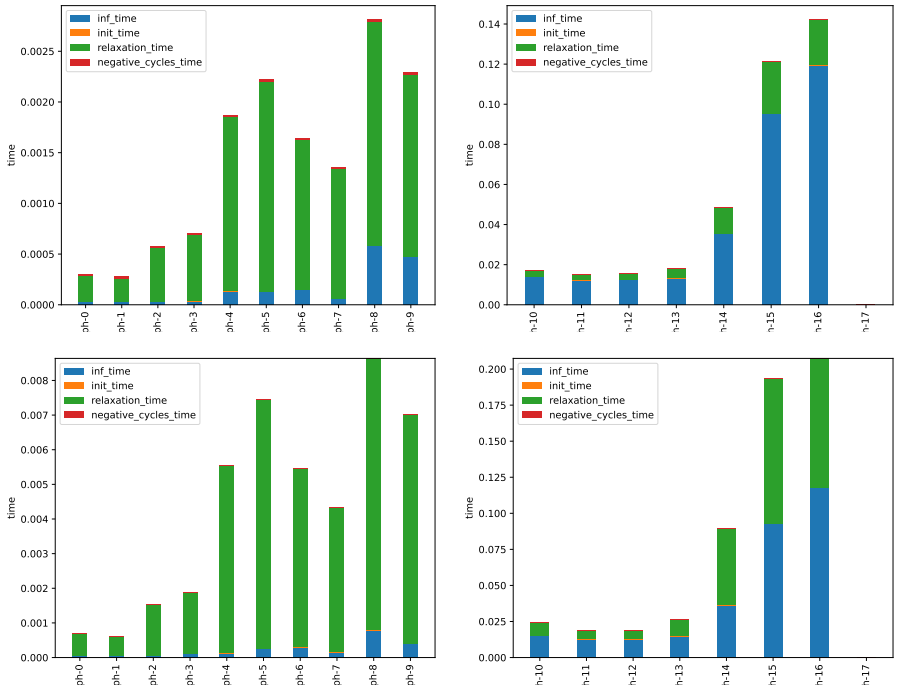


Figure 11: Average time taken by each section of the algorithm on the edge (top) and nodes (bottom) versions on CUDA (local version).