Algorithms and Data Structures (MA316)
Sorbonne Université

Some project proposals

October 16, 2025

Instructions

Projects should be tackled in teams of at most two students (but different groups can work independently on the same subject). Reports should be typeset (ideally in Latex or Markdown). They must be accompanied with source code and build instructions, and should be uploaded on Moodle or Github upon completion in a single archive following the name scheme `name_surname-MA316-project.7z`. Deadline for upload is January 16th 2025. Appointments for project presentation (on an individual basis) should be taken between the beginning of December 2025 and January 16th 2025, e.g. by e-mail at `didier.smets@sorbonne-universite.fr` or `ani.miraci@sorbonne-universite.fr`.

In case of guidance needed, be it for the choice of a project or during its realization, contact us by e-mail or during TP/lecture classes. Contact us also if you have your own subject proposal.

In all cases, **you should write us at the latest on December 1st which subject you have chosen** and who (if applicable) is your team member.

1 Point in polygon

The goal of this project is write a function which can check whether a given input point in $\mathbb{R}^2$ belongs to (the interior of) a given polygon P . The data structures (in particular how to encode a polygon in the plane) and the exact prototype of the function are left open. You should first consider the case of simple polygons (i.e. non self-intersecting and with no holes), and then discuss the general case. For the latter, you should describe in the API in which precise sense the inclusion is understood. Note that there are different possible strategies (scan lines, winding number), you may test and discuss more than one. One typical use of such a function is object picking : the user clicks on screen (this yields the query point) and the code must return the object (be it button, widget, or more complicated as below) which got the click, in order to process it.

To test your code, download and adapt to your set-up (this may require a bit of data manipulation) a polygon describing the boundaries of a country of your choice (note that some of them are multi-polygons) in geographic coordinates.

2 Task scheduling

For this project we are assuming that we are given a number of different tasks that have dependencies between each other (in the sense that some tasks need to be finished before others can be started). The only guarantee that we have is that there are no circular dependencies (these would otherwise imply that not all tasks can be completed).

In a first step you should formalize this problem in terms of a graph data structure with certain properties, and then describe and code an algorithm that will provide one among the possible sequential tasks scheduling. Your guiding keyword is topological sorting of acyclic directed graph.

In a second step you will consider the case where each task also has a known duration, and the tasks can now be performed in parallel (with an unlimited number of workers) with the only condition that a task cannot be started before all of its prerequisites have finished. Describe and code an algorithm that will schedule them so that the total completion time is (as close as possible to being) minimal.

In a third step, you may consider the case of a finite (a priori given) number of workers.

3 R-trees and Point location

An R-tree is a data structure whose goal is to allow for efficient search of nearest neighbours (in a set of points of more generally of spatially extended objects). For this project, you are expected to

1. Read, understand and describe the notion of a R-tree. One advisable reference is in the original paper by Antonin Guttman (cfr e.g. `http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf`) where they were first introduced.

2. Search for open source code implementing them (or code it from scratch but that may be somewhat longer), and more importantly understand the code structure. The implementations (C or C++) from `http://superliminal.com/sources/sources.htm`, section R-Trees, are particularly recommended, you will encounter the widely used notion of *call-back function*.

3. As an application, write a function which, given a 2D mesh (as the ones we studied in some tutorials), builds an R-tree containing the bounding boxes of all of its triangles.

4. Based on the previous, write a function which, given a 2D mesh and an arbitrary input point, finds the triangle in the mesh (assuming e.g. there is one and only one) that contains it. Compare its efficiency against the naive implementation that would check sequentially all triangles for point inclusion.

4 Meshes and motion by mean curvature

The motion by mean curvature of a surface is an evolution flow of that surface for which the normal velocity vector of each point of the surface is given by (a constant times) the curvature of the surface at that point. It arises in many physical models where a surface (e.g. a biological cell or a fluid boundary) is subject to surface tension. For this project it is expected that you:

1. Understand and describe the notion of curvature for a non smooth surface such as a triangular mesh. It requires some explanation since a priori the curvature is zero inside triangles and singular a vertices or along edges. The starting point is the Gauss-Bonnet theorem.

