CoMMA, a geometric unstructured agglomerator

Riccardo Milani

ONERA, Aerodynamics, Aeroelasticity, Acoustics Department - NFLU
riccardo[dot]milani[at]onera[dot]fr

Novembre 2023

Abstract

CoMMA, which stands for COarsening Mesh Multigrid Agglomerator, is a C++ library which provides agglomeration algorithms (in the context of multigrid solvers) for unstructured meshes.

The goal of this document is to clearly state and explain how CoMMA works, that applies both to algorithms and their actual implementation in C++ (e.g., which data structures have been used). After having read this document, the user should be able to understand what CoMMA actually does under the hood and should have the essential insights to use it (e.g., which input parameters should one provides, how they will impact the final results...).

Disclaimer: The content of this report has been adapted from R. Milani (Nov. 2023). *CoMMA, a geometric unstructured agglomerator*. Tech. rep. RT 7/30485. ONERA. People wishing to cite this work may use the following:

```
@techreport{CoMMA23,
  author = {Milani, Riccardo},
  title = {{CoMMA}, a geometric unstructured agglomerator},
  institution = {ONERA},
  number = {RT 7/30485},
  year = {2023},
  month = {November},
}
```

Keywords: Multigrid, Agglomerator, Unstructured

Contents

Glossary

anisotropic line Line joining anisotropic cells through their highest coupling interface.

compactness degree In a coarse cell, the minimum number of neighbours belonging to the same coarse cell.

neighbourhood order Cell c is a neighbour of cell d of order n if the minimum length of the path joining c to d is n.

seeds pool Set of fine cells (e.g., their ID) which will be used as seed (i.e., where to start) to grow a coarse cell.

Cell types

corner Cell with 3 boundary faces.

ridge Cell with 2 boundary faces.

valley Cell with 1 boundary faces.

interior Cell with no boundary faces.

Acronyms

AR aspect ratio.

BL boundary layer.

CC coarse cell.

CRS compressed row storage.

FC fine cell.

1 Introduction

CoMMA, which stands for COarsening Mesh Multigrid Agglomerator, is a `C++` library which provides agglomeration algorithms (in the context of multigrid solvers) for unstructured meshes.

The main features of CoMMA are:

- **Ability to work both with 2 and 3D meshes**.
- **Minimal knowledge of the mesh**. Indeed, it works on a graph representation of the mesh where, roughly speaking, what matters is the connections between the cells and related weights (in 3D, surfaces of the shared faces and volumes of the cells).
- **Detection and special treatment of anisotropic regions**: anisotropic cells are tagged (whenever the aspect ratio (AR) is above a user-defined threshold), then lines joining anisotropic cells are identified and coarse cell (CC) are created by joining fine cell (FC) pairwise.
- **Structured-like coarsening of structured-like regions**: CoMMA aims to obtain structured CC out of structured FC.
- **Preservation of fine faces and connectivity inside a CC**: each mesh entity (vertex, edge, face or cell) of the underlying fine mesh is kept untouched (no remeshing nor modification), moreover the connectivity of the FC belonging to the same CC is kept as well.
- **Optimization of the CC w.r.t. the AR**.
- **Sequential**: CoMMA is not coupled with the partitioner, it takes a domain and has no knowledge of other existing domains (possibly none).

CoMMA has been registered to APP, Agency for the Protection of Programs Paris[1], its IDDN[2] identification number is: IDDN.FR.001.420013.000.S.X.2023.000.31235.

In section 2 the general structure of CoMMA is given. Then, the main concepts and ideas of CoMMA are detailed in section 3 and the references on which the design of CoMMA hinges are given in section 4. In section 5, we give some insights about how CoMMA has been coded, for instance which classes are used. Finally, in section 6 we give some insights about best practices, known drawbacks, and possible improvements. Some appendices describe the latest developments (appendix A), make some remark and possible improvements about the AR (appendix B), and present some deprecated data structure who are still part of the project (appendix C).

2 Overview of CoMMA

CoMMA is mainly developed as a header-only `C++` library. However, a `python` interface has been exposed with `pybind11`: this has been proven particularly useful for development and debugging purposes.

The usage of CoMMA is simple. First of all, being header-only, CoMMA does not require any compilation. Moreover, the final user can access one function only, `agglomerate_one_level`. This function takes as input a graph-based description of the computational mesh (plus some additional pieces of info) and several parameters to tune the agglomeration algorithm. Then, it returns a vector telling which FC belongs to which CC.

[1] `https://www.app.asso.fr/lagence`

[2] `https://www.iddn.org/cert`

Figure 1: CoMMA's logo: a comma "`,`" over a honeycomb pattern which should remind the ability of CoMMA to work with generic meshes.

2.1 Input arguments

We give below the complete list of input arguments of `agglomerate_one_level`. We grouped them by affinity. Nonetheless, their order of appearance respects the order needed by the above mentioned function.

- Graph description: the first input arguments provide the compressed row storage (CRS) representation of the graph of the computational mesh, that is, the connectivity. Two cells are connected if they are direct neighbours, that is, if they share an interior face (in 2D, an edge). The weight (the value itself of the CRS graph) associated to a connection is the area of the shared face (in 2D, the length of the shared edge). The weight associated to a cell is its volume (in 2D, area).
  - `adjMatrix_row_ptr`, `adjMatrix_col_ind`, `adjMatrix_areaValues`, `volumes`
  - Example: given a cell with ID `i`, its neighbours are those in `adjMatrix_col_ind` from index `adjMatrix_row_ptr[i]` to index `adjMatrix_row_ptr[i+1]`
  - The user should prepare their mesh data to respect this format
- Additional info about the mesh:
  - `centers`: the centres of the cells are used to compute the AR of CC.
  - `priority_weights`: additional user-defined weights are used to define the order of the seeds pool (the higher the weight, the higher the priority).
  - `anisotropicCompliantCells`: only a sub-set of the cells might be considered when looking for anisotropic cells.
  - `n_bnd_faces`: vector telling how many boundary faces (in 2D, edges) a cell has.
- Anisotropy-related info
  - `isFirstAgglomeration`: if it's the first call to CoMMA, some preprocessing steps (e.g., computation of anisotropic lines) should be performed. Notice that the name is somewhat misleading and the argument mainly acts as switch for the computation of anisotropic lines.
  - `is_anisotropic`: turn on the special treatment of anisotropic cells.
  - `threshold_anisotropy`: a cell is considered anisotropic if its AR is above this value. If negative, no check is done and all the compliant cells (see above) are considered as anisotropic.

- seeds pool computation (see section 3.2)
  - `seed_ordering_type`: seeds pool type.
- Outputs
  - `fc_to_cc`: once the computation is done, the result is given under the form of a vector, `fc_to_cc`, telling which FC belongs to which CC.
  - `agglomerationLines_Idx`, `agglomerationLines`: If the anisotropic treatment has been activated, the description (in a CRS-like format) of the anisotropic lines are returned as well: it will be used in the following coarsening level.
- Tuning the agglomeration algorithms
  - `correction`: whether a correction step (avoid leaving CC composed of just one FC) should be performed.
  - `dimension`: 2- or 3D.
  - `goal_card`, `min_card`, `max_card`: desired, minimum and maximum cardinalities for the CC.
  - `fc_choice_iter` (default: `1`): how many iterations should the algorithm for identifying the best FC should do.
  - `type_of_isotropic_agglomeration` (experimental, we advise to use its default value): how the search for new FC should be performed.

2.2 Agglomeration overview

`agglomerate_one_level` performs the following steps:

1. Initialization:
   1.1. Create the seeds pool (see section 3.2);
   1.2. Create the dual graph, that is, the CRS graph of the mesh;
   1.3. Create the CC container;
2. Agglomerate:
   2.1. If requested, agglomerate anisotropic cells;
      2.1.1. Agglomerate;
      2.1.2. Update the seeds pool;
   2.2. If necessary, initialize the seeds pool: always done if no anisotropic agglomeration, otherwise it depends on the chosen seeds pool type;
   2.3. Agglomerate isotropic cells: CC are created by joining neighbouring cells with a front-advancing greedy algorithm in order to have a cardinality included in the user-defined bounds;
   2.4. If requested, fix the CCs, that is apply corrections aiming to avoid CCs composed by only one FC. This is achieved by merging this FC into a neighbouring CC.

3 Main concepts

In the following sections, we explain the main concepts (both entities and strategies) on which CoMMA relies in order to give more details on how the steps presented in section 2.2 work.

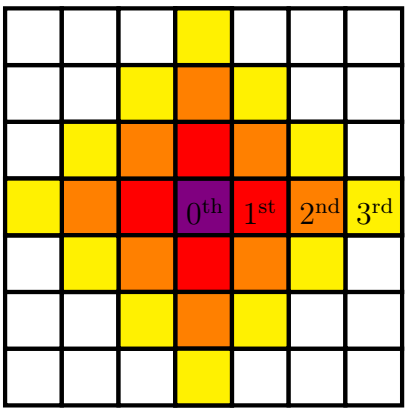


Figure 2: Neighbourhood orders with respect to the violet cell. Red - first order; orange - second order; yellow - third order.

3.1 Neighbourhood

The neighbourhood of a (fine or coarse) cell is the set of cells which are *close* to the reference once. The definition of "*close*" is intentionally left generic at this point because which cells truly belong to a neighbourhood as CoMMA interprets it depends on several factors.

First of all, it is useful to introduce the notion of neighbourhood order. A vertex (i.e., cell) is of order n with respect to another vertex if the minimum-length path joining the two vertices is composed of n edges (interfaces). For instance, suppose: i) cell A shares a face with cell B, ii) cell B shares a face with cell C, iii) A and C do not have any common faces. Then C is a *second* order neighbour of A (and vice-versa) since moving from A to C involves two steps, that is, from A to B, then from B to C. For a graphical example, see fig. 2.

3.1.1 Neighbours and candidates

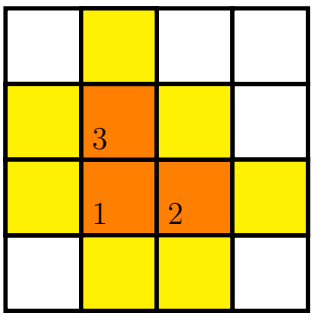
When evaluating which FC to include next in the current CC, not all the neighbours are considered: we define *candidates* those cells that are actually considered.

First of all, before starting agglomerating, a maximum neighbourhood order is set: once the seed of the CC (that is, the FC from which CC grows, see section 3.2) has been selected, only the its neighbours of order less or equal to the maximum one can be chosen to join the CC; see bottom strip of fig. 3 to understand how this can limit the agglomeration. Setting a maximum order aims to have more *compact* CC. The maximum order is not controlled by the user and it is computed using the dimension and the maximum cardinality.

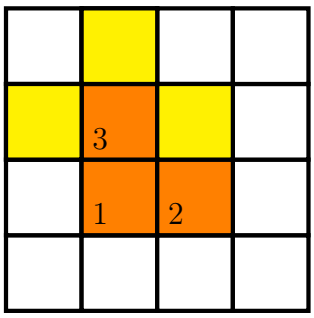
In CoMMA there exist two types of neighbourhood and whether a cell is considered a candidate or not mainly depends on them.

- In the **extended** neighbourhood all the direct neighbours of the current CC are candidates (unless they are already part of a CC, it could be the current one, or outside the maximum allowed neighbourhood order).

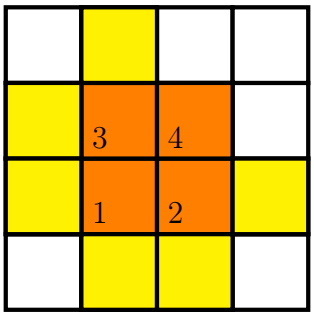
- In the **pure front-advancing** neighbourhood only the direct neighbours of the lastly added FC are considered candidates (unless they are already part of a CC, it could be the current one, or outside the maximum allowed neighbourhood order). An exception is considered if the cardinality of the current CC is less than or equal to the dimension of the problem, in this case, the extended type applies. This exception has been introduced to favour more compact cells. If no direct neighbours of the last FC are eligible (e.g., already part of a CC, outside the maximum neighbourhood order,...), we consider candidates already considered in the previous steps.



(a) Extended neighbourhood



(b) Pure front-advancing neighbourhood



(c) Limitation by maximum neighbourhood order

Figure 3: Candidate FC (yellow) to add in the current CC (orange, the numbers state the order in which the FC were added to the CC). Top: difference between the two available neighbourhood types: extended (left) and pure front-advancing neighbourhood (right). Bottom: having considered a maximum neighbourhood order of 2, the neighbours of cell 4 are not considered as candidates since they are all of order 3; case with extended neighbourhood.

The top part of fig. 3 shows the difference between the two types of neighbourhood in a common situation.

The type of the neighbourhood is controlled by an input argument, defaulted to use the extended one.

Warning: the pure front-advancing neighbourhood is still experimental, moreover, since it usually yields less candidates, its results might be worse than the (standard) extended type; for this reason, we advise to use the extended neighbourhood.

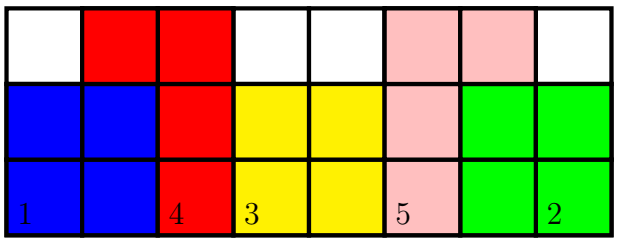
3.2 Seeds pool

A so-called seed is a FC from which a new CC starts growing. By consequence, a seeds pool is the set of the FC used as seed.

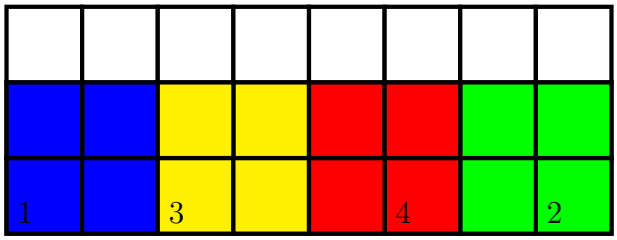
The order in which the FC are considered in the pool is of paramount importance for the results. How this order is computed depends on the initialization and, once the agglomeration is started, a set of criteria (we do not include whether a cell has been agglomerated which is trivial):

SPO.BF The number of boundary faces of a FC: the higher this number, the higher the priority. It has been often identified in the literature (e.g., by Mavriplis) as a good practice: we consider corners, then ordinary boundary cells, and, lastly, interior ones.

