

Mesh optimization for the virtual element method: How small can an agglomerated mesh become?

Original

Mesh optimization for the virtual element method: How small can an agglomerated mesh become? / Sorgente, Tommaso; Vicini, Fabio; Berrone, Stefano; Biasotti, Silvia; Manzini, Gianmarco; Spagnuolo, Michela. - In: JOURNAL OF COMPUTATIONAL PHYSICS. - ISSN 0021-9991. - ELETTRONICO. - 521:(2025), pp. 1-20. [10.1016/j.jcp.2024.113552]

Availability:

This version is available at: 11583/2995780 since: 2024-12-20T17:33:05Z

Publisher:

Elsevier

Published

DOI:10.1016/j.jcp.2024.113552

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

Mesh Optimization for the Virtual Element Method: How Small Can an Agglomerated Mesh Become?

T. Sorgente*, S. Berrone[†], S. Biasotti*, G. Manzini[‡], M. Spagnuolo*, F. Vicini[†]

April 18, 2024

Abstract

We present an optimization procedure for generic polygonal or polyhedral meshes, tailored for the Virtual Element Method (VEM). Once the local quality of the mesh elements is analyzed through a quality indicator specific to the VEM, groups of elements are agglomerated to optimize the global mesh quality. The resulting discretization is significantly lighter: we can remove up to 80% of the mesh elements, based on a user-set parameter, thus reducing the number of faces, edges, and vertices. This results in a drastic reduction of the total number of degrees of freedom associated with a discrete problem defined over the mesh with the VEM, in particular, for high-order formulations. We show how the VEM convergence rate is preserved in the optimized meshes, and the approximation errors are comparable with those obtained with the original ones. We observe that the optimization has a regularization effect over low-quality meshes, removing the most pathological elements. This regularization effect is evident in cases where the original meshes cause the VEM to diverge, while the optimized meshes lead to convergence. We conclude by showing how the optimization of a real CAD model can be used effectively in the simulation of a time-dependent problem.

Keywords: mesh optimization, mesh agglomeration, mesh quality indicators, virtual element method, optimal rates convergence

1 Introduction

During the last decades, solving Partial Differential Equations (PDEs) has experienced a substantial surge in its influence on research, design, and production.

*RAISE Ecosystem, Genova, Italy (tommaso.sorgente@cnr.it, silvia.biasotti@cnr.it, michela.spagnuolo@cnr.it).

[†]Dipartimento di Scienze Matematiche “Giuseppe Luigi Lagrange”, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy (stefano.berrone@polito.it, fabio.vicini@polito.it).

[‡]Istituto di Matematica Applicata e Tecnologie Informatiche “Enrico Magenes”, Consiglio Nazionale delle Ricerche, Via Ferrata 5/A, 27100 Pavia, Italy (marco.manzini@cnr.it).

PDEs are indispensable tools for modelling and analyzing phenomena across physics, engineering, biology, and medicine. The predominant methods for solving PDEs, such as the Finite Element Method, rely on suitable descriptions of geometrical entities, like the computational domain and its attributes, generally encoded by a mesh.

Despite extensive research and notable achievements, developing techniques for generating meshes with suitable geometrical properties remains an ongoing endeavour. Contemporary meshing algorithms typically generate an initial mesh, ideally dominated by well-shaped elements, and optionally execute optimization steps to enhance the geometrical quality of the elements [1]. Pivotal to these optimizations is the definition of the concept of “quality” of a mesh and its elements [2]. Although there is some consensus in the literature regarding the notion of quality for triangular/tetrahedral and quadrangular/hexahedral meshes, defining a universal quality indicator for generic polytopal (i.e., polygonal or polyhedral) meshes remains contentious, as evidenced by the myriad attempts [3, 4, 5, 6, 7, 8, 9, 10].

The main reason behind this difficulty is that the concept of “geometric quality of a cell” becomes quite vague when the cell is a generic polytope. Many classical quality indicators become meaningless when the element has a generic number of vertices or faces, or their extension is not straightforward. For instance, indicators based on the Jacobian operator can be extended to generic convex polygons, but are not defined for non-convex ones [3]. Most mesh generators and most numerical schemes get around this problem by only allowing convex or star-shaped elements and classifying the others as unlikely configurations. Alternatively, many mesh optimization strategies are tied solely to the optimization of the elements’ size, or the number of incident edges/faces [2].

The development of numerical schemes that support generic polytopal elements [11, 12, 13, 14, 15, 16] and the great advantages that they provide over classical methods, make it more and more urgent to devise suitable mesh optimization strategies. In particular, the Virtual Element Method (VEM) was appositely designed to enable computations over any polytopal cell, as it does not require the explicit computation of the basis functions [17]. To fully exploit its potentialities, we therefore need to be able to generate and optimize polytopal meshes. Polytopal mesh generation from scratch is generally addressed through Voronoi tessellations [18], while polytopal mesh optimization is recently emerging as a topic [19, 20, 21] but mainly for two-dimensional domains.

In this study, we tackle the problem of optimizing a given planar or volumetric mesh for the VEM, so that it contains the smallest possible number of elements, while still producing accurate results in a VEM numerical simulation. This is pursued through the use of a polytopal quality indicator [8, 9] which analyzes the geometrical shape of the elements and indicates groups of elements that can be merged into a single cell without compromising (and sometimes improving) the global mesh quality. The algorithm can be seen as a topological optimization method, which modifies the connectivity without changing the vertices position. To investigate its true potential, we test it on a collection of

appositely built non-optimal meshes. Indeed, if the quality of the original mesh is already high (e.g., if it mainly contains equilateral triangles), there is no real need to optimize it. Briefly, our algorithm generates optimized meshes that:

- contain up to 80% less elements than the original mesh, consequently reducing the number of degrees of freedom in the discrete problem defined over it;
- produce approximation errors comparable or improved to those produced by the original mesh;

and these achievements allow us to:

- preserve the optimal VEM convergence rates, both in L^2 and H^1 error norms;
- recover the optimal convergence rate in low-quality meshes, by removing pathological elements;
- reduce the total computational time, especially in time-dependent simulations.

The paper is structured as follows. In Section 2, we introduce the numerical problem to be solved and the VEM discretization used. We devote Section 3 to the description of the mesh quality indicator and the mesh quality optimization algorithm. In Section 4 we build a collection of meshes and optimize them with our algorithm. These meshes will be used for numerical tests in Section 5, where we report the performance of the VEM over them. Finally, in Section 6, we present an application of our algorithm to a real CAD model within the context of a time-dependent problem. To balance completeness and readability, we collect in A all the tables omitted from the other sections.

1.1 Notation and technicalities

We adhere to the standard definitions and notations of Sobolev spaces, norms, and seminorms, cf. [22]. Let k be a nonnegative integer. The Sobolev space $H^k(\omega)$ consists of all square-integrable functions with all square-integrable weak derivatives up to order k that are defined on the open, bounded, connected subset ω of \mathbb{R}^d , where $d = 1, 2, 3$. In the case where $k = 0$, we use the notation $L^2(\omega)$. The norm and seminorm in $H^k(\omega)$ are represented by $\|\cdot\|_{k,\omega}$ and $|\cdot|_{k,\omega}$ respectively, and for the inner product in $L^2(\omega)$ we use the $(\cdot, \cdot)_\omega$ notation.

Table 1 reports a brief summary of the notation used in this paper. We denote by $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$ an open, connected, and bounded set, with boundary $\partial\Omega$. We employ the symbol \mathcal{T}_h to represent a tessellation of Ω with mesh size h , i.e., a partition of Ω into non-overlapping polytopal elements E such that $\bigcup_{E \in \mathcal{T}_\Omega} E = \bar{\Omega}$. A polytopal element E (a polygon for $d = 2$ or a polyhedron for $d = 3$) is a compact subset of \mathbb{R}^d with boundary ∂E , size (area or volume) $|E|$, barycenter \mathbf{x}_E , and diameter $h_E = \sup_{\mathbf{x}, \mathbf{y} \in E} |\mathbf{x} - \mathbf{y}|$. The mesh

Table 1: Notation summary.

<i>Symbol</i>	<i>Description</i>
$d = \{2, 3\}$	Geometric dimension
$\Omega \subset \mathbb{R}^d$	Open, connected, and bounded set
$\partial\Omega \subset \mathbb{R}^{d-1}$	Boundary of Ω
\mathcal{T}_h	Tessellation of Ω with mesh size h
$E \in \mathcal{T}_h$	Polytopal mesh element
$ E , \mathbf{x}_E, h_E$	Size, barycenter and diameter of E
$\mathcal{V}_E, \mathcal{E}_E, \mathcal{F}_E$	Set of vertices, edges and faces of E
$\mathcal{V}_h, \mathcal{E}_h, \mathcal{F}_h$	Set of vertices, edges and faces of \mathcal{T}_h

size labeling each mesh \mathcal{T}_h is defined by $h = \max_{E \in \mathcal{T}_h} h_E$. We denote by \mathcal{V}_E , \mathcal{E}_E , and \mathcal{F}_E (if $d = 3$) the set of vertices, edges and faces of the element E , respectively. With the same intention, we use \mathcal{V}_h , \mathcal{E}_h , and \mathcal{F}_h to indicate the set of vertices, edges and faces (if $d = 3$) of the tessellation \mathcal{T}_h .

