Final project
The Minimum Bisection Problem

Introduction

The task of effectively dividing the vertices within a given graph into subsets, ensuring that specific conditions are met, stands as a fundamental challenge in algorithmic studies. Beyond its clear theoretical significance, problems related to graph partitioning hold substantial practical relevance across various domains, including computer vision, image processing and VLSI layout design.

In particular, the issue of partitioning a graph into components of equal size, while minimizing the number of connections between these components, plays a pivotal role in parallel computing. To illustrate, in order to parallelize applications, it is typically necessary to evenly distribute the computational workload among processors, while simultaneously minimizing the inter-processor communication

Formally, let $G = (V, E)$ be an undirected simple graph such that $|V| = 2n$. (Note that G has an even number of vertices and it is not necessarily connected.) The Minimum Bisection Problem (MBP) consists in finding a partition of V, into two sets V_1 and V_2 such that $V = V_1 \cup V2$, $V_1 \cap V_2 = \emptyset$ and $|V_1| = |V_2| = n$, that minimises the number of edges x_1x_2 with $x_1 \in V_1$ and $x_2 \in V_2$.

The MBP is a well-known combinatorial optimization problem in graph theory and computer science and has a history dating back several decades. It was proved that the MBP is NP-hard, which means that it is computationally challenging to find an optimal solution in a reasonable amount of time for large graphs. This result underscored the problem's difficulty and led to the development of approximation algorithms. The MBP remains a fundamental problem in graph theory and combinatorial optimization, and its study has not only contributed to theoretical advancements but has also led to practical solutions for optimizing various real-world applications, particularly in the field of computer science and engineering.

Exercises

1. Describe real-life situations that can be modelled as MBP where at least one is an original idea of yours.
2. Develop and implement an exact algorithm to solve the MBP with at least one improvement as explained in class (not doing so will make you lose a considerable amount of points).
3. Develop and implement a constructive heuristic to solve the MBP.
4. Develop and implement a local search heuristic to solve the MBP.
5. Develop and implement an algorithm that uses the Tabu Search meta-heuristic [1] for the MBP. Consider the use of different sizes for the tabu list and analyse possible stop criteria. Optionally, you can add aspiration level, intensification and diversification techniques. **SET** all the parameters involved (size of the tabu list, stop criteria, etc.) through experiments. **JUSTIFY** your elections and the choice of the test instances considered to decide your parameters.
6. For each of the methods 2 to 5:
   - First explain in detail the ideas and how your algorithm works. You can use examples to illustrate your ideas. Then give its pseudo-code, **DO NOT** give the code!
   - Calculate its time complexity using the pseudo-code given before.
   - Try to describe instances of the MBP for which the method does not provide an optimal solution and explain how bad the obtained solution can be with respect to the optimal solution.
   - Run experiments that allow to observe the performance of the algorithm[1]. Compare the execution time with its theoretical time complexity. Also, compare the quality of the obtained

[1]To create test instances in graphs, you should vary its density. It is normally used a probability of having an edge or not, using three values: 25%, 50% and 75%. Finally for each density, you should vary the size n of the graph

solutions and the execution time with respect to the input size (and other parameters if appropriate). In case the algorithm has some configurable parameters that determine its behaviour (the meta-heuristic for example, although it remains open to others as well), you should run experiments varying the values of the parameters and choose, if possible, the settings that provide best results for the instances used. Present the results obtained using appropriate graphics/tables.

7. Once you have chosen the best configuration values for each heuristic, run experiments on a **new set of instances** to observe the performance of all the methods against each other, and to compare the quality of the solutions obtained and the execution time. Present all the results obtained by using appropriate graphics/tables and present conclusions about your work.

General guidelines

1. A report tackling all points described above should be written (in English, of course) and uploaded in pdf format. You should follow the guidelines described in the document "Normes de mise en page des dossiers", Version 3.3, by Matéo Sorin. Use of LaTeX [2] is **highly recommended**. For drawing graphs, you can use "yEd Graph Editor" software that is downloadable for free here: https://www.yworks.com/downloads#yEd. For drawing graphics, you can use "GNU Plot" software that is downloadable for free here: http://www.gnuplot.info/download.html.

2. You can use any programming language you prefer to implement your algorithms **BUT** use the same for all of them.

3. Your algorithms should read an input file (check the examples below) that contains two lines as follows:

   - The first line has the number n of vertices and m of edges separated by one space.
   - The second line contains a list of pairs of numbers separated by two spaces that represents the edges. Each pair of numbers u, v such that $0 \leq u \neq v \leq n-1$, represents two adjacent vertices and are separated by one space.