SPO.Neigh Whether the cell is a direct neighbour of an already agglomerated CC: higher priority if it is. It enables one to avoid holes between agglomerated cells that can lead



(a) Without neighbour priority



(b) With neighbour priority

Figure 4: Example of agglomeration with cardinality 4 to show the interest of neighbour priority, item SPO.Neigh. Without it, late CC might be constrained to grow in undesirable shapes dictated by the earliest CC.

to undesirable results, see fig. 4, and gives CoMMA the characteristic feature of front-advancing agglomeration.

SPO.Wei Priority weights given by the user: the higher the weight, the earlier the cell is considered. It kicks in when the previous two fail to define a unique order, for instance, at the very beginning it is used to choose from which corner the agglomeration should start.

SPO.ID The cell numbering in ascending order: the lower the ID, the earlier the cell is considered. It is used as very last resort if everything else failed.

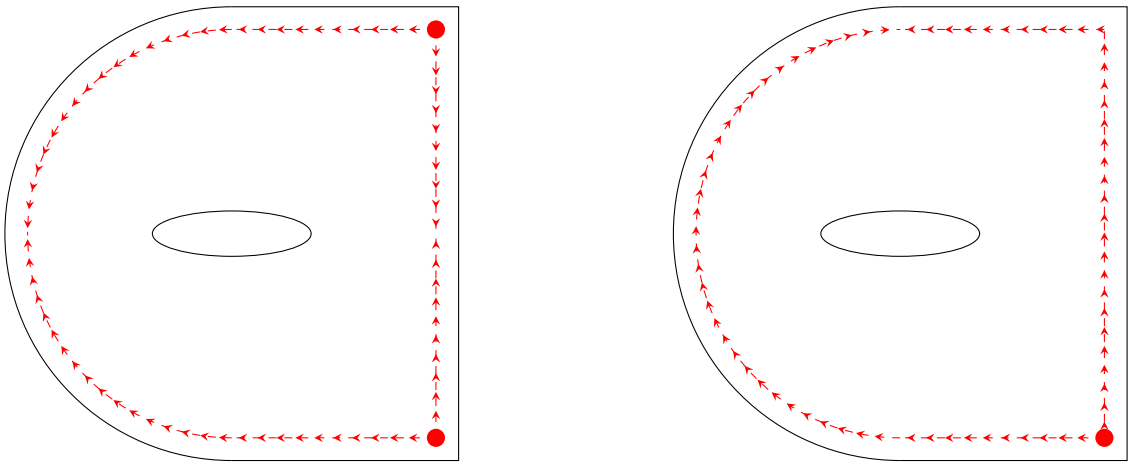
The initialization and the priority given to the above criteria can be chosen by the user via input argument `seed_ordering_type`.

The sections that follow give more details about the relations between the seeds pool and the agglomeration (section 3.2.1) and the initialization (section 3.2.2) and the priority criteria (section 3.2.3). Finally, some insights about the actual implementation are given, section 3.2.4.

3.2.1 The impact of the seeds pool on the agglomeration

The seeds pool applies only to the isotropic agglomeration, but it is influenced by the anisotropic stage, whenever that is activated.

More specifically, the anisotropic stage is completely independent of the seeds pool, that is, the order of agglomeration during the anisotropic stage is not dictated by the seeds pool (see section 3.3 below for more details). However, the results of the anisotropic stage do affect the seeds pool. Indeed, the neighbours of the cells agglomerated during the anisotropic agglomeration are added to the seeds pool, and, hence, those cells will appear among the first seeds during the isotropic agglomeration.



(a) Full initialization

(b) Point initialization

Figure 5: Seeds pool initialization on a typical mesh with an airfoil at its centre. With full initialization (left), the 2 corners (red dots) are put in the initial seeds pool; accordingly, the agglomeration will advance on 4 fronts (2 per each corner), see red paths arrows. With point-initialization (right), only one corner is considered and only 2 fronts are hence formed.

3.2.2 Initialization

At the initialization step, criterion SPO.Neigh cannot be taken into account, clearly. Among those who are left, the one with the highest priority is the boundary faces, SPO.BF: as mentioned above, this seems to be identified as a beneficial strategy in the literature. Then, priority weights SPO.Wei are considered, and finally, ID, SPO.ID. This chain of priority means that boundary cells will be used as seeds at early stages of the agglomeration and before interior faces.

For instance, in a square mesh, the 4 corners will be considered first. However, the user can decide if include all these four corners in the very first seeds pool (so-called *full initialization*), or just one (so-called *point initialization*).

With the *full initialization*, all the, let's say, corners are sought, ordered (according to SPO.Wei and SPO.ID), put into the seeds pool, and, only, then the agglomeration is started. Going back to the square mesh example, the corners will be the first seeds, then CoMMA will consider neighbours of the CC.

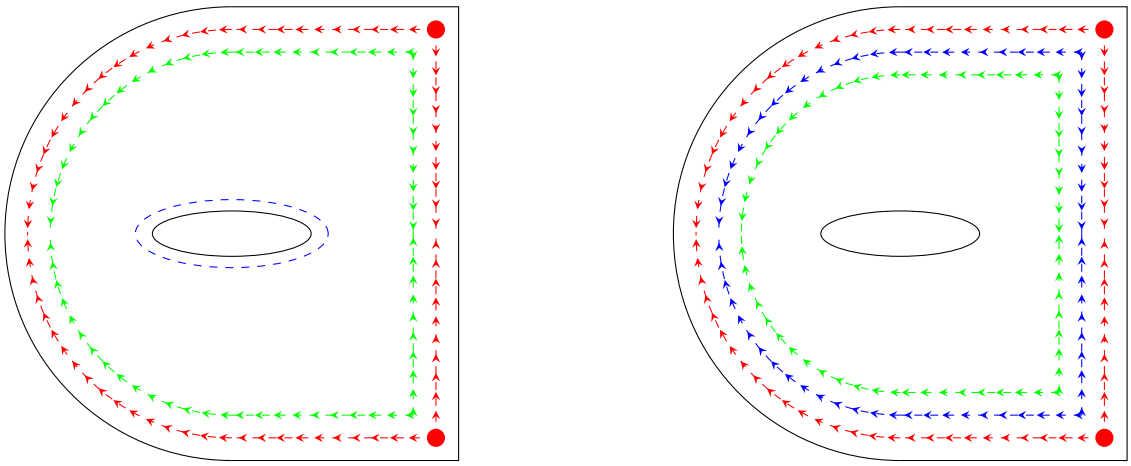
With the *point initialization* only one corner is put into the seeds pool (chosen on the bases of SPO.Wei and SPO.ID). Hence, in the square mesh example, the first seed would be a corner, but the second will not be, since a neighbour of the first CC is chosen.

Figure 5 provides a graphic example of the initialization on a typical mesh and the impacts that have on the later stages of the agglomeration.

3.2.3 Priority chain

The user can (partially) chose the priority to give to the criteria mentioned above and which will be considered when updating the seeds pool. In particular, the user can choose to give higher priority to the boundary faces, criterion SPO.BF, or to neighbourhood, SPO.Neigh, whereas, criteria SPO.Wei and SPO.ID will always have lower priority.

A typical case where choosing SPO.BF over SPO.Neigh (or vice-versa) has a true impact is pictured in fig. 6. We consider the same mesh as before, and in particular, we retain the full initialization, fig. 5a. We also start agglomerating and complete the exterior region, see red paths and arrows. At this point, in the seeds pool there are only the neighbours of the



(a) SPO.BF: Seeds pool ordering by boundary priority

(b) SPO.Neigh: Seeds pool ordering by neighbourhood priority

Figure 6: Seeds pool evolution on a typical mesh with an airfoil at its centre. The red lines show the starting state of the example. The starting points are marked with red dots. Left: with priority by boundary faces, one then agglomerates the inner border around the airfoil, see blue line, and then switch back to the outer zone, green line. Right: with priority by neighbourhood, one continues in the outer zone, blue line, then green line.

CC formed so far. These neighbours are all interior cells, however there are still boundary cells which are not agglomerated yet, the airfoil.

- If the priority by boundary faces is chosen, fig. 6a, then, the inner boundary has priority and the seeds pool is rebuilt to put those cells in front. Hence, the second stage of the agglomeration happens at the interior border, see blue path. Notice that no direction (arrow) is given since the order is determined by criteria SPO.Wei and SPO.ID. After having finished the inner border, there are no more boundary cells and we switch to interior cells. We thus consider neighbours of the first CC and we will jump back to the exterior part of the mesh, see green path and arrows: this time the direction is known and reflects the initial one. After that, CoMMA will keep on jumping back and forth from outer and inner part of the mesh, hence resulting in two fronts, one moving inwards, and the other in the opposite direction.

- If the priority by neighbourhood is chosen, fig. 6b, we then use the interior cells and stay in the outer part of the mesh, see blue path and arrows, direction is known and reflects the one of the first stage. The third stage will still happen in the outer zone, see green path and arrows, here again the direction is known. This will result in only one agglomeration front advancing from the outer to the inner part of the mesh.

3.2.4 Implementation

The seeds pool can be thought of, although not exactly equivalent to, a queue, that is, a FIFO (First In First Out) container: we insert elements at the tail and we pop elements at the head. Behind the scenes, the behaviour is more complex and relies on 4 different queues (more particularly STL container `deques`), one for each type of cell according to the boundary faces (corners, 3 faces; ridges, 2 faces; valleys, 1 face; interior, 0 faces). The queue from which the next seed is taken is initially chosen with criterion SPO.BF. Once this queue is completely spoiled, the next chosen is chosen (or built) according to the priority asked by the user.

The life of a seeds pool follows the following steps:

1. When the seeds pool is created, only the queue with highest priority is created
   1.1. Seek the highest level of boundary faces (corners, ridges,...).
   1.2. Set the current reference queue to the one identified by this level.
   1.3. Initialize the queue according to the user choice (cf. section 3.2.2).
2. Every time a CC is created, the seeds pool is updated:
   2.1. The direct neighbours of the CC which are yet to be agglomerated are computed.
   2.2. They are organized with respect to neighbourhood order (see section 3.1), computed with respect to the original seed of the CC.
   2.3. Starting from the lowest order and moving up (this ensure that cells closer to the seed have are inserted in the queue before the farer ones, hence giving the former a slightly higher priority which should help in creating more compact CC), take all the neighbours with the same order
      2.3.1. They are organized in temporary queues according to their boundary faces.
      2.3.2. The temporary queues are sorted according to the priority weights (or cell numbering).
      2.3.3. The temporary queues are added at the back of the related seeds pool queue.
3. When a new seed is requested, if the current queue has still some not-yet-agglomerated elements, return the first one, otherwise an update is necessary:
   - If the priority by neighbourhood has been chosen, take the first non empty queue and return the first element.
   - If the priority by boundary faces has been chosen:
     3.1. As in 1.1., compute the maximum number of boundary faces for which there are still some cells to agglomerate (it might be the same as the previous one).
     3.2. Set the current reference queue:
        – If not empty, return the head.
        – If empty, build it (this step also includes ordering) and return the head.

3.3 Anisotropic agglomeration

Before getting into the details of the agglomeration itself, let us introduce two definitions.

A cell is *anisotropic* if their AR is greater than a (user-defined) threshold. In CoMMA, the AR of a FC is computed as follows:

$$\mathrm{AR}_{fc} = \sqrt{\frac{\max_i w_i}{\min_i w_i}} \geq 1 \tag{1}$$

where $\{w_i\}$ is the set of the weights of the neighbours of the current cell, that is areas (resp., length) of the shared faces (resp., edges) in 3D (resp., 2D). In 2D, the square root of (1) is not needed. With (1), we are trying to approximate the following definition of AR:

$$\mathrm{AR}_{\mathrm{ref}} \coloneqq \frac{\max\limits_{i,j,i\neq j} d\left(v_i, v_j\right)}{\min\limits_{i,j,i\neq j} d\left(v_i, v_j\right)} \tag{2}$$

where $\{v_i\}$ are the vertices of the cell and $d(\cdot,\cdot)$ is the Euclidean distance. Unfortunately, in CoMMA one does not have information about the vertices.

Remark - anisotropy threshold. The inequality at the right-hand side simply follows from the definition (1): this is the reason why whenever the input argument `anisotropy_threshold` is less than 1, its inverse will be considered instead. However, if the input argument is negative, no check is performed and all the compliant cells are considered as anisotropic. Hence:

- If `anisotropy_threshold`< 0, no anisotropy check is performed.
- If $0 <$`anisotropy_threshold`< 1, the check is evaluated against $\dfrac{1}{\texttt{anisotropy\_threshold}}$.
- If `anisotropy_threshold`> 1, the check is evaluated against `anisotropy_threshold`.

For more details about the AR, see appendix B.

A second definition concerns the *agglomeration lines*, that is, as a line joining anisotropic cells through their highest coupling interface. We will use *head* and *tail* to refer to the two ends of a line, but it is important to stress that there is no order intrinsic to the line itself, and those are just terms borrowed from the data-structure domain (cf. queue) which are used for clarity sake.

The algorithm for anisotropic agglomeration is:

1. If agglomeration lines need to be computed (cf. `isFirstAgglomeration` input parameter), they are created.

   1.1. First of all, CoMMA analyses all the compliant cells provided by the user and tags those with a high AR as described above.

   1.2. We select one anisotropic cell that will be the starting point and build a line. Some brief remarks concerning the construction of a anisotropic line:

   - The ordered on which the cells as starting point are considered is determined by the priority weights provided by the user.
   - One cell is considered and its line is built entirely. Then the next non agglomerated cell is considered, its line is built and so on.
   - Only lines of length of at least 2 are considered. This means that, for instance, an anisotropic cell which has only isotropic neighbour won't belong to any anisotropic line, hence it will be agglomerated using the isotropic agglomeration strategy.
   - The computation of a line ends when nor its head nor tail has valid candidates to join. A cell is considered among the candidate pool if:

   AnC1 It's anisotropic,

   AnC2 It does not belong yet to an anisotropic lines,

   AnC3 It's a neighbour of the current head/tail of the line, and

   AnC4 It's connected to the current head/tail through the side with highest weight (that is, face with largest surface) with respect to the head/tail.

2. Anisotropic agglomeration is performed. Some remarks:

   - Anisotropic agglomeration always happens before the isotropic one.
   - It agglomerates cells belonging to the *same agglomeration line*.
   - The CC are composed of 2 FC. This quantity is fixed and cannot be set by the user. However, there could be clusters composed of 3 FC only if only 3 cells are left in the line.