2 The Virtual Element Discretization

In this section, we consider an elliptic second-order differential problem in $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$, written in its variational formulation: given the functional space V , we look for $u \in V$ the solution of

$$a(u, v) = F(v) \quad \forall v \in V, \quad (1)$$

where

- V is the Sobolev space $H^1(\Omega)$, or $H_\Gamma^1(\Omega)$, Γ being a non-empty subset of $\partial\Omega$ with a non-zero $(d-1)$ dimensional Lebesgue measure where some set of essential boundary conditions are possibly imposed;
- $a : V \times V \rightarrow \mathbb{R}$ is a continuous and coercive bilinear form;
- $F : V \rightarrow \mathbb{R}$ is a continuous linear functional.

Under these hypothesis, the Lax-Milgram lemma implies that Problem (1) is well-posed; see, e.g., [23].

We approach the discretization of such a variational problem through the Virtual Element Method in a d -dimensional setting, as described in [24, 25]. Let E be a generic d -dimensional polytopal element of the domain partition \mathcal{T}_h . We fix a positive integer k as the order of the discrete approximation. We denote $\mathbb{P}_k^d(E)$ the set of d -dimensional polynomials of degree less than or equal to k defined on E . We conventionally assume that $\mathbb{P}_{-1}(\omega) = \{0\}$. In our implementation of the VEM for numerical tests, we use the standard basis of scaled monomials for all polynomial spaces [11].

We introduce two elliptic projection operators on E : namely, $\Pi_k^\nabla : H^1(E) \rightarrow \mathbb{P}_k^d(E)$ defined as

$$\begin{cases} (\nabla p, \nabla(\Pi_k^\nabla v - v))_E = 0, & \forall p \in \mathbb{P}_k^d(E), \\ (1, \Pi_k^\nabla v - v)_{\partial E} = 0, \end{cases}$$

and $\Pi_k^0 : L^2(E) \rightarrow \mathbb{P}_k(E)$ as

$$(p, \Pi_k^0 v - v)_E = 0, \quad \forall p \in \mathbb{P}_k^d(E).$$

On every element E , we consider the local virtual element space of order k that is defined as

- if $d = 2$,

$$\begin{aligned} V_k^E &= \{v \in H^1(E) : \Delta v \in \mathbb{P}_k^2(E), v|_e \in \mathbb{P}_k^1(e) \ \forall e \in \mathcal{E}_E, v|_{\partial E} \in C^0(\partial E), \\ &\quad (p, v - \Pi_k^\nabla v)_E = 0 \ \forall p \in \mathbb{P}_k^2(E) / \mathbb{P}_{k-2}^2(E)\}, \end{aligned} \quad (2)$$

- if $d = 3$,

$$\begin{aligned} V_k^E &= \{v \in H^1(E) : \Delta v \in \mathbb{P}_k^3(E), v|_F \in V_k^F \ \forall F \in \mathcal{F}_E, v|_{\partial E} \in C^0(\partial E), \\ &\quad (p, v - \Pi_k^\nabla v)_E = 0 \ \forall p \in \mathbb{P}_k^3(E) / \mathbb{P}_{k-2}^3(E)\}. \end{aligned} \quad (3)$$

The symbol $\mathbb{P}_k^d(E) / \mathbb{P}_{k-2}^d(E)$ denotes the set of polynomials in $\mathbb{P}_k^d(E)$ that are L^2 -orthogonal to $\mathbb{P}_{k-2}^d(E)$, and V_k^F is the virtual element space defined on the two-dimensional face F defined for $d = 2$.

With the local spaces of Equations (2)-(3), we define the global virtual element space

$$V_k = \{v \in C^0(\bar{\Omega}) \cap V : v|_E \in V_k^E \ \forall E \in \mathcal{T}_h\}. \quad (4)$$

Each function $v \in V_k \subset V$ can be uniquely identified, see [11], by the set of Degrees of Freedoms (DOFs)

- the value of v in each vertex of the set \mathcal{V}_h ;
- if $k > 1$, the value of v in the internal $k - 1$ Gauss quadrature nodes of each edge of set \mathcal{E}_h ;
- if $k > 1$ and $d = 3$, the value of the internal scaled moments of v of order $k - 2$ on each face of set \mathcal{F}_h ;
- if $k > 1$, the value of the internal scaled moments of v of order $k - 2$ on each $E \in \mathcal{T}_h$.

Both projection operators Π_k^∇ and Π_k^0 are computable on every E from these degrees of freedom, although the construction of Π_k^∇ on E recursively requires the construction of Π_k^∇ on every face $F \subset \partial E$.

We discretize Problem (1) using the local bilinear discrete form $a_h^E : V_k^E \times V_k^E \rightarrow \mathbb{R}$:

$$a_h^E(u, v) := a^E(\Pi_k^\nabla u, \Pi_k^\nabla v) + S^E(u - \Pi_k^\nabla u, v - \Pi_k^\nabla v).$$

where

- $a_h^E(\cdot, \cdot)$ is an approximation of the bilinear form $a^E(\cdot, \cdot)$, the restriction of the bilinear form $a(\cdot, \cdot)$ to the element E ;
- the bilinear form $S^E(\cdot, \cdot)$ is the *stabilization term*, which can be any computable, symmetric, positive definite bilinear form satisfying the stability condition

$$c_* a(v, v) \leq S(v, v) \leq c^* a(v, v), \quad \forall v \in V_k^E \cap \ker(\Pi_k^\nabla),$$

for some pair of real, positive constants c_* and c^* .

In the implementation for the numerical results, we apply the standard *dof-dof* stabilization proposed in [11].

This leads to the discrete counterpart of Problem (1): *Find $u_h \in V_k$ such that:*

$$a_h(u_h, v_h) := \sum_{E \in Th} a_h^E(u_h, v_h) = \sum_{E \in Th} F_h^E(v_h) =: F_h(v_h) \quad \forall v_h \in V_k, \quad (5)$$

where, for every $v_h \in V_k^E$, we set $F_h^E(v_h) = F^E(\Pi_k^0 v_h)$ as a local approximation of $F(v_h)$. Other details about $a(\cdot, \cdot)$, $F(\cdot)$, and their virtual element approximation are given in Section 5.

3 Mesh Quality Optimization Algorithm

The mesh optimization process involves two distinct steps. Initially, we evaluate the quality of the mesh elements by employing a local quality indicator. Subsequently, these elements are categorized and tagged, followed by the agglomeration of elements that bear identical tags. This methodology is applicable to both polygonal and polyhedral meshes. We detail the process specifically for three-dimensional meshes ($d = 3$), noting any variation that is relevant to two-dimensional cases ($d = 2$). The entire workflow, shown in Figure 1, has been implemented using the *cinolib* library [26].

3.1 Quality Indicator

We borrow the notion of quality from Sorgente et al [8, 9], who propose a mesh quality indicator specifically tailored for the Virtual Element Method. We briefly report the indicator definitions and refer the reader to the original

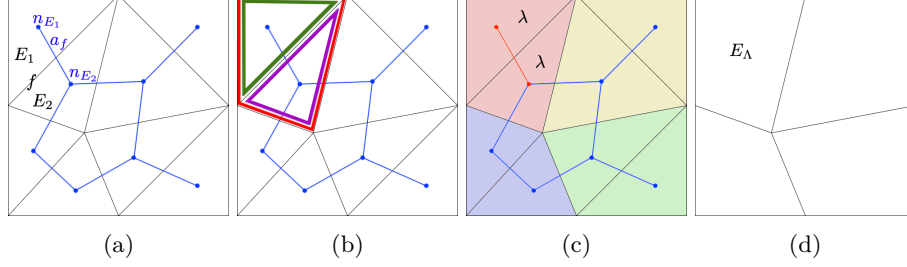


Figure 1: Optimization pipeline: (a) input mesh (black), with elements E_1, E_2 and their shared edge f , and mesh dual graph (blue), with nodes n_{E_1}, n_{E_2} and arc a_f ; (b) weights assignment: $w(n_{E_1})$ is the quality of the green polygon, $w(n_{E_2})$ is the quality of the purple polygon and $w(a_f)$ is the quality of the red polygon; (c) input mesh colored with respect to METIS labeling: nodes n_{E_1} and n_{E_2} are assigned the same label λ ; (d) agglomeration of the elements with the same label: elements E_1 and E_2 become a single element E_Λ .