4. Your algorithms should write a file containing the solution (check the examples below) as follows:

   - The name of the file should be "`instance_method.out`" where "`instance`" is the name of the input file (without the extension if any) and "`method`"should be one among "`exact`", "`constructive`", "`local_search`" and "`tabu_search`".
   - The file should contain three lines. The first one contains the number vertices of the graph followed by the number of edges of the solution separated by one space. The second and third line contain the number of the vertices in each of the two sets of the solution separated by one space from each other.

5. You should handout as well the test instances you have used to set the configurable parameters and the final set of instances used to compare performances (as they must be referenced in the report).

6. **VERY IMPORTANT:** While you are running measured experiments,

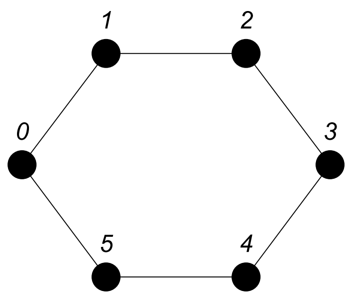
   - **DO NOT** do anything else with your computer **AND** close all unnecessary programs (like chrome, firefox, etc) that might be running. Not doing so might affect the performance of your algorithms and give inaccurate results. **DO NOT** plug/unplug your computer either.
   - **USE** the same computer (and under the same conditions specified above) to compare time execution of algorithms. It makes no sense (therefore it will give you wrong results), if you compare time execution, for example, of algorithm A against algorithm B, in two different computers with different processors, memory size, etc.
   - **DO NOT** waste your time running experiments on single instances for many hours (or days!).

Examples

- Consider the file "`test1.in`":

```
6 6
0 1  0 5  1 2  2 3  3 4  4 5
```

This file corresponds to the following graph on 6 vertices and 6 edges:



Therefore if we run the exact algorithm in that instance, the file "`test1_exact.out`" will be created containing the following three lines only.

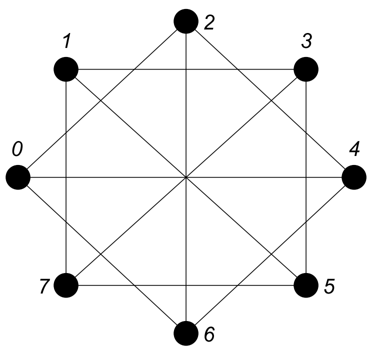
```
6 2
0 1 5
2 3 4
```

This corresponds to the sets $V_1 = \{0,1,5\}$ and $V_2 = \{2,3,4\}$ with 2 edges in-between, 12 and 45. Note that other solutions are also possible, for example, $V_1 = \{0,1,2\}$ and $V_2 = \{3,4,5\}$

- Consider now the file "`test2.in`":

```
8 12
0 2  0 4  0 6  2 4  2 6  4 6  1 3  1 5  1 7  3 5  3 7  5 7
```

This file corresponds to the following graph on 8 vertices and 12 edges:



Therefore if we run the exact algorithm in that instance, the file "`test2_exact.out`" will be created containing the following three lines.

```
8 0
0 2 4 6
1 3 5 7
```

This corresponds to the sets $V_1 = \{0, 2, 4, 6\}$ and $V_2 = \{1, 3, 5, 7\}$ with 0 edges in-between. Note that the input graph is not connected.

Deadline & Upload instructions

The deadline of the project is on Thursday, 21 December, 2023, 2pm, on the ENT.
NO FILES WILL BE ACCEPTED AFTER THE DEADLINE.

You should upload only **ONE** compressed file called "`team_X.zip`", where "`X`" is the number of your team. Only **ONE** member of the team should upload the file. This file should contain the following:

- A file called "`README.md`" that explains how the project folders/files are organised, how to compile/execute the source code, where the executable files will be created, where the input files should be located, where the output files are going to be written and all necessary instructions.
- A folder called "`report`" containing the pdf file called "`report_team_X.pdf`" of your report and its source files.
- A folder called "`src`" having inside the following sub-folders:
  i. Sub-folder "`model`": containing the source code of your graph model and basic functions shared by all algorithms.
  ii. One different sub-folder for each algorithm called "`method`", where "`method`" should be "`exact`", "`constructive`", "`local_search`" and "`tabu_search`". Each of these folders must contain the source code of each different algorithm.
- A folder called "`instances`": containing one different sub-folder for each of your algorithms called "`method`", where "`method`" should be "`exact`", "`constructive`", "`local_search`" and "`tabu_search`". Each of these folders must contain the instances you used to test your algorithms individually and/or to set configurable parameters (that should be referenced in the report). Also it must contain a folder called "`new_instances`" containing the **new set of instances** you used to compare performance between algorithms as described before. You should compress all instances files if their size is big.

If you use `github` or any other repository manager, **DELETE** all hidden files and folders that the manager creates. **Not doing so will lead to a penalty of points**.

References

[1] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[2] *An introduction to LaTeX*. LaTeX project (`https://www.latex-project.org/about/`).