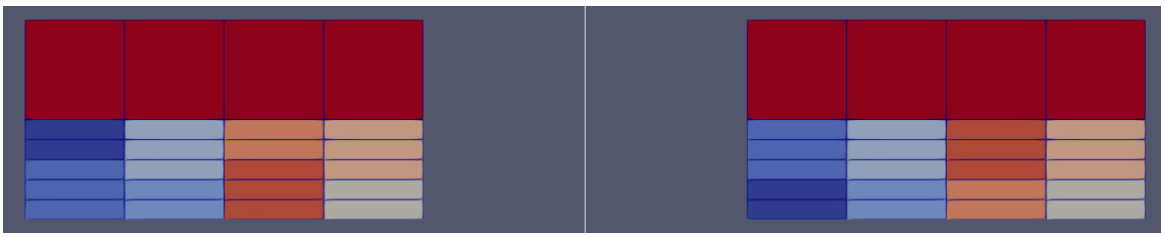


Figure 7: Result of anisotropic agglomeration with (right) and without (left) forcing the agglomeration of the line to start at the boundary.

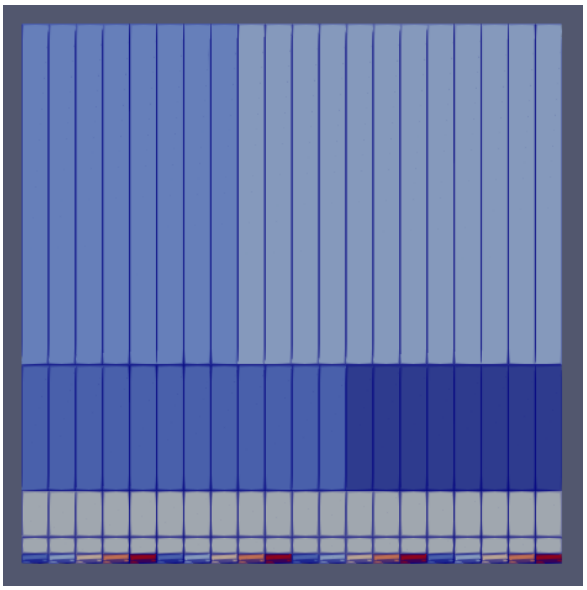


Figure 8: Result of an anisotropic agglomeration where the head-or-tail trick did not lead to the desired results.

- The agglomeration starts from the head or the tail of the line accordingly to which of the two has more boundary faces: this is important in order to get homogeneous treatment on the border if 3-cells clusters had to be created, see fig. 7. The trick, however, do not solve major pathological cases, see fig. 8.

3. The last step is updating the seeds pool with the neighbours of the anisotropic cell (according to the seeds pool type a further initialization might be needed).

    - This step has been introduced to addresses issues as the one presented in fig. 4.

In the case of several calls to CoMMA (e.g., multigrid algorithm with several coarsening stages), the lines are created during the first call, hence on the finest level, returned, and used also in the in the following calls.

Notice that the input argument `isFirstAgglomeration` is somewhat misleading and mainly acts as switch whether or not the computation of anisotropic lines should be performed. Hence, one may pass to CoMMA lines computed by another tool, provided that they follow the format accepted by CoMMA.

3.3.1 Practical definition

The definition of the anisotropic lines returned by CoMMA relies on three vectors, similarly to the CRS format, `fc_to_cc` (fine-to-coarse), `aniso_ln_idx`, `aniso_ln`. Recall that these are three output parameters of `agglomerate_one_level`. Here is how it works:

- The i-th line is composed of the **CC** with indices from `aniso_ln_idx[i]` to `aniso_ln_idx[i+1]`
- In order to see to which **FC** this corresponds to we have to take into account `fc_to_cc`

We provide the `python` code building a dictionary with key the indices/IDs of the anisotropic lines and with values the IDs of the FC composing the line:

```
import numpy as np
# fc_to_cc, aniso_ln_idx, aniso_ln = CoMMA.agglomerate_one_level(...)
fc_to_cc = np.asarray(fc_to_cc, dtype=int)
dic = {}
# Option 1:
# For every line...
for i in range(len(aniso_ln_idx)-1):
    mask = np.full(len(fc_to_cc), False)
    # For every coarse cell in the line...
    for cc in aniso_ln[aniso_ln_idx[i]:aniso_ln_idx[i+1]]:
        # Take all the fine cells that are in the coarse cell
        mask |= fc_to_cc == cc
    dic[i] = np.flatnonzero(mask)
# Option 2 [list-comprehension]:
dic = {
    i: np.flatnonzero(
        np.isin(
            fc_to_cc,
            aniso_ln[aniso_ln_idx[i]:aniso_ln_idx[i+1]]
        )
    )
    for i in range(len(aniso_ln_idx)-1)
}
```

3.4 Isotropic agglomeration

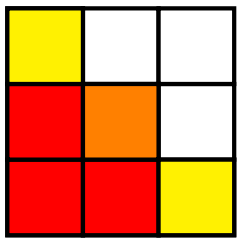
The algorithm with which isotropic cells are treated is greedy and under no circumstances the algorithm is forced to preserve the basic structure of the underlying mesh, that is, it would not (always) give quads when agglomerating quads. This preserving feature is sometimes referred to "isotropy" but this is not how we mean it. With *isotropic* agglomerator we refer to the algorithm that takes care of the isotropic cells.

This is how a CC is created:

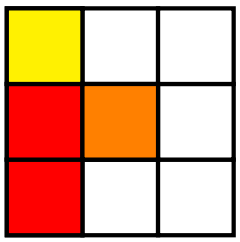
1. Select a seed (see section 3.2 for more details): among the FC that are not yet agglomerated, choose the one from where to start building the next CC
2. Compute the largest neighbourhood of allowed cells: starting from the seed, compute the set of FC that can be potential candidates for the CC. This is done by providing a neighbourhood order.
3. Until there are no FC left (in the neighbourhood) to agglomerate or until we have reached the requested maximum cardinality for a CC, perform the following steps:

   3.1. Update the neighbourhood by adding the direct neighbours of the last added FC.

   - Mind: these new neighbours should also be in the list of the allowed cells computed before (see point 2 and bottom part of fig. 3).



(a) Choice by shared faces - BFC.1



(b) Choice by AR - BFC.3

Figure 9: Examples of choice of the best next FC to agglomerate. Colours: red - current CC; orange - chosen FC; yellow - other candidates.

3.2. Test all the candidate FC and select the best one. For more details, see section 3.4.1 below.

3.3. Store the current intermediate CC.

4. Among all the intermediate CC, choose the one ensuring the **highest compactness degree** and, in case of equality, the **lowest number of external faces**.

   - Only intermediate CC with cardinality greater than the requested minimum one are considered in the choice.
   - If there are several intermediate CC yielding similar features, the algorithm chooses
     - the one with cardinality equal to the goal one, otherwise
     - the one with highest cardinality.

The above steps are repeated until all the FC have been agglomerated.

3.4.1 Choice of the best next FC

Three criteria are used to choose which cell among all the available candidates will be agglomerated in the current CC (in descending order of importance):

BFC.1 Number of shared faces: the higher, the higher the priority.

BFC.2 Neighbourhood order: the lower, the higher the priority.

BFC.3 AR: the lower, the higher the priority.

Instead of the actual AR, we compute an approximation which, roughly speaking, tells us how much a (coarse) cell is stretched with respect to a cube with the same volume as the cell. It is computed with the following formula:

$$\mathrm{AR}_{cc} = \frac{\max\limits_{fc_i, fc_j \in cc} d\left(c_{fc_i}, c_{fc_j}\right)}{\sqrt[3]{\mathrm{vol}(cc)}} \tag{3}$$

where cc is the CC, $\{fc_i\}_{cc}$ is the set of the FC in cc, c_{fc_i} is the centre of fc_i, $d(\cdot,\cdot)$ is the Euclidean distance, and $\mathrm{vol}(\cdot)$ the volume. The formula slightly changes if considering a 2D problem: the volume becomes a surface and the square-root (instead of the cube one) is considered. For more details about the AR, see appendix B.

For the sake of clarity, pseudocode detailing the choice of the best FC is given in algorithm 1, while fig. 9 gives two graphical examples.

Algorithm 1: Pseudocode for the choice of the best FC

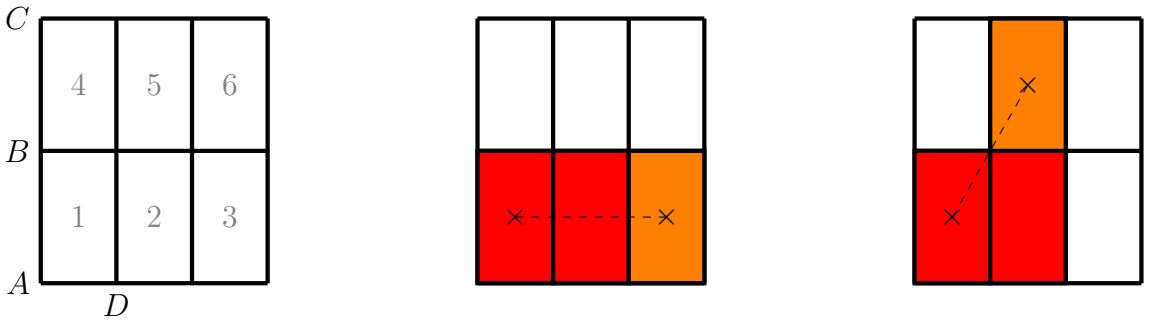
```
Data: cur_CC: set of FC composing the current CC;
      seed: seed of the CC
Result: best_FC: FC to agglomerate to the CC
best_FC ← ∞;
max_shared_faces ← 0;
min_AR ← ∞;
ref_order ← ∞;
foreach FC in ComputeCandidatesFC(cur_CC) do
    shared_faces, AR ← ComputeCCFeatures(cur_CC ∪{FC});
    order ← ComputeNeighOrder(seed, FC) ;
    if shared_faces < max_shared_faces then          /* Criterion BFC.1 */
        best_FC, max_shared_faces, min_AR, ref_order ← UpdateReference(FC,
         shared_faces, AR, order);
    else if shared_faces = max_shared_faces then
        if order < ref_order then                    /* Criterion BFC.2 */
            best_FC, max_shared_faces, min_AR, ref_order ← UpdateReference(FC,
             shared_faces, AR, order);
        else if order = ref_order and AR < min_AR then  /* Criterion BFC.3 */
            best_FC, max_shared_faces, min_AR, ref_order ← UpdateReference(FC,
             shared_faces, AR, order);
        end
    end
end
```

Iterative algorithm Consider the mesh in fig. 10: it is composed of two layers of slightly stretched rectangles, with $\overline{AB} = \frac{7}{4}\overline{AD}$ and $\overline{BC} = \overline{AB}$. For the sake of simplicity take also $\overline{AD} = 1$. Suppose that we have already agglomerated two cells, cells 1 and 2, the red ones, with cell 2 being the seed. We try to trace the steps of CoMMA's algorithm for the choice of the best FC and consider cells 3 and 5, both direct neighbours of the seed (cell 2). One should also consider cell 4, however, it is of second order (cf. neighbourhood order) with respect to the seed, hence it would have lower priority, moreover, its AR is the same as the one obtained with cell 4. It is clear that, aiming to have a CC of cardinality 4, one would prefer to grow as in the second case, fig. 10c, that is, including cell 5, so that in the next round, by adding cell 4, a nice rectangle is recovered.

Cells 3 and 5 share the same number of faces with the current CC and are of the same order with respect to cell 2, hence, the choice comes down to the AR. We now compute it following (3). The final CC of the two cases, fig. 10b and 10c, have the same volume. The diameter of fig. 10b is 2, whereas the one of fig. 10c is $\sqrt{(}1 + \frac{49}{16}) \simeq 2.01$, and hence it has a greater AR. The algorithm would choose to agglomerate cell 3 as in fig. 10b, and, in the following step, cell 5 will be added, hence forming a T-shaped CC.

The example above pinpoints a drawback of the standard algorithm for the choice of the FC: indeed, we are trying to solve an optimization problem (max shared faces and min AR) on the set of the direct neighbours of the CC (in the example above, all the CC of cardinality 3 containing cells 1 and 2). However, the solution is local and might not be optimal on a different set (for instance, the set of CC of cardinality 4 containing cells 1 and 2).

One improvement would be to consider an iterative algorithm that try to foresee n agglomeration steps and retain the direct FC yielding the best CC after those n steps. This behaviour is activated when the input parameter `fc_choice_iter` is greater than 1. Applying a 2-steps



(a) Reference mesh (b) Case 1 (c) Case 2

Figure 10: Example of stretched mesh where T-shaped cluster are possibly formed. FC composing the CC are in red, candidates in orange. The diameters of the CC (max distance between two cell centres) are drawn in dashed lines

iterative algorithm to the mesh presented in fig. 10 would lead to choosing cell 5 as one would expect: indeed, see fig. 11.

3.5 Correction stage

As the agglomeration progresses, it might happen that a seed has no neighbours available for agglomeration, hence a CC of cardinality 1 is created. This could be the case if all its neighbours have already been agglomerated or if it is a cell unconnected to rest of the domain (most of partitioners do not ensure that the domain are connected). Unless it's the latter case (for which there is nothing that can we do since CoMMA is not partition-agnostic), we can try to fix this and agglomerate the singleton with one of its neighbouring CC: that's what the correction does.

The correction is not activated by default and should specifically requested by the user.

Only isotropic CC are considered as candidates: under non circumstances a singular cell is joined to an CC resulting from the anisotropic agglomeration. For instance, in the case all the neighbours are anisotropic, then the singleton remain singular.

The criteria used to choose the CC to which the singleton will be agglomerated are listed below in **descending order**:

Corr.1 If by adding the FC the compactness degree of a CC increases, then this CC has higher priority.

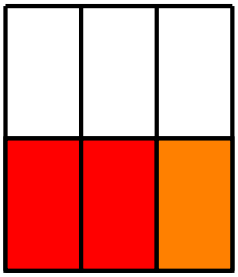
Corr.2 Number of shared faces: the higher, the higher the priority. It allows to have more compact and convex CC.

Corr.3 Cardinality of the CC: the lower, the higher the priority. It allows to have more balanced CC.