papers for a more complete discussion. Given a polygonal mesh \mathcal{T}_h , for each element $E \in \mathcal{T}_h$ we have the following indicators:

$$\begin{aligned}
 \varrho_{2,1}(E) &= \frac{|kernel(E)|}{|E|} = \begin{cases} 1 & \text{if } E \text{ is convex} \\ \in (0, 1) & \text{if } E \text{ is concave and star-shaped} \\ 0 & \text{if } E \text{ is not star-shaped} \end{cases} \\
 \varrho_{2,2}(E) &= \frac{\min(\sqrt{|E|}, \min_{e \in \mathcal{E}_E} |e|)}{h_E} \\
 \varrho_{2,3}(E) &= \frac{3}{\#\{e \in \mathcal{E}_E\}} = \begin{cases} 1 & \text{if } E \text{ is a triangle} \\ \in (0, 1) & \text{otherwise} \end{cases} \\
 \varrho_{2,4}(E) &= \min_j \frac{\min_{e \in \mathcal{I}_E^j} |e|}{\max_{e \in \mathcal{I}_E^j} |e|} \tag{6}
 \end{aligned}$$

For polyhedral meshes, we assess the interior quality of an element E with a volumetric quality operator. Additionally, we evaluate the quality of the element's faces, denoted as \mathcal{F}_E , by employing the two-dimensional quality indicators $\varrho_{2,i}$. This dual approach ensures a comprehensive analysis of both the volumetric and surface attributes of the mesh.

$$\begin{aligned}
\varrho_{3,1}(E) &= \frac{|kernel(E)|}{|E|} \prod_{F \in \mathcal{F}_E} \varrho_{2,1}(F) \\
&= \begin{cases} 1 & \text{if } E \text{ and all its faces are convex} \\ \in (0, 1) & \text{if } E \text{ and all its faces are concave and star-shaped} \\ 0 & \text{if } E \text{ or one of its faces are not star-shaped} \end{cases} \\
\varrho_{3,2}(E) &= \frac{1}{2} \left[\frac{\min(\sqrt[3]{|E|}, \min_{F \in \mathcal{F}_E} h_F)}{h_E} + \frac{\sum_{F \in \mathcal{F}_E} \varrho_{2,2}(F)}{\#\{F \in \mathcal{F}_E\}} \right] \\
\varrho_{3,3}(E) &= \frac{1}{2} \left[\frac{4}{\#\{F \in \mathcal{F}_E\}} + \frac{\sum_{F \in \mathcal{F}_E} \varrho_{2,3}(F)}{\#\{F \in \mathcal{F}_E\}} \right] \\
&= \begin{cases} 1 & \text{if } E \text{ is a tetrahedron} \\ \in (0, 1) & \text{otherwise} \end{cases} \tag{7}
\end{aligned}$$

The *kernel* operator in $\varrho_{2,1}$ and $\varrho_{3,1}$ computes the kernel of an element (polygon or polyhedron), intended as the set of points from which the whole element is visible. The sets \mathcal{I}_E^j in $\varrho_{2,4}$ are the 1-dimensional disjoint sub-meshes corresponding to the edges of E (we consider each \mathcal{I}_E^j as a mesh, as it may contain more than one edge) such that $\mathcal{I}_E = \cup_j \mathcal{I}_E^j$, where \mathcal{I}_E is the 1-dimensional mesh induced by ∂E (see [8]). Indicator $\varrho_{2,4}$ does not have a natural 3D extension [9].

These indicators are combined together into an *element quality indicator*, which measures the overall quality of an element $E \in \mathcal{T}_h$:

$$\begin{aligned}
\varrho_2(E) &= \sqrt{\frac{\varrho_{2,1}(E)\varrho_{2,2}(E) + \varrho_{2,1}(E)\varrho_{2,3}(E) + \varrho_{2,1}(E)\varrho_{2,4}(E)}{3}} \\
\varrho_3(E) &= \sqrt{\frac{\varrho_{3,1}(E)\varrho_{3,2}(E) + \varrho_{3,1}(E)\varrho_{3,3}(E)}{2}} \tag{8}
\end{aligned}$$

For $d = 2, 3$ we have $\varrho_d(E) \approx 1$ if E is a perfectly-shaped element, e.g. an equilateral triangle/tetrahedron or a square/cube, $\varrho_d(E) = 0$ if and only if E is not star-shaped, and $0 < \varrho_d(E) < 1$ otherwise.

It is also possible to define a global function, which we call *mesh quality indicator* and denote by $\varrho_d(\mathcal{T}_h)$ with a small abuse of notation, to measure the overall quality of a d -dimensional mesh \mathcal{T}_h :

$$\varrho_d(\mathcal{T}_h) = \sqrt{\frac{1}{\#\{E \in \mathcal{T}_h\}} \sum_{E \in \mathcal{T}_h} \varrho_d(E)}. \tag{9}$$

All indicators in (6), (7), and consequently the local and global indicators (8), (9), only depend on the geometrical properties of the mesh elements; therefore their values can be computed before applying the VEM, or any other numerical scheme.

3.2 Mesh Partitioning

METIS [27] is a collection of serial programs implementing graph partitioning algorithms and finite element meshes, and producing fill-reducing orderings for sparse matrices. In the context of our work, we use it to perform a K -way partitioning of the dual graph of a mesh. We construct this graph by representing each mesh element as a node and establishing arcs between adjacent elements, as depicted in Figure 1a. Additionally, we assign weights to both nodes and arcs based on the element quality indicator (8), cf. Figure 1b. In detail:

- Each mesh element $E \in \mathcal{T}_h$ becomes a graph node n_E with the associated weight $w(n_E) = \varrho_d(E)$;
- Each internal mesh $(d-1)$ -dimensional object (which correspond to a face $F \in \mathcal{F}_h$ if $d = 3$ or an edge $e \in \mathcal{E}_h$ if $d = 2$), shared by elements E_1 and E_2 , becomes a graph arc a_F with weight $w(a_F) = \varrho_d(E_1 \cup E_2)$;
- $(d-1)$ -dimensional boundary objects (boundary faces if $d = 3$ or boundary edges if $d = 2$) are not part of the graph, as they are only shared by one element.

We store the weights in two arrays \mathbf{w}_n , \mathbf{w}_a that will be passed to METIS together with the mesh. To maximize their impact on the mesh partition process, and also because METIS only accepts integer weights, we re-scale the weights in \mathbf{w}_n from $[0, 1]$ onto the interval

$$\left[1 + \left\lfloor \min(\mathbf{w}_n) \frac{\#\mathbf{w}_n}{c_w} \right\rfloor, 1 + \left\lfloor \max(\mathbf{w}_n) \frac{\#\mathbf{w}_n}{c_w} \right\rfloor \right]$$

and similarly for \mathbf{w}_a . We add a constant 1 because zero-weights are not accepted, and divide by $c_w \in \mathbb{R}$ to avoid extremely big weights in very large meshes which may cause overflow problems. For the meshes in this work we used $c_w = 10$.

The routine *METIS-PartGraphKway* partitions a graph into K parts using multilevel K -way partitioning. We compute the number of required partitions K as a percentage of the number of elements in the mesh $\#\mathcal{T}_h$. This percentage is called the *optimization parameter* \mathcal{K} and it is defined by:

$$K = \mathcal{K}\%(\#\mathcal{T}_h) := \mathcal{K} \frac{\#\mathcal{T}_h}{100}.$$

If we set $\mathcal{K} = 20$ we are asking the optimization to partition the mesh into $K = 20\%(\#\mathcal{T}_h)$ parts, i.e., to generate a mesh with $1/5$ of the elements of \mathcal{T}_h . In our experiments, we could observe that values of \mathcal{K} below 5 lead to collapsing all the elements into a single one, while for \mathcal{K} values higher than 40 the number of partitions computed by METIS does not increase proportionally. These observations could highlight possible limitations of the partitioning algorithm that are independent of the quality indicator used. The *METIS-PartGraphKway* routine can be configured with a number of flags. In particular, we set *PartitionType: CutBalancing* and activated the flags *ContiguousPartitions*, *CompressGraph*, and *MinimizeConnectivity* (see the METIS documentation for the details [27]).

3.3 Mesh Agglomeration

The partition computed by METIS consists of an array containing a flag $\lambda_E \in [1, K]$ for each element E of the mesh, see Figure 1c. We aim to pick all the elements with the same flag and replace them with their union, which will be treated as a single element. In particular, for each flag $\lambda = 1, \dots, K$ we consider the set $\Lambda := \{E \in \mathcal{T}_h : \lambda_E = \lambda\}$ and build a new element E_Λ defined by all the faces (edges if $d = 2$) that are shared by only one element in Λ . While doing this operation, we check that the arising element is still a manifold. If not, we stop the agglomeration and skip to the next label, maintaining the old elements. This situation is extremely rare, but a single non-manifold element may compromise the whole mesh for certain applications.

At the end of the optimization process with input parameter K , the resulting mesh will have approximately K elements optimized for the computation of the virtual element basis functions. We will have removed around $\#\mathcal{T}_h - K$ elements from the mesh and at least $\#\mathcal{T}_h - K$ faces (edges if $d = 2$) between them, see Figure 1d. Moreover, if K is particularly low, we may happen to remove also edges (if $d = 3$) and vertices, but in a smaller number.