2. Learn to use an existing open source library, in this case the one called libigl, to perform that task (along with interactive visualization of the resulting flow). That library has been coded to be user friendly (Python or Matlab like); it is not the most speed efficient one, but it comes with an extended online documentation and tutorials.

3. Use libigl for your own variation (depending on your math interests) of one of the existing tutorial examples it comes with. This will open you the way for many more geometric analysis experiments.

5 Kd-trees and Point location

Kd-trees is another data structure allowing for efficient search of nearest neighbours. In this project you will use them as an acceleration structure in the process of reconstructing or approximating an a priori unknown surface S from a point cloud (think of the latter as a "dense" given sample of points on or close to the surface S). In recent years, mesh modeling through LIDAR scanning point clouds has become the method of choice in many applications, where only CAD (both time consuming and limited in scope) was available in the past. The data collected by the laser contains (in particular) the point positions. For a number of PDE based algorithms trying to reconstruct the surface S, it is important to first estimate the approximate normal vectors to the surface S at each of the sample points. This is our goal in this project. You will have to:

1. Have a look at how LIDAR scanning works in practice (you do not need to understand the details, but what it does and what it records).

2. Write or use code to be able to read an airborne LIDAR point cloud in .las format, and record point positions into an array.

3. Understand the notion of a Kd-tree.

4. Write or use code to build a Kd-tree (here K=3) from the point cloud obtained in 2).

5. For each of the sample point in the cloud, locate (using the Kd-tree) the nearest p ones (you may use p=10 at first) and compute the best plane approximation of these neighbours (this is the eigenvector corresponding to the smallest eigenvalue of some covariance matrix, but you may use a linear regression as well). The normal to that plane is then used as the approximation of the surface normal at the given sample point.

6. Note that the previous only gives us a direction up to an orientation (we may flip it). Discuss/study the possibility to best orient these normals so that the orientation is globally "continuous" all across the surface.

Test your code using airborne LIDAR files for Switzerland available at `https://www.swisstopo.admin.ch/fr/geodata/height/surface3d.html`. You may save you positions and normals to the format of your choice (e.g. .ply or .xyz) and visualize them in a mesh editing tool like Meshlab.

If you are interested in the PDE method of choice for surface reconstruction from positions and normals, read the paper Screened Poisson reconstruction by M. Kazhdan and H. Hoppe (https://www.cs.jhu.edu/ misha/MyPapers/ToG13.pdf). It is also available as a plug-in in Meshlab.

6 PageRank and Sparse matrices

The goal of this project is to implement and analyze a parallel PageRank computation on a randomly generated and possibly large directed graph. The project combines sparse linear algebra, graph partitioning, and domain-decomposition (DD) methods.

The PageRank algorithm aims to rank web pages by relevance and to this end, the web is modeled as a directed graph where nodes represent web pages and edges represent hyperlinks. From this, a column-stochastic transition matrix P is built, where each entry $P_{i,j} = 1/\text{outdeg}(j)$, if page j links to page i and 0 otherwise. The goal is to find a probability vector p representing the long-term likelihood of a random web surfer visiting each page. This is expressed as a fixed-point equation

$$p = \alpha P p + (1 - \alpha)v,$$

with $\alpha \in (0, 1)$ being a damping factor (typically we choose it 0.85) that represents the probability of following a link, and v is the teleportation vector modeling random jumps to any page. The problem can then be rewritten as the linear system

$$(I - \alpha P)p = (1 - \alpha)v.$$

1. Generate a directed graph and set up this problem via sparse matrices and solve it via direct solver.

2. Use a graph partitioning library (e.g., Metis) to subdivide the graph and use PETSc to solve it iteratively via domain decomposition associated to the partitioning.