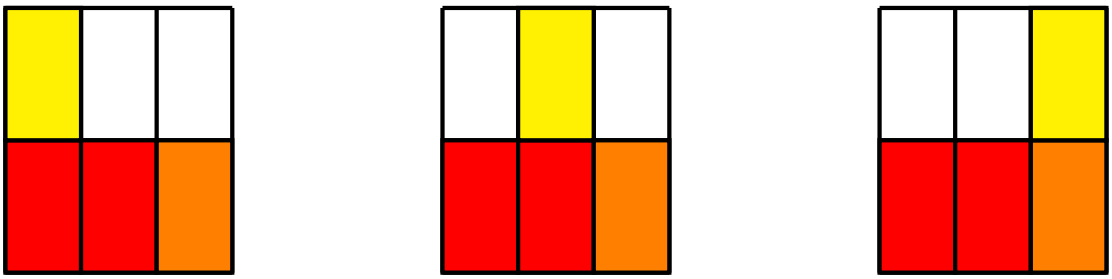
Corr.4 ID of the CC: the lower, the higher the priority. This is used only as last resort if the above ones failed.

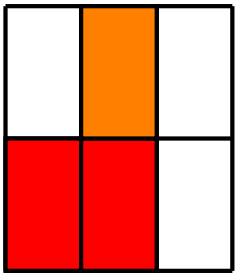
Remark: The above criteria prevail on the requested maximum cardinality, meaning that, for instance, the singleton would always be added to the CC with most shared faces, even though that would mean exceeding the maximum cardinality.

For examples about how corrections work, see fig. 12.



(a) Starting point with candidate cell 3, fig. 10b





(b) Starting point with candidate cell 5, fig. 10c

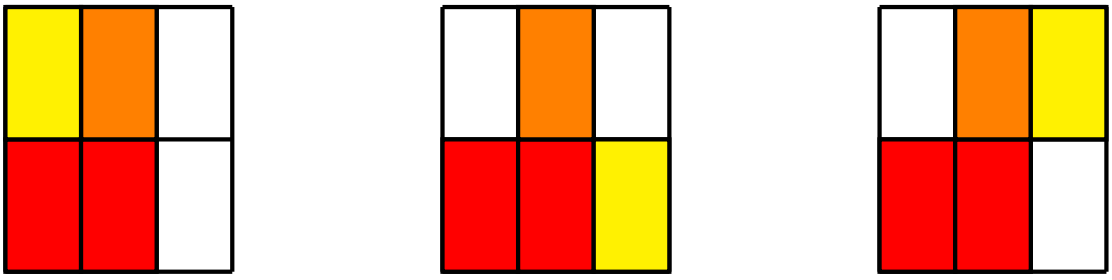
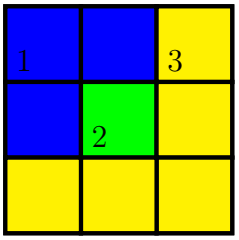
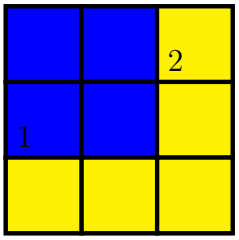


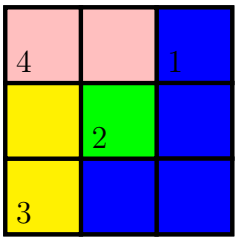
Figure 11: Example of 2-steps iterative algorithm for the choice of FC based on the case presented in fig. 10. The starting CC is in red. First level candidates (first and third rows) are pictured in orange. Since two steps are requested, the algorithm tries to agglomerate one more step, thus it adds also the yellow cells (one at the time), before choosing the FC to agglomerate. The algorithm will consider CC on the second and fourth rows, respectively coming from the intermediated CC on first and third rows. The bottom-left case yields the best CC (max number of shared faces), then its parent, that is, cell 5 (third row), is retained.



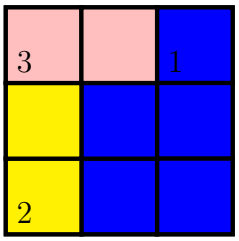
(a) Case 1: Before correction



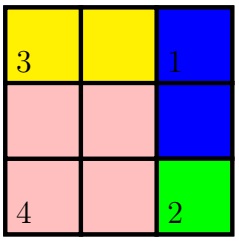
(b) Case 1: After correction - Criterion Corr.1



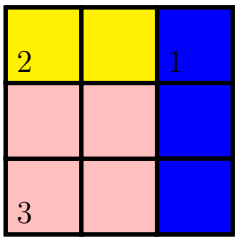
(c) Case 2: Before correction



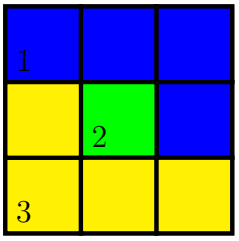
(d) Case 2: After correction - Criterion Corr.2



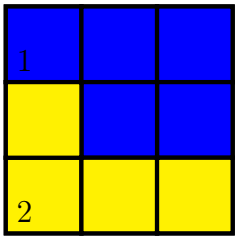
(e) Case 3: Before correction



(f) Case 3: After correction - Criterion Corr.3



(g) Case 4: Before correction



(h) Case 4: After correction - Criterion Corr.4

Figure 12: Example of corrections. Notice as, with the removal of the singleton the IDs of the CC change

4 CoMMA references

CoMMA algorithm and strategy have been inspired by Mavriplis and collaborators: indeed, in their works one can find the pivotal ideas for both isotropic and anisotropic agglomeration, see for instance Mavriplis and Venkatakrishnan, 1995, 1996; Mavriplis, 1997, 1999 and references therein.

Indeed, the priority ranking used for the seeds pool (see section 3.2) and the advised values for the goal cardinality are already present in Mavriplis and Venkatakrishnan, 1995, section 2; whereas, the special treatment for the anisotropic regions was introduced in Mavriplis and Venkatakrishnan, 1996, section 4 and later improved by taking into account a direct use of the high-coupling lines in Mavriplis, 1999, section IV.

There exist, however, some differences. First of all, Mavriplis mainly used vertex-based discretizations thus relying on control-volumes and often, in isotropic region, he just agglomerated all the neighbouring control-volumes of the vertex under consideration. Moreover, CoMMA having minimal knowledge of the mesh and only related to its graph representation, it cannot distinguish between boundary types (e.g., wall, farfield, periodic...); for this reason we are not able to reproduce the exact ranking of the seed priority of Mavriplis and Venkatakrishnan, 1995. Moreover, the ranking was explicitly changed to give higher importance to the neighbours of the already agglomerated cells (see section 3.2).

Let us now have a look at the differences in the anisotropic agglomeration. In order to identify the anisotropic cells used the following quantity (often referred to in his papers with α, see for instance Mavriplis, 1999):

$$\mathrm{AR}_{fc}^{\mathrm{Mavr}} = \frac{\max_i w_i}{\operatorname{avg}\{w_i\}} \tag{4}$$

where avg is the average. This formula should be compared to our (1). Mavriplis often advises to use as weights algebraic values, coming, for instance, directly from the linear system, rather than geometric ones (e.g., surfaces, lengths,...). Usually, Mavriplis used seeds pool even for the anisotropic lines and prioritized those seeds with higher $\mathrm{AR}_{fc}^{\mathrm{Mavr}}$.

5 Practical implementation details

We give here a brief overview of how the concepts introduced in section 3 are practically taken into account in the implementation: this aims to be a general view of CoMMA code. We also put in appendix C a briefly overview of some data structure which were once used but are now deprecated.

All the items below except of agglomeration lines are indeed custom `C++` classes.

- **Graph**: structure used for storing a CRS representation of a graph. It also holds the weights of the edges (that is, in a mesh context, the area of the surfaces shared by two cells) and of the nodes (that is the volumes of the cells).
  - **Dual graph**: representation in *Graph* form of the fine mesh given as input. It also holds the centres of the cells and a set of cells which should be considered when seeking anisotropic cells.
  - **Subgraph**: local graph of FC inside the same CC.
- **coarse cell**: holds the set of (fine) cells composing the CC, the reference to the global dual graph, its own sub-graph and a mapping to switching from global (dual/global graph) to local (sub-graph) indices.

- **coarse cell container**: a container for CC. It also holds info about the ongoing agglomeration, i.e., the number of CC, the mapping from fine to coarse cells (indeed, the most important output).
- **Seeds pool**: object managing the order of the seeds, that is the FC from which the CC will start growing. It holds the priority weights.
- **Neighbourhood**: object keeping track of the neighbourhood of the growing CC. It provides the FC candidate to being agglomerated in the CC next. Two versions are available:
  - **Extended**: all current direct neighbours of the CC are considered candidates.
  - **Pure front-advancing** (experimental): only the direct neighbours of the FC lastly added to the CC are considered candidates.
- **Agglomeration line** & Agglomeration-line indices: an agglomeration line is basically a list / vector of cell IDs. This list is divided in groups, whose starting points are given by the agglomeration-line indices. Hence, the first group (index `0`) starts at index `agglo_line_idx[0]` and ends at `agglo_line_idx[1]`. Each group contains the IDs of FC agglomerated in the same CC.
- **Agglomerator**: interface class for agglomerators. It holds the dual graph and CC container. Its method `agglomerate_one_level` is where the work is actually done.
  - **Isotropic**
    - **Biconnected**: in theory, it agglomerates cells so that each FC of the CC has at least 2 neighbours FC in the same CC; in practice it is greedy.
      - **Iterative**: it addresses the iterative algorithm for the choice of the FC to include, see section 3.4.1.
  - **Anisotropic**

6 Conclusion

6.1 General remarks and best practices

We gather here some remarks about the general behaviour of CoMMA and give some advice for the final user concerning the input parameters.

6.1.1 An example

We detail below an example with anisotropic agglomeration in order to better understand how the global strategy work. We consider the same mesh as fig. 5, in addition we suppose that there is a BL with anisotropic cells, see dotted zone in fig. 13a. We now start the agglomeration, treated zone will be cross-hatched in the figure; moreover, the zones are numbered accordingly to when they have been agglomerated.

1. The anisotropic agglomeration is the first step, see hatched red zone in fig. 13b.
2. The seeds pool is updated and the cells around the BL are added.
3. The next step is the isotropic agglomeration. At this point, the algorithm bifurcates according to the type of seeds pool:

- The second row of fig. 13 depicts the agglomeration when the priority by **boundary** is chosen:

1BND. At this point, the seeds pool is not empty but there are still boundary cells which are not agglomerated, hence an initialization is needed.

2BND. The outer zone is agglomerated, see blue zone in fig. 13c.

3BND. Once all the boundary cells have been treated, we now consider interior ones. Since the seeds pool has been updated just after the anisotropic step, the next zone to be agglomerated is the one next to the BL, see green zone in fig. 13d.

4BND. We then jump back on the outer zone, and then back on the inner zone, and so on: two agglomeration fronts are hence formed.

- The second row of fig. 13 depicts the agglomeration when the priority by **neighbourhood** is chosen:

1NEI. The seeds pool is not empty, so no initialization is need.

2NEI. We agglomerate the blue blue zone in fig. 13e.

3NEI. We agglomerate the green blue zone in fig. 13f.

4NEI. The agglomeration front keeps on moving outwards.

6.1.2 About the cardinalities

It is often advised to have a goal cardinality of 4 for 2D problems and of 8 for 3D ones. This value should (help to) recover structured CC from structured FC, see for instance fig. 14.

Concerning now the minimum and maximum cardinalities, even values are advised, especially with quads/hexas in order to try to keep a shape similar to the original one in any case. The cardinalities should at least be equal to 2: this does not leave a wide choice for the lower bound in 2D; in 3D, 4 could be a reasonable value. As for the upper bounds, a reasonable value could be to choose it so that the allowed interval is symmetric with respect to the goal cardinality.

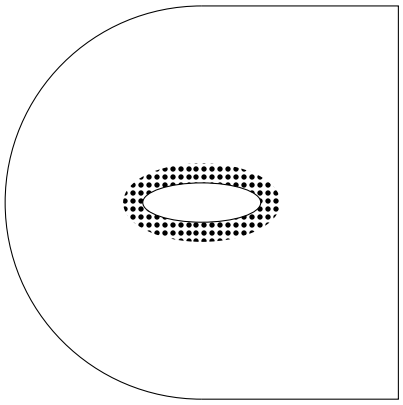
The user may choose to force a certain cardinality by setting minimum, goal, and maximum to the same value. This choice might lead to very good results on some meshes (perfect structured behaviour), but it clearly constrains the algorithm (for instance, forcing a CC to grow even though that would mean worsen the AR) and might lead to undesirable results when the CC lacks space to grow.

6.1.3 Final CC numbering

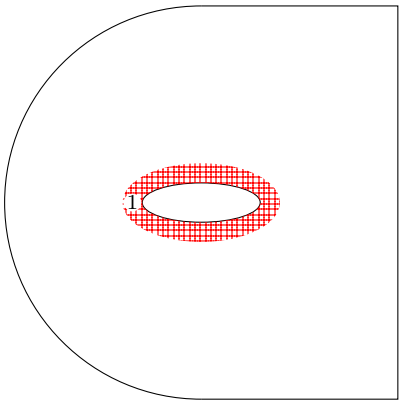
The IDs of the CC just reflect the order in which they have been created. This order is deterministic and depends on whether the anisotropic treatment is activated (the anisotropic agglomeration being done always in the first stages of the algorithm, anisotropic CC have smaller IDs than isotropic ones), the neighbourhood and seeds pool type, the priority weights, and whether the correction is activated.