4 Datasets

This section details the generation and the optimization of a number of meshes specifically created for testing our algorithm. More precisely, we call *dataset* a sequence of meshes defined over the same domain, with decreasing mesh size, and built with the same technique, so that they contain similar elements organized in similar configurations. We generate four datasets to span the most common types of meshes currently used in numerical simulations: triangular and quadrangular in 2D, and tetrahedral and hexahedral in 3D. These meshes are built with random strategies so that they contain low-quality elements that make it meaningful, if not necessary, to think about their optimization. All the meshes presented in this paper are available for download at [GitHub](#) (*GitHub link to be added before publication*).

4.1 Planar Datasets

We generate two planar datasets on the unit square $\Omega = [0, 1]^2$, containing five meshes each. To compare our results, the i -th mesh of each dataset contains a similar number of vertices and elements.

Dataset *Tri* contains triangular meshes with randomly-shaped elements, generated by calling *Triangle* [28] on Ω with a set of initial points to be preserved and no other constraints. The initial points are randomly sampled on Ω . Hence, the resulting meshes contain several needles and flat elements. Dataset *Quad* contains quadrangular meshes with randomly-shaped quads. We start from an equispaced planar grid and randomly move all the points belonging to a certain plane π_1 to another plane π_2 , parallel to π_1 . In practice, we pick all the points sharing the same x -coordinate and randomly change x by the same quantity;

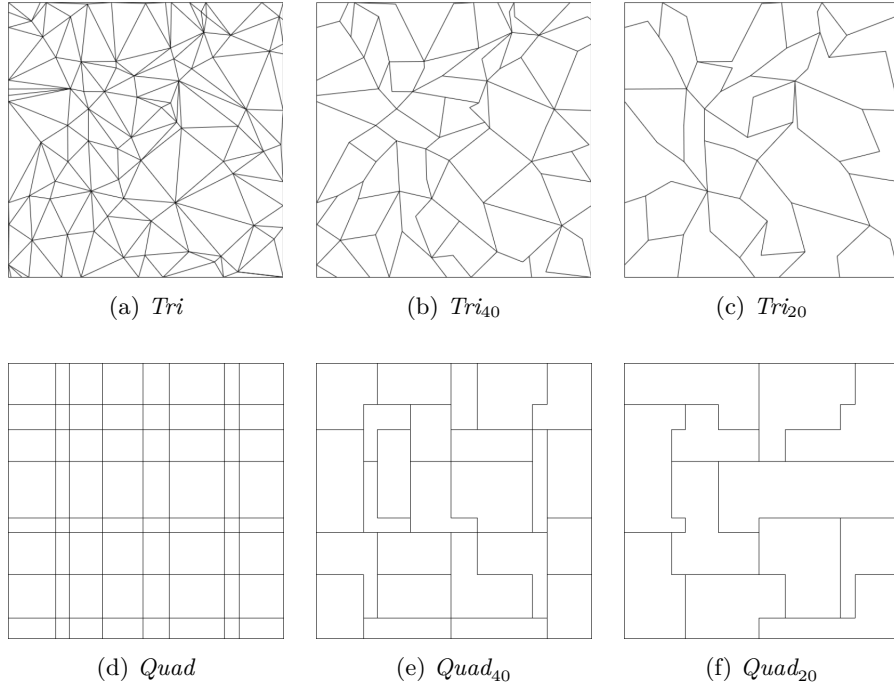


Figure 2: First mesh of datasets *Tri* and *Quad* optimized with $\mathcal{K} = 40$ and $\mathcal{K} = 20$.

then, we repeat this operation for the y coordinates. The resulting meshes contain quadrangular elements with very different sizes and aspect ratios. In Figure 2 we present the first mesh of datasets *Tri* and *Quad*, and their optimized versions with parameters $\mathcal{K} = 40$ and $\mathcal{K} = 20$.

In the datasets optimized with parameter 40, each mesh has less than half the elements of its corresponding original mesh, while in datasets optimized with parameter 20 each mesh has around $1/5$ of the elements of its corresponding original mesh. Moreover, while in the original datasets all the elements are guaranteed to be convex (triangles or quads), in the optimized ones a significant number of elements are not convex or even not star-shaped.

4.2 Volumetric Datasets

We also define two volumetric datasets on the unit cube $\Omega = [0, 1]^3$, containing five meshes each, with similar numbers of vertices and elements. They are built in analogy to their planar versions: for *Tet* we use *TetGen* [29] and for *Hex* we also perturb the z -coordinates. In Figure 3 we show the first mesh of these datasets and their optimized versions with parameters $\mathcal{K} = 40$ and $\mathcal{K} = 20$.

In Table 2, we present some numerics about dataset *Tet* and its optimized versions; analogous tables for the other datasets are reported in A (Tables 8, 7).

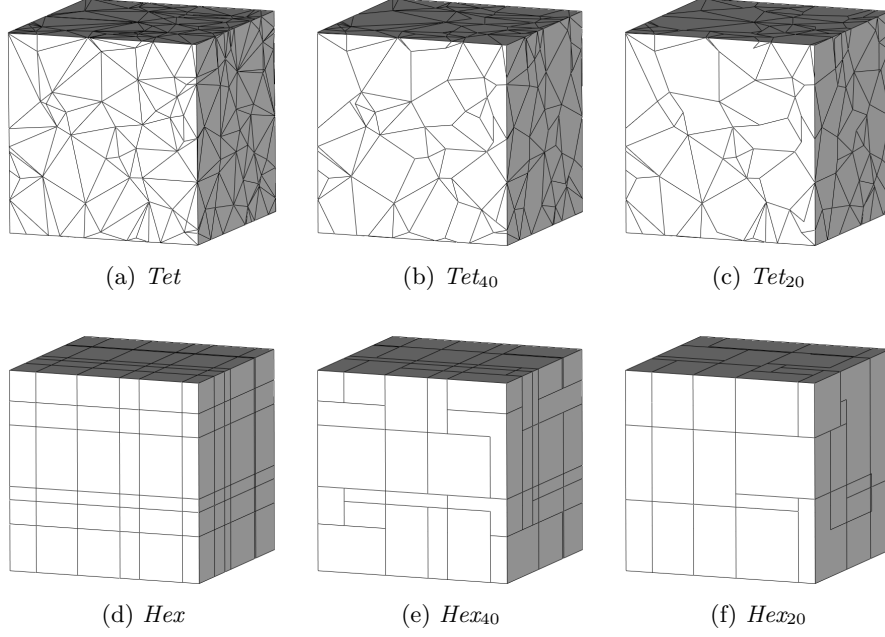


Figure 3: First mesh of datasets *Tet* and *Hex* optimized with $\mathcal{K} = 40$ and $\mathcal{K} = 20$.

Table 2: Geometric properties of datasets *Tet*, *Tet*₄₀, and *Tet*₂₀: number of vertices, faces and elements of each mesh; number of internal DOFs for $k = 1, 2, 3$; mesh quality (9).

<i>Dataset</i>	<i>Mesh</i>	# \mathcal{V}_h	\mathcal{T}_h		$k=1$	#DOFs		$\varrho_3(\mathcal{T}_h)$
			# \mathcal{F}_h	# \mathcal{T}_h		$k=2$	$k=3$	
<i>Tet</i>	1	512	4958	2332	216	9173	27164	0.827
<i>Tet</i>	2	1728	18836	9055	1000	36769	108032	0.829
<i>Tet</i>	3	4096	47086	22868	2744	94261	275900	0.830
<i>Tet</i>	4	8000	95184	46509	5832	193369	564776	0.831
<i>Tet</i>	5	13824	167952	82389	10648	344505	1004744	0.832
<i>Tet</i> ₄₀	1	510	3198	887	216	6023	16399	0.717
<i>Tet</i> ₄₀	2	1726	12298	3455	1000	24677	66602	0.719
<i>Tet</i> ₄₀	3	4095	30906	8728	2744	63751	171345	0.718
<i>Tet</i> ₄₀	4	7997	62630	17774	5832	131245	351996	0.720
<i>Tet</i> ₄₀	5	13819	110792	31351	10648	234607	627923	0.720
<i>Tet</i> ₂₀	1	505	2498	506	216	4775	12517	0.656
<i>Tet</i> ₂₀	2	1719	9506	1879	1000	19459	50405	0.651
<i>Tet</i> ₂₀	3	4081	23992	4671	2744	50675	130541	0.645
<i>Tet</i> ₂₀	4	7979	48775	9560	5830	104721	269262	0.649
<i>Tet</i> ₂₀	5	13794	86517	16989	10647	187679	481881	0.650

As already noted in Section 3.3, the number of vertices $\#\mathcal{V}_h$ does not change significantly after the optimization process. Instead, the number of elements $\#\mathcal{T}_h$ decreases as expected: meshes in Tet_{40} and Tet_{20} contain, approximately, 40% and 20% the elements of the corresponding meshes in Tet . The number of faces $\#\mathcal{F}_h$ decreases a bit more than the number of elements (in absolute value), given that each couple of merged elements shares at least one face which gets erased. For instance in mesh 5 from Tet to Tet_{40} we remove 51038 elements and 57160 faces, obtaining a mesh with 38% of the elements and 66% of the faces of the original one.