6.1.4 Working with a boundary layer

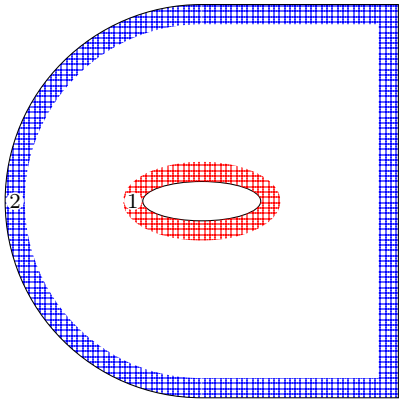
When working with a boundary layer (BL), most of the times one activates the anisotropic agglomeration, which, desirably, should be applied to the BL only. This might not be always the case when passing to CoMMA the full list of FC as being anisotropic compliant (see section 6.2.1 and fig. 15 below for more details). Whenever possible and easily achievable, it is advised to fill the input argument of compliant anisotropic cells, `anisotropicCompliantCells`, only with cells inside the BL or verifying other relevant criteria: for instance, users calling CoMMA from



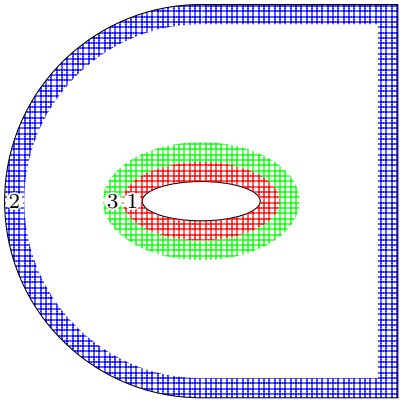
(a) Initial state: mesh with BL (dotted)



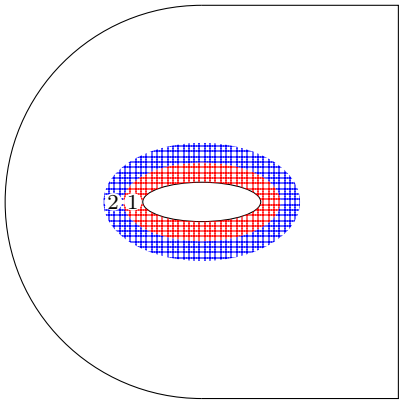
(b) Anisotropic agglomeration



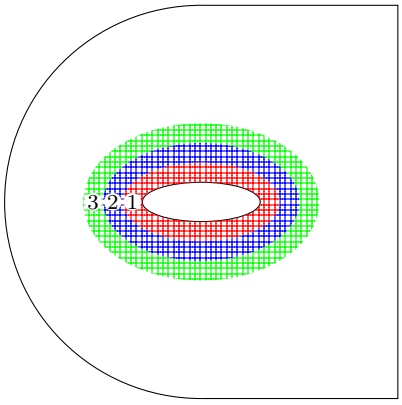
(c) Priority by **boundary**: first stages of isotropic agglomeration



(d) Priority by **boundary**: late stages of isotropic agglomeration



(e) Priority by **neighbourhood**: first stages of isotropic agglomeration



(f) Priority by **neighbourhood**: late stages of isotropic agglomeration

Figure 13: Typical example of agglomeration, the mesh is the same as fig. 5. First row: original mesh with an anisotropic BL (left), and state after the anisotropic agglomeration (right). Second row: steps with a seeds pool with a priority by boundary. Third row: steps with a seeds pool with a priority by neighbourhood.

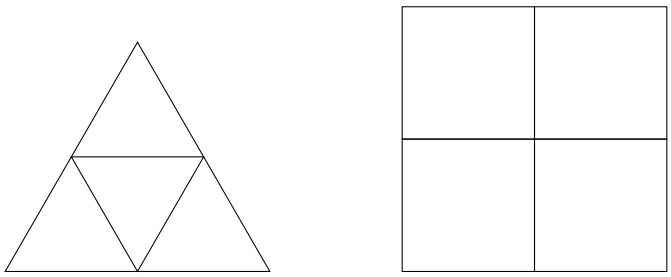


Figure 14: A cardinality of 4 in 2D should recover structured CC from structured FC. Left: case with triangles; right: case with squares.

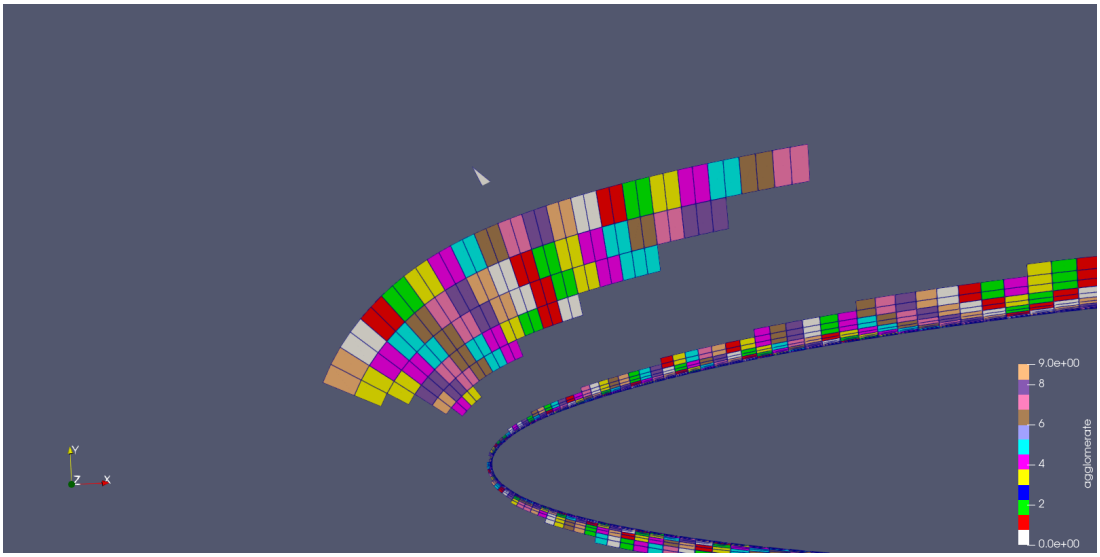


Figure 15: Result of an anisotropic agglomeration on a 2D RAE mesh - zoom on the zone of the BL at the leading edge of the airfoil. Only anisotropic cells are depicted. The (fine) cells are colour-coded according to the CC to which they belong. We can identify 3 pitfalls: isolated cell, anisotropic zone not spanning the whole BL, different anisotropic lines lengths and directions.

a CFD solver have used the wall distance as criterion. Another parameter to consider and tune is the anisotropy threshold, value that determines whether a cell is anisotropic or not, recall (1).

6.2 Known drawbacks

6.2.1 Anisotropic agglomeration and BL

Several common pitfalls might occur when working with anisotropic agglomeration and with a mesh with a BL, some of them are depicted in fig. 15. We describe them briefly just below.

If the anisotropy threshold is too high, only a sub-set of the BL is considered anisotropic: indeed, in fig. 15 (which shows only the anisotropic cells) large portions of the BL are missing, meaning that they are considered as being isotropic. On the other hand, CoMMA might still consider cells outside the BL as being anisotropic: indeed, see the white isolated triangular cell in the middle of the picture. It is worth mentioning that, in this particular case, since the cell is isolated, CoMMA fails to compute an anisotropic line, hence the cell will be treated with the standard isotropic agglomeration algorithm.

According to the mesh and the value chosen for the anisotropic threshold, anisotropic lines close to each other might have different lengths. Take, for instance, the third and fourth lines on the right part of fig. 15. To identify them, first notice, that, here at the very neighbourhood

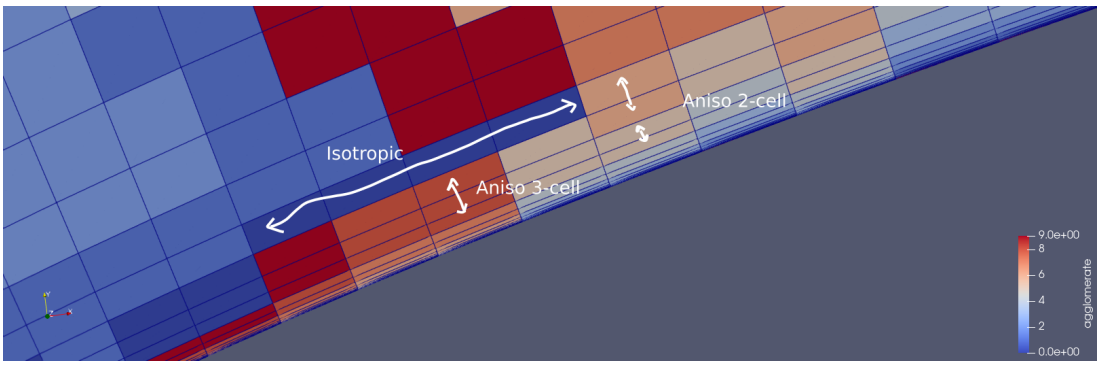


Figure 16: Result of an anisotropic agglomeration on a 2D RAE mesh - zoom on the zone of the BL at the leading edge of the airfoil. The (fine) cells are colour-coded according to the CC to which they belong. The lack of space constrain isotropic CC inside the BL to grow in undesirable shapes.

of the airfoil, a line grows perpendicularly with respect to the airfoil, that is, vertically in the current picture, hence we can think of a line as a "column". Then, the third line is the one whose top three cells are depicted in yellow, while the two top cells of the fourth line are in purple. It is clear that the fourth line is shorter, whereas one would wish for the anisotropic zone to be compact and/or convex.

The previous two remarks combine in an unfortunate situation when the agglomeration of isotropic cells is considered. Indeed, recall that the anisotropic agglomeration is performed always before the isotropic one. Hence, even though the nature of the FC of the BL (structured quads) should often lead to structured CC, this might not happen since the isotropic CC are constrained to grow in undesirable shapes, see fig. 16.

Finally, two anisotropic zones are clearly visible in fig. 15. By looking at the colors and, hence, at the CC, in the bottom zone, anisotropic lines grow vertically (recall the remark above), whereas in the top zone they grow horizontally. If several agglomeration levels are performed and since the anisotropic agglomeration always relies on those lines identified on the finest level, the difference in the growth direction of the lines can lead to undesirable results.

Some improvements that might fix these drawbacks are suggested in section 6.3. However, on the user side, one possible way to fix most of these issues is, as it has been suggested in the previous section, to pass a limited number of cells as anisotropic compliant (for instance, only the cells inside the BL or those whose wall distance is less than a given threshold) and select a small anisotropy threshold.

6.2.2 Anisotropic lines

Building agglomeration lines works fine in 2D: only a few issues, see for instance fig. 8, have been identified.

Some problems have been observed in 3D: in particular, the algorithm generates an "**infinite line**" or, better, only one line that joins all the anisotropic cells. Consider fig. 17. Let the coloured path be a line whose current tail is cell A. One would expect the next cell to be included in the line would be cell B since it has the same direction as the line. However, the interfaces A/B and A/C are the same (A has a square basis), hence it may happen that cell C is included instead of B. Consequently, the line bends. Imagine accumulating several turns like this and the same unique line may span all the anisotropic zone.

Suppose now that the situation of fig. 17 (cells with squared basis) happens at the border. One might start in a direction parallel to the border, which is undesirable.

Issues may rise also with triangular/tetrahedral meshes, since the definition itself of agglom-

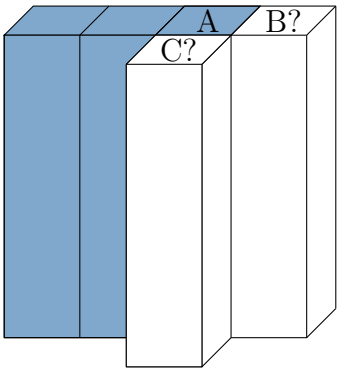


Figure 17: Anisotropic cells in 3D and an agglomeration line, in blue

eration line might not be so straightforward.

A way to avoid the above-mentioned drawbacks is discussed in section 6.3.

6.2.3 Seeds pool ordering

We give here two behaviours related to the choice of the seeds pool order which the user should be aware of.

In the late stages of the agglomeration, not-yet-agglomerated FC are very few and new CC might be constrained to grow in undesirable shapes due to lack of space. The order of the seeds pool has an important impact on where these issues start appearing, and one should have that in mind.

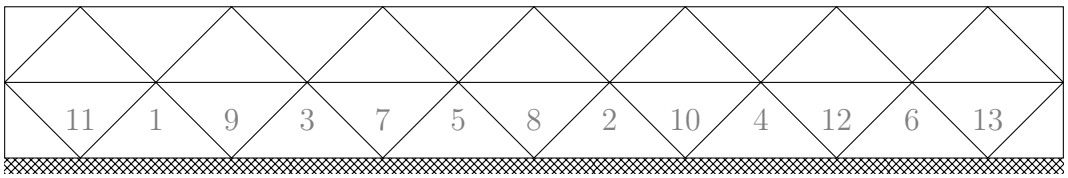
Considering again the mesh of figs. 5 and 6, we have seen that the boundary priority generated two agglomeration fronts, one advancing inwards and the other outwards. The two fronts meet inside the mesh, somewhere between the inner and outer borders. So, if undesirable CC appear, they would be located inside the mesh. On the other hand, with the neighbourhood priority only one front is formed, starting to the outer border and moving inwards. In this case, undesirable CC might appear at the inner boundary, just on the airfoil, which is of course an undesirable behaviour.

With no doubt, the example above highly depends on the mesh, but still depicts the consequences (possibly very negative) of the seeds pool ordering.

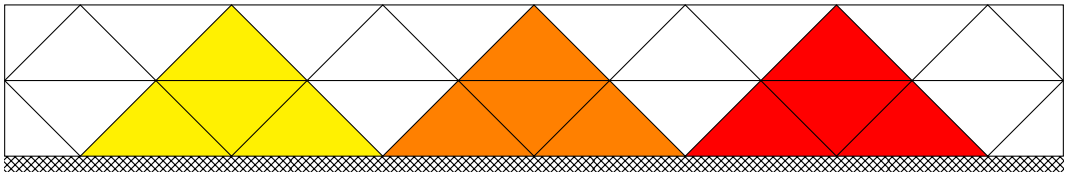
Consider the triangular mesh of fig. 18. The boundary is depicted with a crosshatch pattern. The IDs of some boundary cells are given in the top row of the figure. Suppose also that every cell has the same priority weight. In such a regular mesh, one would expect a result as the one in fig. 18b. However, let us now consider different strategies for handling of the seeds pool.

- **Boundary priority and full initialization** and **neighbourhood priority and full initialization.** Having requested a full initialization, the initial seeds pool contains cells 1, 2, 3, 4, and 5, in this order. The algorithm starts from cell 1, creates the yellow CC in fig. 18c, then moves to cell 2, creates the orange one, and finally creates the red one, which features an undesirable shape. Here, two adjacent cells do not always have consecutive ID numbers, which, having chosen a full initialization, resulted in a hole being left and then badly filled (red cell).

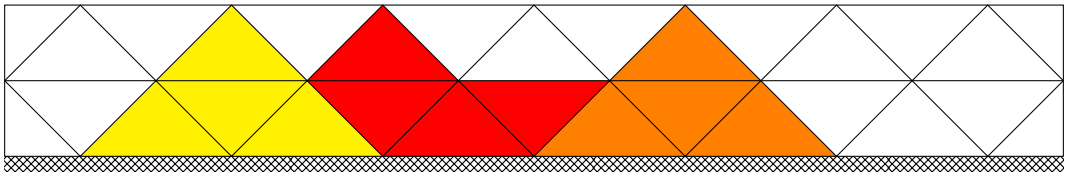
- **Boundary priority and point initialization.** The issues of the previous point seemed to be caused by the initialization. Hence, we consider now a point one. We retain the boundary priority. We start from cell 1, create again the yellow CC, and add all its neighbours (cells 7, 11,...) to the seeds pool: notice that they are all interior cells. At this point, there are still boundary cells not yet agglomerated but they are not in the



(a) Mesh and cell IDs



(b) Expected result



(c) Actual result

Figure 18: Unexpected behaviour at the boundary of a triangular mesh.

seeds pool, hence, having chosen the priority by boundary, we rebuild the seeds pool to put the boundary cells on top. One now ends up in the same situation as before, hence agglomerating the orange cell and finally then the red one.