These considerations are reflected in the last three columns of the tables, in which we report the number of internal degrees of freedom, for $k = 1, 2, 3$, that we would define over each mesh in a VEM simulation. For $k = 1$ the DOFs coincide with the internal vertices and therefore remain almost identical, but for $k > 1$ we have an important reduction. For Tet_{40} we save on average the 33% in DOFs for $k = 2$ and the 38% for $k = 3$, while for Tet_{20} we save on average the 47% in DOFs for $k = 2$ and the 53% for $k = 3$. This means that, for instance, optimizing mesh 5 with $\mathcal{K} = 20$ we can save up to 5×10^6 DOFs for $k = 3$, obtaining a mesh that is significantly lighter to store and easy to handle.

We point out that the primary goal of the algorithm is to reduce the number of elements to the expected percentage. In this sense, the global mesh quality is likely to decrease after the optimization, as visible in the last column of Table 2. However, the loss of quality is optimal with respect to the number of elements removed. In other words, the optimized mesh has the maximum quality we can achieve by removing that number of elements.

5 Convergence Tests

In this section, we compare the performance of the VEM over the original datasets and the optimized ones.

5.1 Test Problem

On all the datasets from Section 4 we solve the Poisson problem:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{in } \partial\Omega, \end{aligned} \tag{10}$$

reported in the variational form of Problem (1) with the definition of

$$a(u, v) := (\nabla u, \nabla v)_\Omega, \quad F(v) := (f, v)_\Omega.$$

For the discretization, we employ the VEM described in Section 2. In particular, we define the approximated bilinear form of Equation (2) as:

$$a_h^E(u_h, v) = (\Pi_{k-1}^0 \nabla u_h, \Pi_{k-1}^0 \nabla v)_\Omega + S^E(u - \Pi_k^\nabla u, v - \Pi_k^\nabla v).$$

and $F_h^E(v) := (f, \Pi_k^0 v)_\Omega$. We use the symbol Π_{k-1}^0 for the L^2 -projection operator of a vector-valued function onto the polynomial space, applied component-wisely. Finally, we compute $f \in L^2(\Omega)$, using as ground truth the function

$$u(x, y, z) = 6.4xyz(x-1)(y-1)(z-1).$$

We compare the results plotting the relative L^2 -norm and H^1 -seminorm:

$$\mathcal{E}_{L^2} = \|u - u_h\|_{0,\Omega} / \|u\|_{0,\Omega}, \quad (11)$$

$$\mathcal{E}_{H^1} = |u - u_h|_{1,\Omega} / |u|_{1,\Omega}, \quad (12)$$

of the approximation error $u - u_h$ as the number of degrees of freedom increases. The optimal convergence rate of the method is indicated by the slope of a reference triangle in each plot. We want to ensure that the method converges with the correct rate over the optimized datasets, and we are interested in measuring differences in the approximation errors produced before and after the optimization.

Given a mesh \mathcal{T}_h over which we have computed an approximated solution u_h of u , and an optimized mesh \mathcal{T}'_h with approximated solution u'_h , we measure the difference between \mathcal{T}_h and \mathcal{T}'_h through the following quantities:

$$\Delta\text{DOFs} = \#\text{dofs}(\mathcal{T}_h) - \#\text{dofs}(\mathcal{T}'_h), \quad (13)$$

$$\Delta\mathcal{E}_{L^2} = \frac{\|u - u_h\|_{0,\Omega} - \|u - u'_h\|_{0,\Omega}}{\|u\|_{0,\Omega}}, \quad (14)$$

$$\Delta\mathcal{E}_{H^1} = \frac{|u - u_h|_{1,\Omega} - |u - u'_h|_{1,\Omega}}{|u|_{1,\Omega}}. \quad (15)$$

In practice, $\Delta\mathcal{E}_{L^2} > 0$ means that the relative error produced by the VEM over \mathcal{T}'_h is smaller than the one produced over \mathcal{T}_h , and similarly for $\Delta\mathcal{E}_{H^1}$ and ΔDOFs . We expect optimized meshes to produce less accurate results than the original ones (i.e., $\Delta\mathcal{E}_{L^2} < 0$ and $\Delta\mathcal{E}_{H^1} < 0$) because these discretizations contain many fewer degrees of freedom and, therefore, information. In fact, $\Delta\text{DOFs} \geq 0$ for all meshes. However, if the loss in accuracy is limited, the optimization might be convenient. For instance, if $\Delta\mathcal{E}_{L^2} \sim -10^{-7}$ and $\Delta\mathcal{E}_{H^1} \sim -10^{-4}$, but $\Delta\text{DOFs} \sim 10^5$, we have a significant gain in terms of space and computations at a very small cost in terms of accuracy.

Finally, we compare the computational time:

$$\Delta T = T_{\text{solve}}(\mathcal{T}_h) - (T_{\text{optimize}}(\mathcal{T}_h) + T_{\text{solve}}(\mathcal{T}'_h)), \quad (16)$$

where $T_{\text{solve}}(\mathcal{T}_h)$ is the time required for solving (10) on \mathcal{T}_h , and $T_{\text{optimize}}(\mathcal{T}_h)$ is the time required for optimizing \mathcal{T}_h into \mathcal{T}'_h .

5.2 Planar Datasets

We begin our analysis by checking the convergence of the VEM for $k = 1, 2, 3$ over the planar datasets *Tri* and *Quad* from Section 4.1 and their optimizations

with $\mathcal{K} = 20$. First, we note from the convergence plots in Figure 4 how the optimized datasets Tri_{20} and $Quad_{20}$ preserve the optimal convergence rate both in L^2 and H^1 norms.

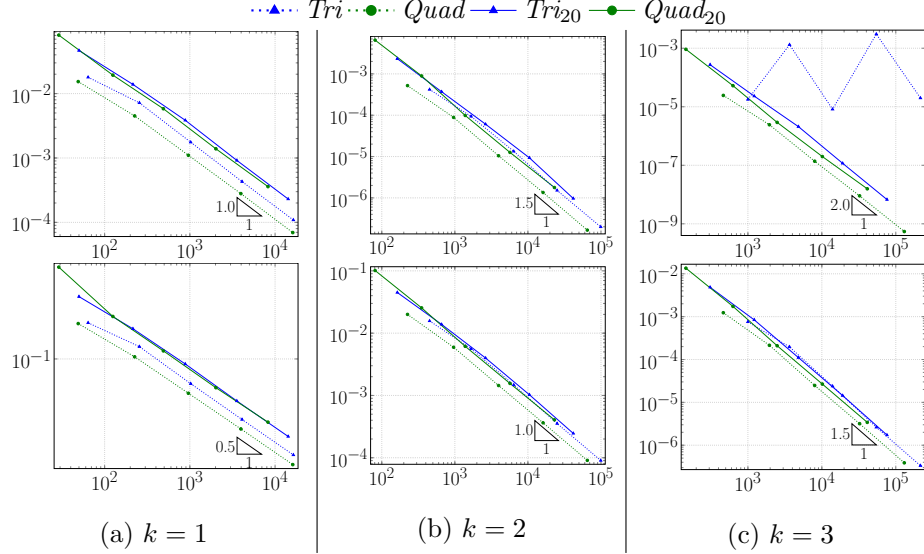


Figure 4: Convergence of datasets Tri_{20} , $Quad_{20}$ (continuous lines) compared to their original datasets Tri , $Quad$ (dotted line). We measure \mathcal{E}_{L^2} (top figures) and \mathcal{E}_{H^1} (bottom figures) with respect to the number of DOFs.

We then consider the quantities (13), (14), and (15), computed between the last meshes (mesh 5) of Tri and Tri_{20} :

- for $k = 1$, we have $\Delta\text{DOFs} \sim 10^3$, $\Delta\mathcal{E}_{L^2} \sim -10^{-4}$, and $\Delta\mathcal{E}_{H^1} \sim -10^{-3}$;
- for $k = 2$, we have $\Delta\text{DOFs} \sim 10^4$, $\Delta\mathcal{E}_{L^2} \sim -10^{-7}$, and $\Delta\mathcal{E}_{H^1} \sim -10^{-4}$;
- for $k = 3$, we have $\Delta\text{DOFs} \sim 10^5$, $\Delta\mathcal{E}_{L^2} \sim +10^{-5}$, and $\Delta\mathcal{E}_{H^1} \sim -10^{-6}$.

Full details for the other meshes are reported in Table 10 of the Appendix. We note how for $k = 1, 2$ the method performs slightly worse on the optimized meshes, despite having significantly fewer degrees of freedom, and this difference decreases as k grows. The difference in ΔDOFs is appreciable because the dots relative to optimized meshes are shifted towards the left in Figure 4. For $k = 3$ the error \mathcal{E}_{L^2} produced by the original meshes is so high, not only in the last mesh, that the method fails to converge. This is due to extremely small and flat triangles in the original meshes, which cause enormous condition numbers. Such elements disappear in the optimized meshes, and the VEM converges properly over them. For instance, in dataset Tri we have $\text{cond} \sim 10^{12}$ and $\text{cond} \sim 10^{15}$ for meshes 3 and 4, while in Tri_{20} we have $\text{cond} \sim 10^5$ and $\text{cond} \sim 10^6$. Therefore, we have fewer DOFs and smaller \mathcal{E}_{L^2} in this case.