- **Neighbourhood priority and point initialization.** The issue in the previous strategy was that the priority chosen forced us to rebuild the seeds pool. We now change it and use the neighbourhood priority. The first step is the same as before: create the yellow cell and add its neighbours, cells 7 and 11 (and other not mentioned for brevity sake), to the seeds pool. The second seed is cell 7, hence we still end up with something similar to the red cell.

6.2.4 Triangular meshes

CoMMA does not always return the expected agglomeration when working on triangular meshes. The case presented in section 6.2.3 is clearly an example. Moreover, notice that if the first stages of the agglomeration yield a complex, undesirable pattern as the one in fig. 18c, it will be extremely difficult for the later agglomeration stages to recover a more smooth shape.

6.2.5 Known bugs

We give below the list of known bugs.

- The formula used for computing the number of (exterior) faces of a CC gives wrong results. Taking the examples in fig. 19, for a squared CC it gives 4, as it should be; for an L-shaped one, 5, instead of 6; and for a deformed squared one, 4 again. The problems here are two: the formula used itself is not good enough, but more importantly, there is no way with the minimal information at CoMMA's disposal to tell whether two faces

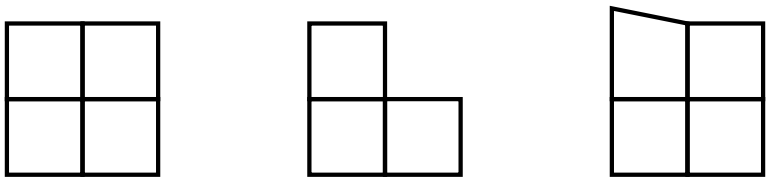


Figure 19: CC for which the formula for computing the number of (external) faces might be wrong.

should be merged together, as in the case of the perfect square, or not, as in the case of the deformed one. All in all, it still does a decent enough job since the value of the L-shaped CC is still greater than the value of the squared ones, hence, these latter would be preferred (see in section 3.4 how the choice among the intermediate CC is performed).

6.3 Possible improvements

Several improvements have been identified, whether by trying to fix an undesirable or unexpected behaviour (see section 6.2) or suggested by users. We gather them here in the hope that this document might be used as a plan for future works.

6.3.1 Boundary treatment

Whether a cell is at a boundary or not plays an import role in determining its positions in the seeds pool, see criterion SPO.BF. However, not all the boundary are the same. Some boundaries might actually be only virtual ones, because they have been created when partitioning the mesh, or because they are associated to overset, periodic or symmetry boundary conditions. In these cases, it would seem logical not to give them high priority, since their cells might be thought of interior ones. Indeed, in the literature (e.g., Mavriplis), only walls and far-field boundaries are considered.

A simple workaround which would implement the above-mentioned treatment, would consist, at the user side, to tweak CoMMA's input argument `n_bnd_faces` and set to 0 the entries related to the cells on those type of boundaries; this, however, might lead to some miscalculations, since CoMMA won't be able to understand the nature of the cell (the boundary faces are used with the others to tell the shape of a FC). Another, cleaner, solution would be to assign very high priority weights to walls and such, but the weights have lower priority than the number of boundary faces, hence it might not lead to the desired results.

One option would be to take as input parameter a more detailed vision of the boundary, in particular, telling to which type of boundary (e.g., periodic, wall,...) each cell belongs to and the priority to give to each boundary type (e.g., no priority to periodic, maximum priority to walls).

Finally, one could also consider to let the user provide a set of cell (this set being a boundary or not) from which the agglomeration should start and which should override all the other seeds pool-related criteria.

6.3.2 Improve anisotropic lines

In the previous section we have seen that in some cases, and especially in 3D, it might be difficult to correctly computing an anisotropic line. One idea would be to take advantage of the centres of the cells to keep track of the growth direction of the line and accept a new cell only if the resulting growth direction "does not deviate" from the previous one.

We briefly explain how this would work. If the line in construction has at least two FC, we compute the current direction of the line as the normalized vector joining the centers of the second-to-last and last added cells. In addition to criteria AnC1-AnC4, we also consider:

AnC5 The direction given by the centers of the considered cell center and the last added one is parallel to the current direction.

This criterion will typically consists in checking the dot product of the two vectors is above a reference threshold.

The strategy of the construction of an anisotropic line is further modified. If the search with criteria AnC1-AnC5 does not identify any candidates, another search is performed this time where criterion AnC4 is relaxed: if this leads to at least one candidate, one new cell is added and the usual procedure begin.

The resulting set of criteria after dropping AnC4 prioritize the direction of the line over the connection with maximum weight.

This feature has been introduced in the newest versions of CoMMA, see appendix A.

6.3.3 Choice of the best next FC

It might be useful to add the compactness degree among the criteria for the choice of the best FC, see section 3.4.1. One might argue that it should even be the most important one. In some sense, however, the compactness degree is related to the number of shared faces, BFC.1, which is already the criterion with the highest priority.

6.3.4 Improve the AR computation

In section 3.4.1, we have seen that the choice of the formula for computing the AR (1) does not always lead to the desired results. A possible modification could be:

$$\widehat{\mathrm{AR}}_{cc} = \frac{\max\limits_{fc_i, fc_j \in cc} d\left(c_{fc_i}, c_{fc_j}\right)}{\min\limits_{fc_i, fc_j \in cc} d\left(c_{fc_i}, c_{fc_j}\right)} \tag{5}$$

where cc is the CC, $\{fc_i\}_{cc}$ is the set of the FC in cc, c_{fc_i} is the centre of fc_i, and $d(\cdot,\cdot)$ is the Euclidean distance. Notice that eq. (5) would apply both to 2- and 3D problems. The main reason for choosing (5) is that it looks more similar than (3) to the reference definition of AR that we consider, that is, (2): in fact, the only difference is that in (5) one uses the distances between the centers of the cells rather then the vertices.

Notice, however, that (5) makes sense only for CC with cardinality of at least 3; for CC of cardinality 2, (3) should still be used.

Experiments have already been conducted with (5). On the case proposed in fig. 10, this new formula still prefers the `I`-shaped CC rather then the desired `L`-shaped one. The reason is that in both cases the minimum would be the same, that is, the distance of the centers of two adjacent cells. Indeed, the main drawbacks of (5) is the computation of the minimum which might be an unpractical value to compute for CC with low cardinalities (and "simple" shapes).

6.3.5 Further extend neighbourhood

In section 3.2.4, all the direct neighbours of a CC are added to the seeds pool. One might choose to include also higher-order neighbours of the CC (and not only the direct ones). This choice, for instance, might turn out to be useful in case like the agglomeration presented in section 6.2.3 and fig. 18. Indeed, with a point initialization, after the first CC, the seeds pool

will contain the boundary cell closest to the first CC, which is a second order neighbour, at the top of the seeds pool. Starting from there, the expected saw-tooth pattern of fig. 18b might be returned. A similar strategy was part of the original version of CoMMA, where all the cells of the allowed maximum-order neighbourhood of the seed were added to the pool.

6.3.6 Unexplored options

Some options that were present in the original version of CoMMA are still in the code but they are not used and might be worth exploring:

- **Choice of FC by order.** In CoMMA's code one can find a switch often call `is_order_primary` that acts on the priorities for the choice of the best FC, see section 3.4.1. In particular, if the switch is activated, CoMMA bypasses the criterion BFC.1 about the number of shared faces, hence CoMMA would consider the neighbourhood order, criterion BFC.2, whence the name of the switch. This option applies to isotropic agglomeration only.

- **Delayed creation.** When agglomerating, by activating a switch called `is_delayed` (or similarly), a CC whose compactness degree is less than the dimension (2- 3D) is only created *virtually*. This means that the FC are not tagged as being agglomerated, on the other hand, the seeds pool would look just as if the CC has been actually agglomerated. This option applies to isotropic agglomeration only.

- **Feature preserving.** This choice acts on how CoMMA's tags anisotropic cells. The target use is the BL: it is usually composed of quadrangular / hexahedral cells. Hence, a cell in order to be tag as anisotropic not only it has to have a large enough AR but it also has to be of one of the two above-mentioned shapes (the shape is estimated by looking at the number of neighbours and/or boundary faces).

- **Triconnected graph.** In a old version of CoMMA, there was an experimental isotropic agglomerator that was supposed to work on triconnected graphs, that is, graphs where every vertex (the mesh-equivalent of a cell) has at least three neighbours; this could have been used in 3D meshes. The agglomerator had some problems and never actually made to the release. This agglomerator is not in the current code of CoMMA.

6.3.7 Parallelization and domain decomposition

One of the main drawbacks of CoMMA is that it works only sequentially per domain. This means that, if a mesh is divided by a partitioner into several domains, the global result highly depends on the number of the domains and their shapes. A great improvement would be to make CoMMA be able to communicates with all the domains, hence being able, for instance, to create CC that span over more than one domain, so that the final global results is deterministic and does not depend on the number of the domains.

Mind that this proposed improvement would surely require a complete rewriting of the code or at least a deep refactorization of the code.

References

Mavriplis, D. J. (1997). "Directional coarsening and smoothing for anisotropic navier-stokes problems". In: *Electron. Trans. Numer. Anal.* 6, pp. 182–197 (cit. on p. 24).

Mavriplis, D. J. (1999). "Directional agglomeration multigrid techniques for high-Reynolds-number viscous flows". In: *AIAA J.* 37.10, pp. 1222–1230 (cit. on p. 24).

Mavriplis, D. J. and V. Venkatakrishnan (1995). "Agglomeration multigrid for two-dimensional viscous flows". In: *Comput. Fluids* 24.5, pp. 553–570 (cit. on p. 24).

Mavriplis, D. J. and V. Venkatakrishnan (1996). "A 3D agglomeration multigrid solver for the Reynolds-averaged Navier-Stokes equations on unstructured meshes". In: *Int. J. Numer. Methods Fluids* 23.6 (cit. on p. 24).

Milani, R. (Nov. 2023). *CoMMA, a geometric unstructured agglomerator*. Tech. rep. RT 7/30485. ONERA (cit. on p. 1).

SALOME MESH-module online documentation (n.d.). `https://docs.salome-platform.org/8/gui/SMESH/aspect_ratio.html`. Accessed: 2023-04-19 (cit. on p. 44).

SimScale online documentation (n.d.). `https://www.simscale.com/docs/simulation-setup/meshing/mesh-quality/`. Accessed: 2023-04-19 (cit. on p. 44).

A Latest developments

Since the release of CoMMA `v1.1`, several developments have been introduced to CoMMA. Furthermore, they have been gathered in a new release, `v1.2`, identified by commit `8da26152`. We describe below the most salient developments of the newest release.

A.1 Conservation of the growth direction for anisotropic lines computation

The improvement proposed in section 6.3 about taking into account the direction during the computation of the anisotropic lines is now part of the reference version of CoMMA (see in particular commit `7c97401b`).

For the sake of clarity, we detail here how the algorithm for the choice of the next cell to add to the anisotropic line works. We suppose that the line is already composed of at least 2 cells, so that it is possible to compute a reference direction by using the centres of the two last cells added to the line.

1. Among all the neighbours of the last added cell, one identifies the candidates, that is, those verifying **all** of the following criteria

AnC1 It's anisotropic,

AnC2 It does not belong yet to an anisotropic lines,

AnC3 It's a neighbour of the current head/tail of the line, and

AnC4 It's connected to the current head/tail through the side with highest weight (that is, face with largest surface) with respect to the head/tail.

AnC5 The direction given by the centers of the considered cell center and the last added one is parallel to the current direction.

2. Two results are possible:

   - If one or more candidates have been found, then take one (the one yielding the "most parallel" direction, practically, the one yielding the highest absolute value of the dot product of the two directions) and add it to the line.
   - If no candidates have been identified, relaunch a search with the criteria AnC1, AnC2, AnC3, AnC5 (no AnC4). If one or more candidates are found, then take one and add it to the line.

The algorithms for the computation of the line ends when even the second search fails to find candidates.

First tests are promising and, on the same configuration as the one in fig. 15, one recovers only one direction (orthogonal to the airfoil) in the whole BL, instead of the two directions of the standard procedure. The results on a 2D RAE foil are given fig. 20: the lines are clearly visible and do not turn. We provide in fig. 21 a comparison on a 3D cylinder (3D case were critical for the anisotropic line computation with the original algorithm): with the original method, the line had turns and grew, at least for some time, parallel to the wall; with the improved strategy, the line is now straight.

It is important to notice that this latest improvement basically forbids the lines to have turns. Hence, it is of paramount important to start with the right direction. The second cell, that is, the first one added after the seed, is added without comparing a reference direction (it is not available yet!), hence it is chosen principally on the bases of the largest connection,

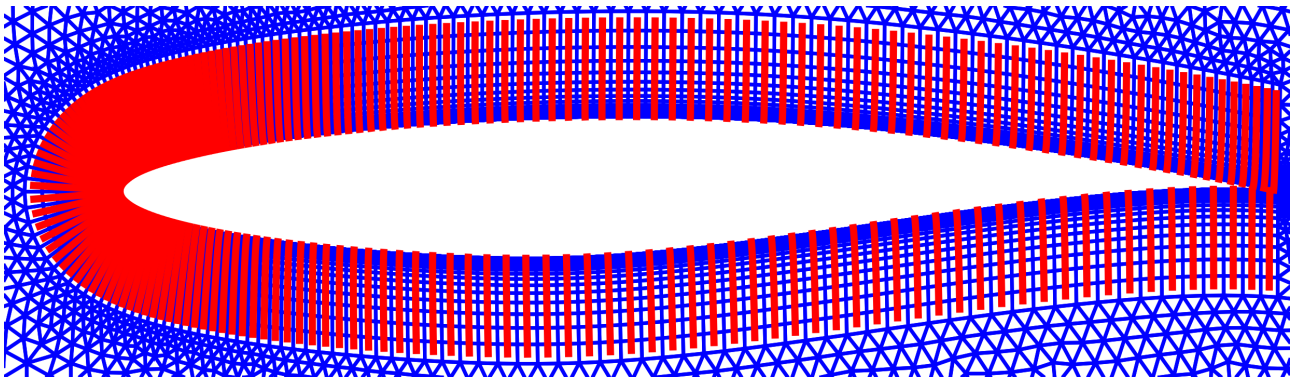


Figure 20: Anisotropic lines (red) computed with additional directional criterion, AnC5. Compare with fig. 15 where two line directions were found (another difference is that, playing with a negative threshold, all the cells of the BL were considered as anisotropic in the current case instead of only a subset in the reference one).

AnC4. If this first added cell yields a direction parallel to the wall, there is no way to fix this. The cells in the BL closer the wall are usually very thin in the direction orthogonal to the wall, hence, starting from there should avoid direction issues: for this reason, it is advised to give high priority weights to the cells on the wall, so that there are greater chances for the agglomeration to start there.

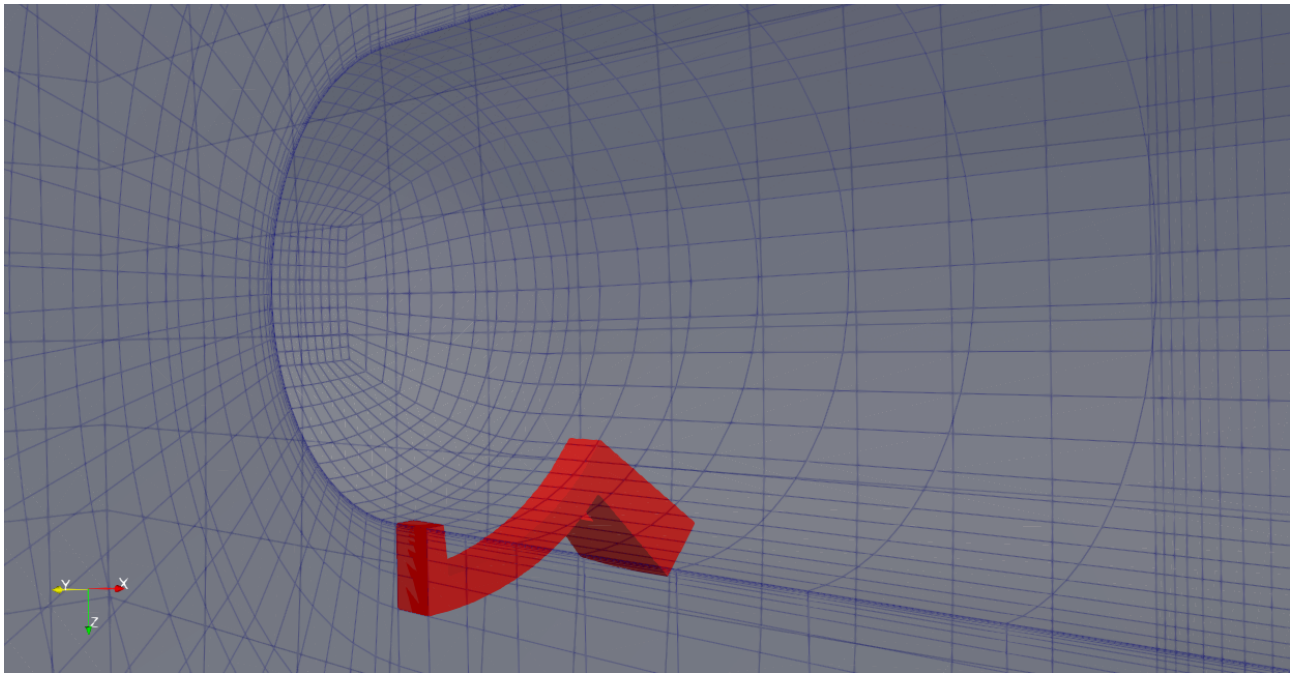
The algorithm fails to build the correct lines in the case of a triangle/tetra meshes since the growth direction might change a lot from one cell to the other. However, in the case, for instance, of a quad/hexa BL cut into triangles/tetras, the agglomeration results should be generally good. See an example in fig. 22. One would expect vertical anisotropic lines, however, the algorithm only cells two-by-two, then, it cannot add any more cell. The result are anisotropic lines of length 2, see red lines in the picture. The two cells in the line will be agglomerated together, hence (re)creating nice quads. If a second agglomeration level would to be called, CoMMA would be able to correctly build the lines working with the quads, but one should force the building by passing `isFirstAgglomeration` equal to true even if it is not.

A.2 Disable 3-cells agglomerates in anisotropic agglomeration

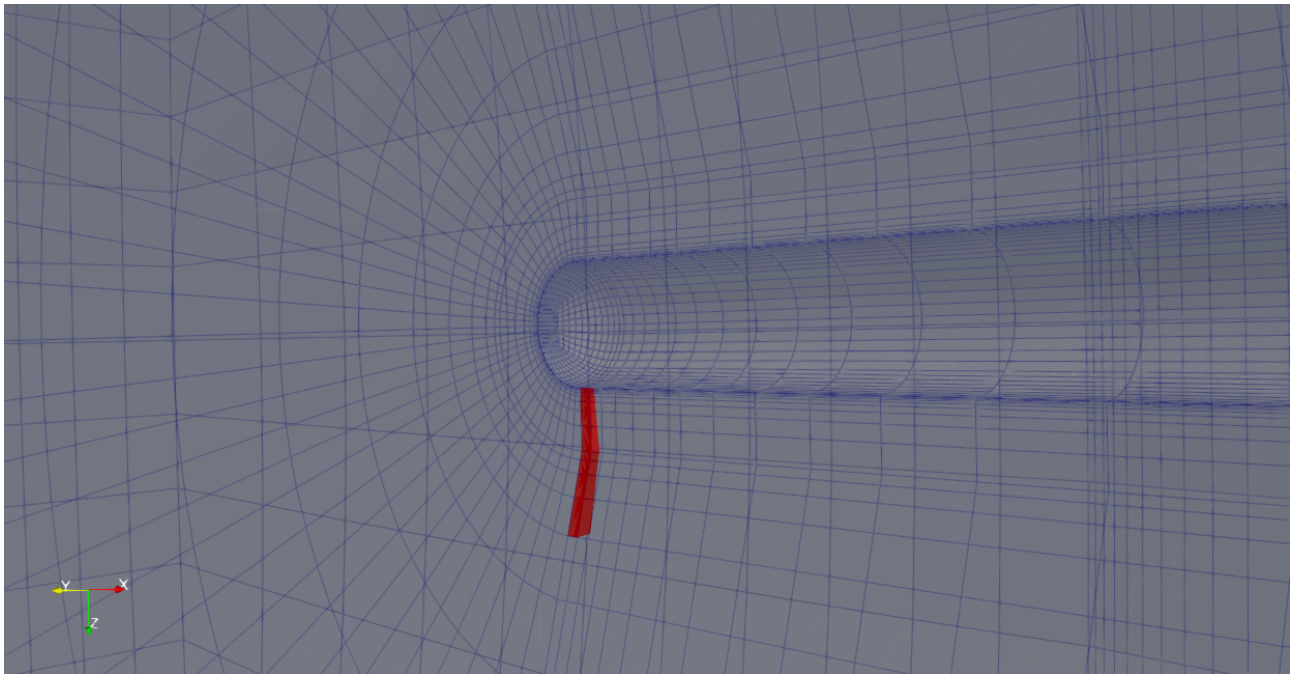
The tagging of anisotropic cells done by CoMMA itself often leads to a stair-like pattern, see fig. 23 for a typical result in 2D. Indeed, several steps 1-cell high are created. This pattern then constrains the following stages of the agglomeration. Indeed, typically, in 2D, with a goal cardinality of 4 in structured zones composed of quads like in the BL of the figure, isotropic CC are quads of 2-cells-long edges. This value of the edge (2-cells) does not couple well with the *height* of the steps (1-cell), hence the stair-like pattern is entertained.

A stair-like pattern with steps 2-cells high can be smoothed out by the isotropic CC generated in the first stages of the isotropic agglomeration, hence producing more convex agglomeration zone and recovering a certain structure to the coarse mesh. Mind that this will not completely fix the problem of the convexity of the anisotropic zone: indeed, steps are still create, yet their number is smaller but their height is higher; the presence of steps might still lead to undesirable results, especially if several agglomeration levels are requested.

Forcing 2-cells-high steps can be achieved by forbidding 3-cells agglomerates that are sometimes created when the anisotropic lines have an odd length. A new input parameter has been introduced by commit `1a3e6156` to control this behaviour: setting the boolean `odd_line_length` to true allows CoMMA to have 3-cells agglomerates, a false value forbids it.



(a) Original strategy



(b) New strategy

Figure 21: Comparison of an anisotropic line computed around a 3D cylinder between the original (top) and improved (bottom) strategy.

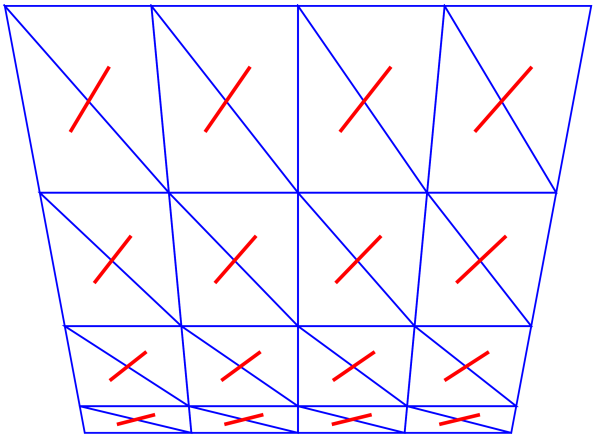
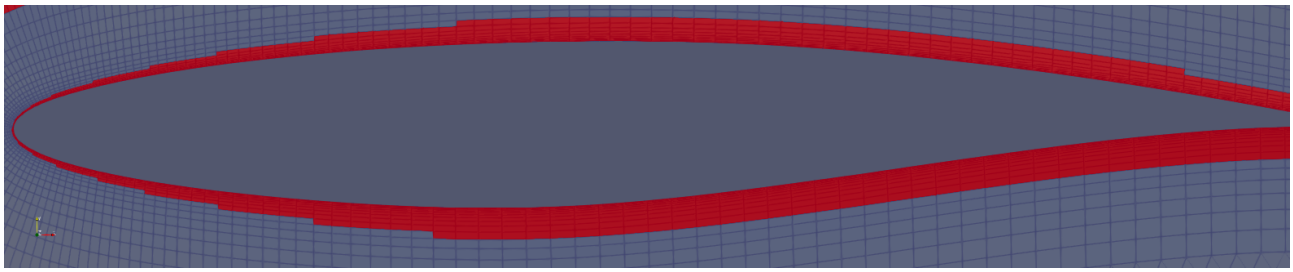
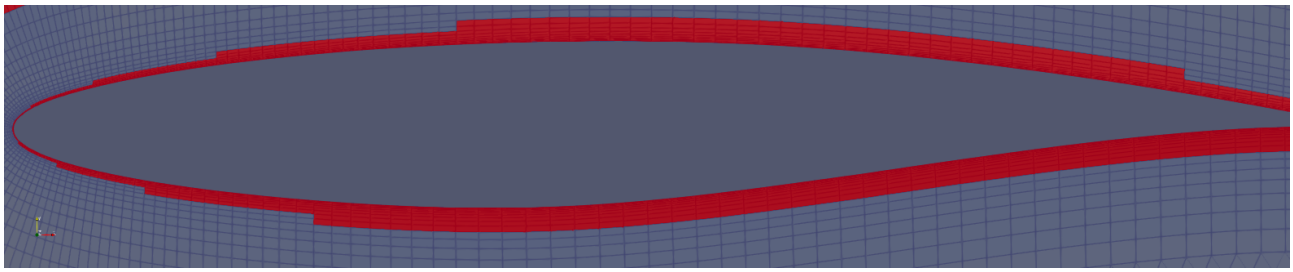


Figure 22: Results of the agglomeration lines (red) detection on a 2D structured BL cut into triangles.



(a) 3-cells agglomerate allowed



(b) 3-cells agglomerate *not* allowed

Figure 23: Typical result of an anisotropic tagging over a BL of a 2D RAE mesh where 3-cells agglomerates are allowed (top) or not (bottom).

A.3 Performance improvements

Thanks to simple profiling, we could identify a few bottlenecks which significantly impacted the performances:

- Building of the anisotropic lines,
- Corrections stage,
- Building of the CC-related subgraph.

In particular, the first two items significantly slowed down the computations: the structure of the two procedure has been revisited to improve the computational times. Concerning the latter, the CSR graph of a CC was indeed build but not used, or better, not exploited. Hence, we now avoid building it (with a saving that could amount up to 10%) and rely only on some global features of the CC, like the cardinality or the compactness degree.

A.3.1 Time comparisons

We report below some comparisons of the computational times for the CoMMA versions before and after the performance improvements.

The baseline version of CoMMA (without the improvements) is identified by commit `78ef75d5`, the improved one by commit `41e888f`.

Cases description Three mesh have been considered:

- **NACA0012**: 2D, no BL, only quadrangular cells. With this mesh, only isotropic agglomeration have been considered.
- **RAE2822**: 2D, with BL, mix of quadrangular (in the BL) and triangular cells, see for instance fig. 20 for a zoom on the BL. With this mesh, anisotropic agglomeration is activated, moreover, only the cells closer to the airfoil (in particular, with $x_C > -0.01$ and $\|\underline{x}_C\| < 1.1$, where $\underline{x}_C$ is the center of the cell) were considered as anisotropic compliant, and high priority weights were given to the boundary cells.
- **Hemisphere and Cylinder (HSC)**: 3D, with BL, only hexa cells; see the online description and an extract in fig. 21. With this mesh, the anisotropic agglomeration is activated. A "wall", that is, the boundary cells on the hemisphere and the cylinder, is identified, the neighbourhood-based distance (cf. neighbourhood order) is computed, and only cells with distance less than 20 are retained as anisotropic compliant. The priority weight values are the inverse of the distance.

We considered three parameter configurations:

- Fixed cardinality (meaning the minimum, goal, and maximum cardinality were set to the same value) and correction stage disabled: this case enable one to evaluate to performance of the baseline algorithm;
- Flexible cardinality and correction stage disabled: by comparing the results of this test with those of the previous, one can evaluate how the performances change with flexible cardinality;
- Fixed cardinality and correction stage enabled: by comparing the results of this test with those of the first item, one can evaluate the performances of the correction stage.

Table 1: Description of the test cases for the performance comparisons.