For *Quad* and *Quad*₂₀, the values of ΔDOFs , $\Delta\mathcal{E}_{L^2}$, and $\Delta\mathcal{E}_{H^1}$ are essentially analogous, see Table 10. The only significant difference is that the method does not diverge over *Quad* for $k = 3$: in mesh 5 we have $\Delta\text{DOFs} \sim 10^4$, $\Delta\mathcal{E}_{L^2} \sim -10^{-8}$, and $\Delta\mathcal{E}_{H^1} \sim -10^{-6}$. The difference $\Delta\mathcal{E}_{L^2}$ is negative in this case, but it is extremely small compared to ΔDOFs . The difference in computational time between the original and the optimized meshes is negligible both for *Tri* and *Quad*, as the considered meshes are relatively small. The quantity ΔT will become significant with the more complex meshes of the next section.

5.3 Volumetric Datasets

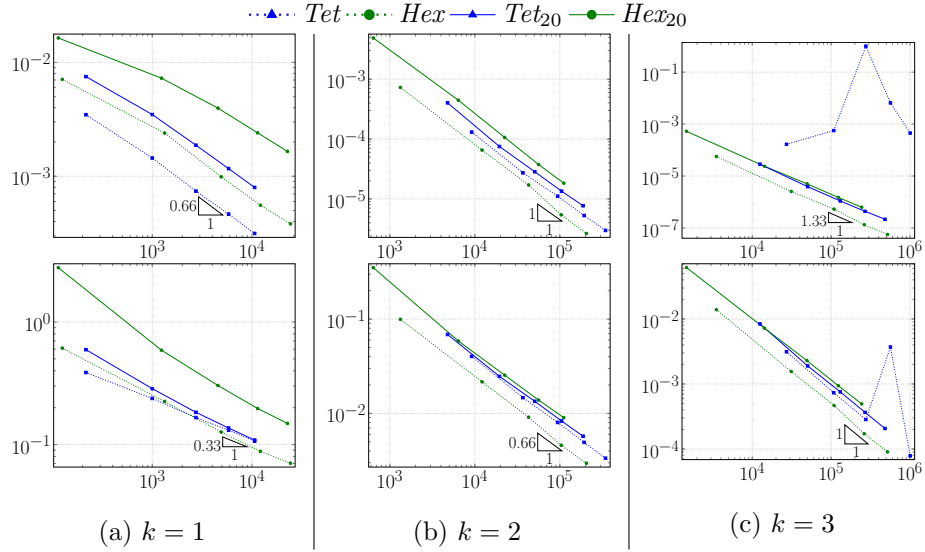


Figure 5: Convergence of datasets *Tet*₂₀ and *Hex*₂₀ (continuous lines) compared to their original datasets (dotted lines). We measure \mathcal{E}_{L^2} (top figures) and \mathcal{E}_{H^1} (bottom figures) with respect to the number of DOFs.

In Figure 5 we present analogous plots for the 3D datasets *Tet* and *Hex*, both optimized with $\mathcal{K} = 20$. As for the planar datasets, all optimized datasets produce optimal convergence rates, even when the original dataset does not (see *Tet* for $k = 3$).

In Table 3 we report the usual data about the differences between *Tet* and *Tet*₂₀:

- for $k = 1$ we have almost no DOFs reduction, with $\Delta\mathcal{E}_{L^2} \sim -10^{-4}$ and $\Delta\mathcal{E}_{H^1}$ increasing until becoming a positive 10^{-4} in the last mesh;
- for $k = 2$ ΔDOFs reaches 10^5 with $\Delta\mathcal{E}_{L^2} \sim -10^{-6}$ and $\Delta\mathcal{E}_{H^1} \sim -10^{-3}$;

Table 3: VEM performance: difference between the number of DOFs (13), the \mathcal{E}_{L^2} error (14), the \mathcal{E}_{H^1} error (15), and the computational time (16) w.r.t. the original mesh.

<i>Mesh</i>	<i>k</i>	ΔDOFs	$\Delta\mathcal{E}_{L^2}$	$\Delta\mathcal{E}_{H^1}$	ΔT
1	1	–	$-3.99 \cdot 10^{-3}$	$-6.76 \cdot 10^{-2}$	$-7.21 \cdot 10^{-1}$
2	1	–	$-2.02 \cdot 10^{-3}$	$-1.60 \cdot 10^{-2}$	$-2.72 \cdot 10^0$
3	1	–	$-1.13 \cdot 10^{-3}$	$-7.06 \cdot 10^{-3}$	$-7.28 \cdot 10^0$
4	1	$2.00 \cdot 10^0$	$-7.03 \cdot 10^{-4}$	$-2.70 \cdot 10^{-4}$	$-1.50 \cdot 10^1$
5	1	$1.00 \cdot 10^0$	$-4.85 \cdot 10^{-4}$	$+9.29 \cdot 10^{-4}$	$-2.72 \cdot 10^1$
1	2	$4.40 \cdot 10^3$	$-2.72 \cdot 10^{-4}$	$-2.80 \cdot 10^{-2}$	$-5.54 \cdot 10^{-1}$
2	2	$1.73 \cdot 10^4$	$-4.80 \cdot 10^{-5}$	$-1.00 \cdot 10^{-2}$	$-2.58 \cdot 10^0$
3	2	$4.36 \cdot 10^4$	$-1.73 \cdot 10^{-5}$	$-5.42 \cdot 10^{-3}$	$-7.04 \cdot 10^0$
4	2	$8.86 \cdot 10^4$	$-8.19 \cdot 10^{-6}$	$-3.38 \cdot 10^{-3}$	$+1.44 \cdot 10^1$
5	2	$1.57 \cdot 10^5$	$-4.69 \cdot 10^{-6}$	$-2.38 \cdot 10^{-3}$	$+1.63 \cdot 10^1$
1	3	$1.46 \cdot 10^4$	$+1.38 \cdot 10^{-4}$	$-5.27 \cdot 10^{-3}$	$-1.23 \cdot 10^0$
2	3	$5.76 \cdot 10^4$	$+5.58 \cdot 10^{-4}$	$-1.17 \cdot 10^{-3}$	$-5.23 \cdot 10^0$
3	3	$1.45 \cdot 10^5$	$+1.00 \cdot 10^0$	$-4.69 \cdot 10^{-4}$	$+3.67 \cdot 10^1$
4	3	$2.96 \cdot 10^5$	$+6.51 \cdot 10^{-3}$	$+3.33 \cdot 10^{-3}$	$+1.27 \cdot 10^2$
5	3	$5.23 \cdot 10^5$	$+4.51 \cdot 10^{-4}$	$-1.28 \cdot 10^{-4}$	$+4.98 \cdot 10^2$

- for $k = 3$ the VEM diverges on the original meshes and we get $\Delta\mathcal{E}_{L^2} > 0$ on every mesh.

Results for *Hex* and *Hex*₂₀ can be found in Table 9. For $k = 1$ the difference between the original and the optimized meshes is higher, also in terms of ΔDOFs , but for $k > 1$ the errors become smaller and smaller, even if the original dataset does not diverge in this case.

Looking at the times, for $k = 1$ we have a negative ΔT for both *Tet*₂₀ and *Hex*₂₀. This is because the elements of the optimized meshes generally have more complicated shapes than the simple tets or hexes of the original ones. At the same time, for $k = 1$ the number of DOFs does not decrease significantly, hence the linear system to be solved is more complex and has the same size. However, once k increases and ΔDOFs becomes significant, the size of the system reduces drastically, and we see positive ΔT values for $k = 2, 3$.

In conclusion, we note how the optimization becomes more interesting for high values of k and more complicated meshes. An optimized mesh containing 10^5 less DOFs, which produces results only 10^{-6} less accurate, leads to a faster and cheaper simulation. Moreover, the optimization removes small and flat elements in meshes from *Tet* that cause the VEM to diverge, and we obtain a numerical approximation that was impossible to compute on the original meshes.

5.4 Role of the Optimization Parameter

We now analyze the impact of parameter \mathcal{K} on the optimization process, comparing the performance produced by dataset *Tet* optimized with $\mathcal{K} = 5, 10, 20, 30, 40$,

Table 4: Role of parameter \mathcal{K} in the optimization of dataset *Tet*. Each column shows the average (among the five meshes in the dataset) percentage reduction with respect to the original dataset.