Case	N cells	nD	Anisotropic	Min card	Goal card	Max card	Corrections
NACA.1	7168	2D	No	4	4	4	No
NACA.2				2	4	6	No
NACA.3				4	4	4	Yes
RAE.1	22842	2D	Yes	4	4	4	No
RAE.2				2	4	6	No
RAE.3				4	4	4	Yes
HSC.1	23688	3D	Yes	8	8	8	No
HSC.2				4	8	12	No
HSC.3				8	8	8	Yes

Other significant parameters worth mentioning were that the neighbourhood priority is considered for all cases, and, if anisotropic agglomeration is enabled, a threshold of 2 is considered.

For each mesh all the three configurations were applied, so that a total of 9 cases have been considered. They are summarised in table 1.

Remark. It is very rare to have singular CC with quadrangular or hexa meshes, hence we expect that no corrections are needed for the meshes NACA and HSC and, consequently, similar results for case NACA.1 and NACA.3, and HSC.1 and HSC.3.

A.3.2 Test results

Each test case has been run 50 times on the same machine (an ONERA personal computer). Wallclock times have been recorder simply using the `chrono C++` library. The results are reported in tables 2 to 4. The improvement are visible and amount up to more than 6 times faster than the baseline case.

A.3.3 Scaling tests

Starting from the RAE mesh, several refinements have been applied in order to test how CoMMA scales. As before, each case has been run 50 times, the average times are reported in fig. 24: notice that the scales of the two plots are different. This proves that new version has superior scaling.

A.4 Improve structure of the anisotropic agglomeration

In some cases, the anisotropic agglomeration might be switched off:

- If usage of user-defined lines is requested (`isFirstAgglomeration` set to false) and at least one of the following items is true:

  1a. no lines provided as argument (e.g., empty `agglomerationLines[_Idx]`)

  1b. all the provided lines are composed of just one cell

- If anisotropic lines building is requested (`isFirstAgglomeration` set to true) and at least one of the following items is true:

  2a. no anisotropic compliant cells provided

  2b. no cell was recognized as anisotropic

  2c. only isolated anisotropic cells

Table 2: Results of the performance comparisons on the NACA mesh. Times in ms. See table 1 for case definition.

(a) Case NACA.1

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	17	8	
Max	19	11	
Avg ± std	17.14±0.45	9.00±0.35	0.53

(b) Case NACA.2

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	33	17	
Max	39	19	
Avg ± std	33.28±0.90	17.06±0.31	0.51

(c) Case NACA.3

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	17	8	
Max	18	11	
Avg ± std	17.06±0.24	8.92±0.44	0.52

Table 3: Results of the performance comparisons on the RAE mesh. Times in ms. See table 1 for case definition.

(a) Case RAE.1

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	65	28	
Max	71	31	
Avg ± std	67.14±1.51	28.48±1.37	0.42

(b) Case RAE.2

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	92	46	
Max	105	51	
Avg ± std	98.32±1.72	47.64±1.37	0.48

(c) Case RAE.3

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	194	30	
Max	204	36	
Avg ± std	196.3±2.02	30.82±0.94	0.16

Table 4: Results of the performance comparisons on the HSC mesh. Times in ms. See table 1 for case definition.

(a) Case HSC.1

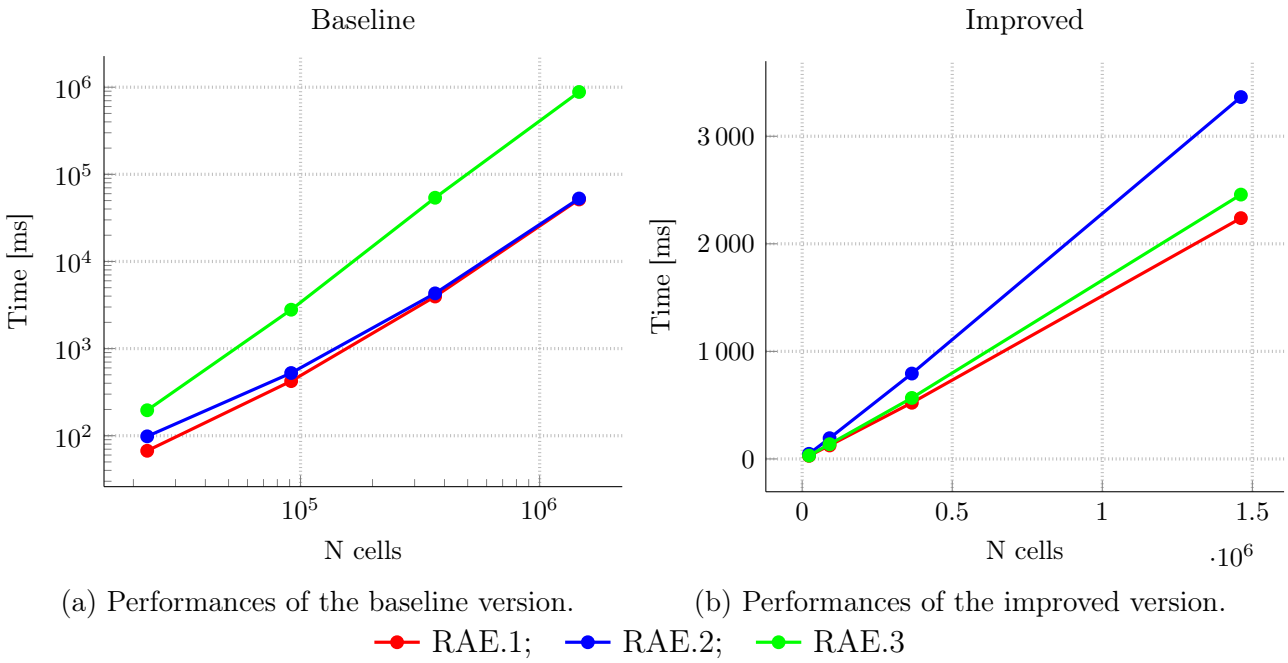
	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	503	64	
Max	517	70	
Avg ± std	509.8±3.91	65.58±1.42	0.13

(b) Case HSC.2

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	660	121	
Max	673	126	
Avg ± std	664.6±2.14	122.8±1.42	0.18

(c) Case HSC.3

	Baseline	Improved	$\frac{\text{Improved}}{\text{Baseline}}$
Min	512	66	
Max	518	74	
Avg ± std	514.8±1.61	69.44±1.93	0.13



(a) Performances of the baseline version.

(b) Performances of the improved version.

Figure 24: Performances of the baseline (left) and improved version (right) on the RAE mesh series. Notice that logarithmic scaling is activated on the left but not on the right. Times in ms. See table 1 for case definition.

Notice the several of the previous items might happen when considering a mesh split into several partitions but only a subset of the partitions holds anisotropic cells/lines.

Moreover, lines provided either by the user or by CoMMA building tool and composed of just one cells will be discarded and their cell thus considered isotropic. Let us consider an example to better understand the consequences. A first agglomeration stage builds the lines, these lines are used in the following agglomeration stages, meaning that the original anisotropic zone will always undergo an anisotropic treatment. After several agglomerations, it might happen that a line becomes composed of only one big cell. This typically results in triggering condition 1b.: the related line would thus been put back into the pool of isotropic cells, hence effectively breaking the special anisotropic treatment.

B A note on the aspect ratio

The aspect ratio (AR) of a cell is a quantity of paramount importance in CoMMA. In particular, it is used in two main occasion: in a preprocessing stage to identify the anisotropic cells, see section 3.3 and (1) in particular, and during the isotropic agglomeration to choose the best possible coarse cell, see section 3.4 and (3). We give here some insights about the two different ways used to compute this variable.

B.1 Reference aspect ratio definition

The reference definition of AR for a cell to which we aim is (recalled from section 3.3):

$$\mathrm{AR}_{\mathrm{ref}} \coloneqq \frac{\max\limits_{i,j,i\neq j} d\left(v_i, v_j\right)}{\min\limits_{i,j,i\neq j} d\left(v_i, v_j\right)} \tag{2}$$

where $\{v_i\}$ are the vertices of the cell and $d(\cdot,\cdot)$ the Euclidean distance. Holding only the graph representation of the mesh, one does not have any knowledge of vertices within CoMMA.

Other CFD solvers and meshing tools consider a slightly versions of the AR. See for instance SALOME *SALOME MESH-module online documentation* n.d. or SimScale *SimScale online documentation* n.d. Notice as both use different definitions of the AR for triangles/tetras and quadrangles/hexas. We report below the definitions given by SimScale *SimScale online documentation* n.d.:

$$\mathrm{AR}^{SS}_{hexa} = \frac{\max_i e_i}{\min_i e_i}, \tag{6}$$

$$\mathrm{AR}^{SS}_{tetra} = \frac{\max_i e_i}{2\sqrt{6}r}, \tag{7}$$

where $\{e_i\}$ are the edges of a cell and r is the radius of the cell's internal sphere. Although at first sight (2) and (6) might look similar, (6) uses only edges, whereas (2) the distances between all the vertices.

B.2 Recognizing anisotropic fine cells

When considering a *fine* cell, one possible way to approximate the distance between vertices is by taking advantage of the weights of the edges of the graph, which geometrically reflect the faces (resp., edges) of a 3D (2D) mesh. Indeed, in 3D, the square root of such a quantity could be considered as an approximation of the diameter of the face, hence, an average of the

distances between the vertices of a given face. With this reasoning, one derives the following definition (recalled from section 3.3):

$$\mathrm{AR}_{fc} = \sqrt{\frac{\max_i w_i}{\min_i w_i}} \tag{1}$$

where $\{w_i\}$ are the weights associated to the neighbours of the cell. In 2D, the simple ratio suffices:

$$\mathrm{AR}_{fc} = \frac{\max_i w_i}{\min_i w_i}. \tag{8}$$

Two remarks are of interest here:

- Both eqs. (1) and (8) are non dimensional as (2), that is the reason for the different formula depending on the geometric dimension.
- With respect to (2) where one computes the distances between *all* the vertices, eqs. (1) and (8) consider only vertices belonging to the same face/edge.

Old versions of CoMMA. Previous versions used the formula often considered by Mavriplis, that is, (4). We have dropped this formula because, in our opinion, the ration of a maximum over an average is a weird quantity, and might lead to some misunderstanding now that the users can choose the anisotropy threshold.

B.3 The aspect ratio of a coarse cell

CoMMA tries to minimize the AR of the CC that it builds. Another aim is that the CC should be *compact* (here, the definition is left willingly generic). CoMMA still has some issues in recognizing the faces of a CC, see fig. 19, for this reason, using again (1) would not be possible. Relying on similar ideas to the ones which inspired (7), that is, comparing a characteristic length of the cell with respect to the one of a cube with the same volume. This led us to (3), reported here for the sake of clarity:

$$\mathrm{AR}_{cc} = \frac{\max\limits_{fc_i, fc_j \in cc} d\left(c_{fc_i}, c_{fc_j}\right)}{\sqrt[3]{\mathrm{vol}(cc)}} \tag{3}$$

where $\{fc_i\}$ the FC of the CC, c_{fc_i} the center of the FC, and finally $\mathrm{vol}(cc)$ the volume of the CC. Mind that, in 2D, one should consider the square root of the surface of the CC. We rely on the centers of the FC to compute a characteristic length which, however, has little to do with, say, the edges of the CC, but, again, it is the best that we can do with the information at our disposal.

Finally, mind that for the purposes of finding the best next cell to agglomerate, we are not really interested in the value itself given by a candidate CC, but on how it compares with other candidates.

Old versions of CoMMA. Previous versions used the following definition of AR of a CC independently of the dimension (2- or 3D):

$$\mathrm{AR}^{old}_{cc} = \frac{\left(\sum\limits_{f \in F^{ext}_{cc}} |f|\right)^{\frac{3}{2}}}{\mathrm{vol}(cc)} \tag{9}$$

where, in 3D (resp. 2D) F^{ext}_{cc} is the set of the external faces (resp. edges) of the CC, that is, faces which are both faces of the CC and of its FC, see top part of fig. 25. Equation (9) has been dropped because it has some drawbacks. First of all, in 2D it does not lead to a non-dimensional value as the AR should be. Moreover, it doesn't always do a good job in identifying the anisotropy of a cell: indeed, see the example at the bottom of fig. 25.

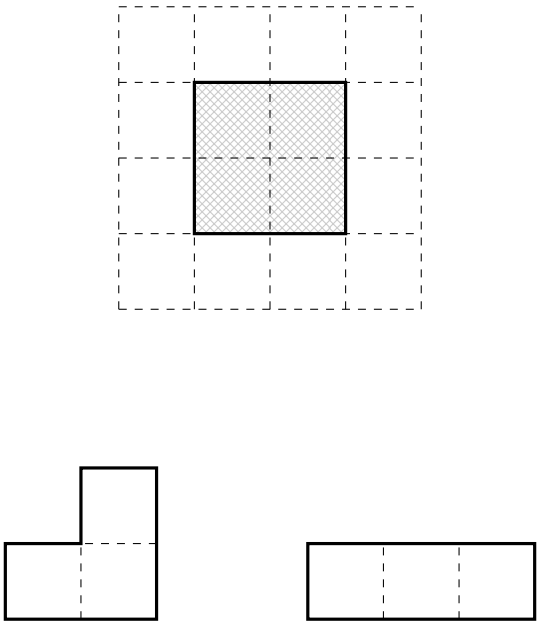


Figure 25: Top: A CC (hatched gray zone) defined from a fine Cartesian grid (dashed lines); bold line: external faces of the CC, F_{cc}^{ext}. Bottom: Two CC composed by 3 squares that would have the same AR when it is computed with (9) (vol = 3, $\sum_{F_{cc}^{ext}} |f| = 8$), while one would expect the left cell to have a lower AR.

C Deprecated data structures

Some data-structures have been created and used in previous version of the code, however they are not any more. They are kept for the sake of information.

- **BiMap**: A bidirectional map, where the link value-to-hash is given as well.
- **Queue**: A wrapper around a `deque` structure.
- **PriorityPair**: A wrapper around an index-weight pair with a custom comparator.
- **Tree** and related Node: the only difference with the standard structure is that it is optimized to access on the same level. Indeed, we have a reference to the parent node, only the *left* child, and to left and right neighbours on the same level. See fig. 26.

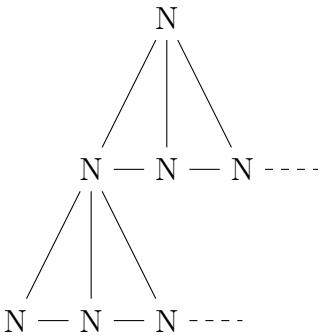


Figure 26: Scheme of the CoMMA `Tree` data structure.