\mathcal{K}	$\#\mathcal{T}_h$ [%]	#DOFs [%]		
		$k = 1$	$k = 2$	$k = 3$
5	89.48	3.21	62.92	68.49
40-25	89.12	0.92	56.92	63.36
10	86.46	0.43	55.58	61.78
20	79.19	0.01	46.53	52.86
30	69.90	0.00	38.51	44.48
40	61.87	0.00	32.72	38.21
50	58.17	0.00	29.93	35.22

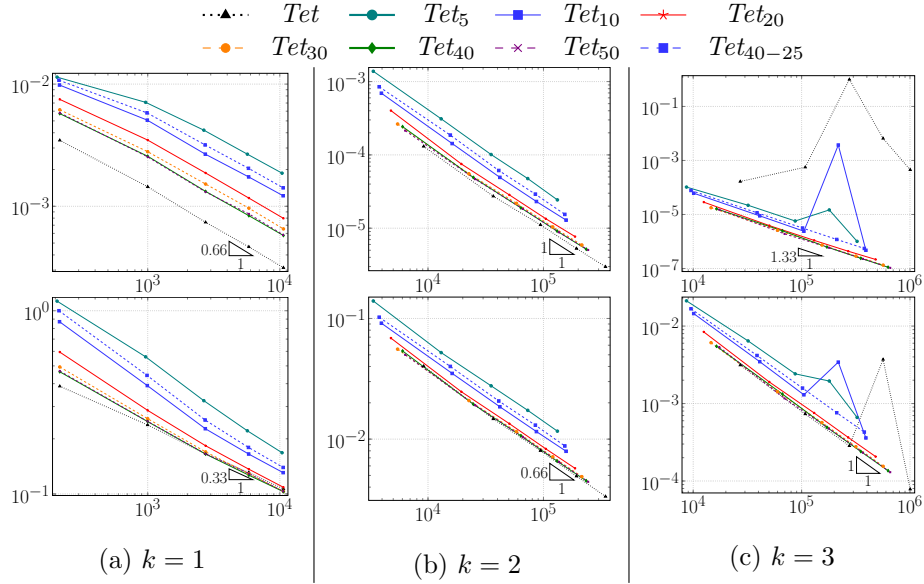


Figure 6: Comparison between dataset *Tet* and its optimized versions with $\mathcal{K} \in \{5, 10, 20, 30, 40, 50, 40-25\}$. We measure \mathcal{E}_{L^2} (top figures) and \mathcal{E}_{H^1} (bottom figures) concerning the number of DOFs.

and 50. We point out that *Tet* and *Tet*₂₀ are the same datasets shown in Figure 5, and that we reported in Table 2 the number of vertices, faces, elements and DOFs for meshes in *Tet*, *Tet*₄₀, and *Tet*₂₀. We also try to process the dataset recursively, i.e., to further optimize *Tet*₄₀ with $\mathcal{K} = 25$, obtaining a dataset *Tet*₄₀₋₂₅. This dataset contains the 25% of the elements of *Tet*₄₀, which contains the 40% of the elements of *Tet*. Overall, *Tet*₄₀₋₂₅ should be comparable with *Tet*₁₀.

In Table 4 we present the reduction, in terms of numbers of elements $\#\mathcal{T}_h$

and DOFs, of each optimized dataset compared to *Tet*, and in Figure 6 the VEM performance over them. First, we note how the algorithm is generally able to satisfy the elements reduction required by the parameter: for $\mathcal{K} = 40$ we have around -60% elements, for $\mathcal{K} = 30$ we have around -70% elements, and so on. As already noted in Section 3.3, we reach a plateau for $\mathcal{K} > 40$ because METIS implementation, therefore results for $\mathcal{K} = 50$ are similar to those for $\mathcal{K} = 40$. At the opposite side of the range, when optimizing with $\mathcal{K} \leq 10$ the correspondence between \mathcal{K} and the elements reduction becomes weaker (86% elements reduction instead of 90%). However, if we subdivide the optimization into two steps, i.e. with $\mathcal{K} = 40 - 25$, we get closer to the desired reduction. Concerning the DOFs, for $k = 1$ we have only a small reduction for the smallest \mathcal{K} values, as expected. For $k = 2, 3$ the DOFs reduction is smaller than the elements reduction, but they vary similarly.

In the plots of Figure 6 we can observe how higher \mathcal{K} values produce higher errors, but also that this difference decreases when increasing the order k . Datasets *Tet*₁₀ and *Tet*₄₀₋₂₅ perform very similarly, but for $k = 3$, the former starts to oscillate while the latter does not. Hence, a single, aggressive optimization produces worse results than two conservative ones.

In conclusion, the choice of the best optimization parameter may depend on the order of the method. In any case, we suggest to set $\mathcal{K} < 50$ because the optimization does not work properly after that value. For low-order VEM, it seems preferable to use a higher optimization value because removing too many elements may significantly increase the errors. For $k > 1$ instead, the difference between $\mathcal{K} = 40$, $\mathcal{K} = 30$, and $\mathcal{K} = 20$ are very small in terms of \mathcal{E}_{L^2} while notable in the number of DOFs. Therefore, it is worth choosing a lower optimization value. Values of \mathcal{K} lower than 20 are not recommended because they do not always produce reliable results, especially for $k > 2$. In those cases, applying multiple optimizations with higher \mathcal{K} values is preferable.

5.5 Importance of Quality Weights

We also analyze the importance of using the quality indicator (8) for assigning METIS weights to nodes and arcs (see Section 3.2). In Figure 7 we consider dataset *Hex* optimized always with $\mathcal{K} = 20$, but with different weights:

- in *Hex*₂₀ we set quality weights on nodes and arcs, as done in the rest of the paper: $w(n_E) = \varrho(E)$ and $w(a_F) = \varrho(E_1 \cup E_2)$ for all nodes n_E and arcs a_F ;
- in *Hex*_{20c} we set constant weights, using METIS in its original mode: $w(n_E) = 1$ and $w(a_F) = 1$ for all n_E, a_F ;

Different weights produce optimized meshes with similar numbers of vertices and elements, but organized in different configurations. Using constant weights, we optimize the mesh following the METIS internal criterion for graph partitioning, which essentially tries to agglomerate elements in compact and convex configurations. This is conceptually not so far from what the quality indicator (7)

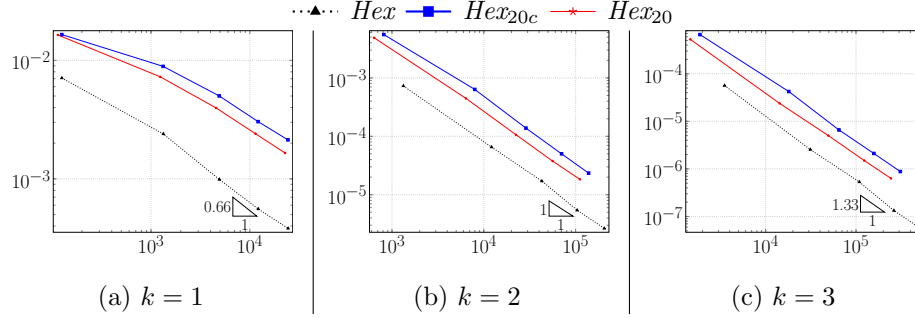


Figure 7: Error \mathcal{E}_{L^2} with respect to the number of DOFs produced by *Hex* optimized with $\mathcal{K} = 20$ and constant weights (*Hex*_{20c}), or quality weights (*Hex*₂₀).

suggests, in particular, $\varrho_{3,1}$ and $\varrho_{3,2}$ promote exactly such types of polytopes. Therefore, it is reasonable that the output meshes will not be so different from each other, and we want to make sure that the use of the quality indicator has an impact.

We see how *Hex*₂₀ produces smaller \mathcal{E}_{L^2} values (and \mathcal{E}_{H^1} values are analogous) than *Hex*_{20c}, and this difference remains constant when increasing k . Moreover, meshes in *Hex*₂₀ contain fewer DOFs than those in *Hex*_{20c}, as indicated by the position of the dots in the plot. It is therefore important to set the weights using a quality criterion, also considering that the weights computation time is negligible compared to the time needed for METIS to partition the graph.

6 CAD Application

In this last section we test our algorithm over a real CAD model of a *heat sink*, downloaded from the repository *Traceparts*¹. Using *fTetWild* [30], we re-meshed the original triangulation obtaining a higher quality surface mesh, then we generated a tetrahedralization of the inside. The considered model is particularly complex because it presents three different levels of detail, which determine the generation of elements with variable size. The resulting mesh, noted *Cad*, is visible in Figure 8, together with its optimization with parameter $\mathcal{K} = 20$. In the “exploded” version, elements are colored with respect to their quality, from yellow ($\varrho_3 = 1$) to blue ($\varrho_3 = 0$). While *Cad* contains only tets, which are very high quality, in *Cad*₂₀ the 80% of the elements have been agglomerated into generic polyhedra. The local quality of *Cad*₂₀ is therefore generally lower, and this is an unavoidable side-effect when optimizing high-quality meshes. However, our guess is that the loss in local quality is negligible when compared to the gain in terms of DOFs.

¹www.traceparts.com/it/product/fischer-elektronik-gmbh-co-kg-strangkuhlkorper-fur-einrasttransistorhaltefeder?CatalogPath=TRACEPARTS%3ATP014002&Product=34-23062017-133913&PartNumber=SK%20593%2025&corid=0737fe6b-01e1-4714-6dec-a98692f099d7

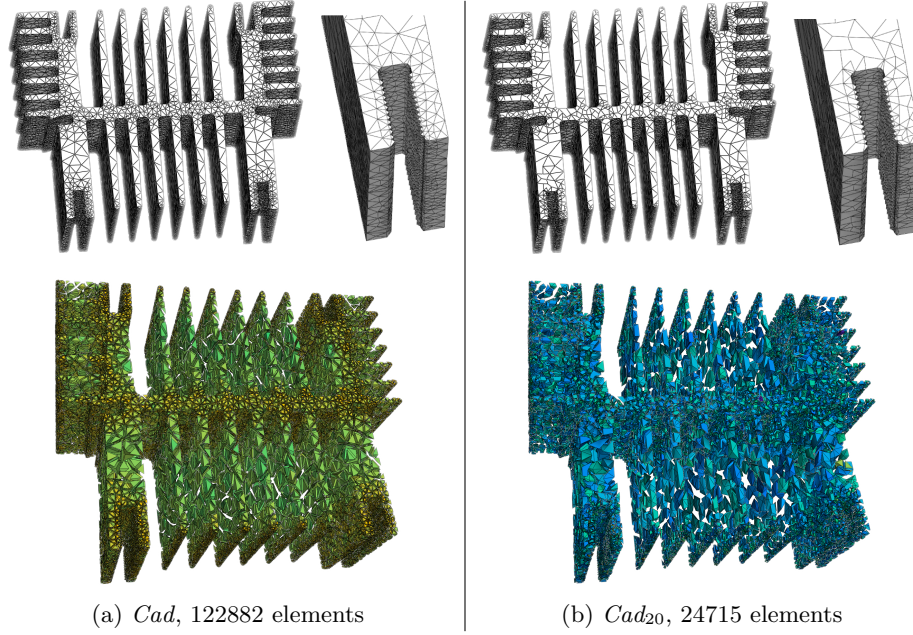


Figure 8: *Cad* model optimized with $\mathcal{K} = 20$, with a close-up on a small-scale detail (top) and elements colored w.r.t. their quality (bottom).

6.1 Time-Dependent Problem

Table 5: Differences between *Cad* and *Cad*₂₀ for $k = 1, 2, 3, 4$ at $t = 1$.

k	$\Delta\mathcal{T}_h$	ΔDOFs	$\Delta\mathcal{E}_{L^2}$	$\Delta\mathcal{E}_{H^1}$	r_{L^2}	r_{H^1}
1	$8.51 \cdot 10^4$	$4.00 \cdot 10^0$	$-3.33 \cdot 10^{-3}$	$+1.25 \cdot 10^{-2}$	$-4.01 \cdot 10^{-1}$	$+2.10 \cdot 10^{-1}$
2	$8.51 \cdot 10^4$	$1.91 \cdot 10^5$	$-1.27 \cdot 10^{-3}$	$-1.19 \cdot 10^{-2}$	$-2.66 \cdot 10^{-6}$	$-3.43 \cdot 10^{-6}$
3	$8.51 \cdot 10^4$	$6.47 \cdot 10^5$	$-7.10 \cdot 10^{-5}$	$-9.75 \cdot 10^{-4}$	$-4.37 \cdot 10^{-8}$	$-8.32 \cdot 10^{-8}$
4	$8.51 \cdot 10^4$	$1.45 \cdot 10^6$	$-1.15 \cdot 10^{-6}$	$-1.89 \cdot 10^{-5}$	$-3.14 \cdot 10^{-10}$	$-7.17 \cdot 10^{-10}$

Differently from Section 5, we now have a single mesh and we measure the advantages of solving multiple problems over a smaller and better (in the sense of quality) mesh. We solve the following time-dependent problem, defined on *Cad* and its optimized version *Cad*₂₀, for a number of time values between 0 and 1:

$$\begin{aligned}
\frac{\partial u}{\partial t} - \Delta u &= f && \text{in } \Omega, \forall t \in [0, 1], \\
u(t, x) &= 0 && \text{in } \partial\Omega, \forall t \in [0, 1], \\
u(0, x) &= 0 && \text{in } \bar{\Omega},
\end{aligned}$$

The two meshes share the same bounding box with centroid \mathbf{x} , origin $(x_0, y_0, z_0) \cong$

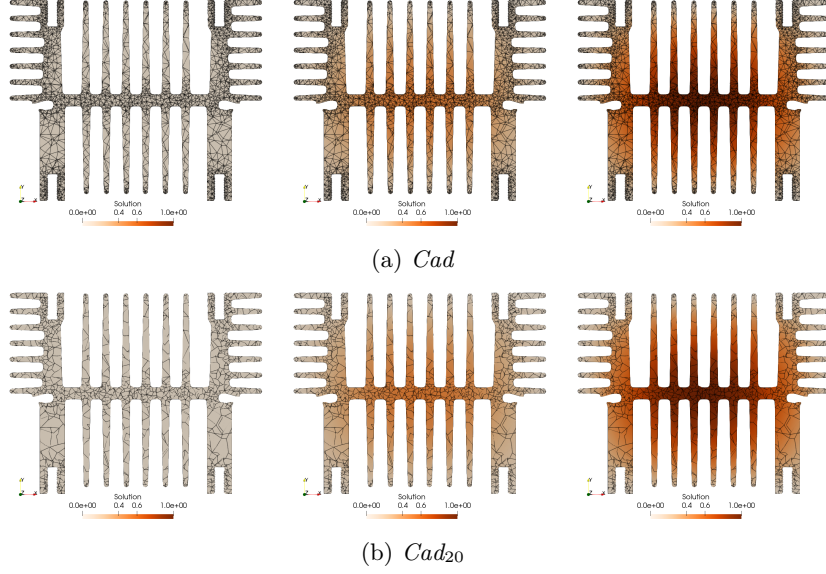


Figure 9: Simulation with $k = 1$ over models Cad (top) and Cad_{20} (bottom). From left to right, $t = 0$, $t = 0.5$, $t = 1$.

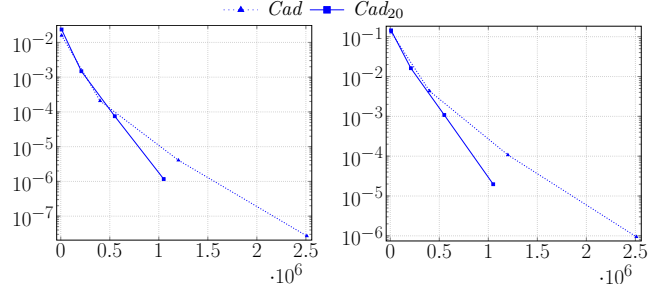


Figure 10: Performance of the VEM with $k = 1, 2, 3, 4$ at $t = 1$ on models Cad and Cad_{20} . We measure \mathcal{E}_{L^2} (left) and \mathcal{E}_{H^1} (right) with respect to the number of DOFs in *semilog-y* scale.

$(-0.035, 0.0, -25.002)$, and size $(x_s, y_s, z_s) \cong (75.58, 60.01, 25.05)$. Given a constant \tilde{u}_0 computed such that $u = 1$ when $t = 1$, we seek for the exact solution:

$$u(\mathbf{x}, t) = t\tilde{u}_0(x - x_0)(y - y_0)(z - z_0)(x - (x_0 + x_s))(y - (y_0 + y_s))(z - (z_0 + z_s)).$$

We use a backward Euler scheme with 10 time steps; visual results are presented in Figure 9.

In Figure 10 we plot \mathcal{E}_{L^2} and \mathcal{E}_{H^1} for $k = 1, 2, 3, 4$ relatively to $t = 1$, but these errors remain very similar for all time steps. Instead of refining the mesh at each iteration, as done in Section 5, we increase the order of the VEM

scheme. The original meshes produce more accurate results point-wise, i.e., the i -th dot of Cad is higher than the i -th dot of Cad_{20} in the plot. However, the difference in error is much smaller than the difference in terms of DOFs. Both plots converge linearly in *semilog-y* scale as expected, but the line relative to Cad_{20} is significantly below the Cad one.

In Table 5 we can see how ΔDOFs grows with k while $\Delta\mathcal{E}_{L^2}$ and $\Delta\mathcal{E}_{H^1}$ decrease, all with an exponential rate. Therefore, despite $\Delta\mathcal{E}_{L^2}$ and $\Delta\mathcal{E}_{H^1}$ being negative, their influence is smaller and smaller. The last column represents the ratio $r_{L^2} = \Delta\mathcal{E}_{L^2}/\Delta\text{DOFs}$, and analogously for r_{H^1} . When we compare two meshes in the plot we have an x -shift represented by ΔDOFs , and an y -shift represented by $\Delta\mathcal{E}_{L^2}$ (or $\Delta\mathcal{E}_{H^1}$). The ratio $r = dx/dy$ then represents the slope of the line connecting the two meshes. The smaller $|r|$, the more is convenient to pass from one mesh to the other, because the difference in DOFs overcomes the difference in error. For instance, for $k = 3$ we have $|r_{L^2}| \sim 10^{-8}$: we are losing a factor 10^{-8} in accuracy for every DOF that we removed from the original mesh.

6.2 Computational Time

Table 6: Differences between computational times, in seconds, for the time-dependent problem with $k = 3$ solved on Cad and Cad_{20} . In the last column, percentage of the total time required by the optimization of the mesh.

<i>Steps</i>	$\Delta\text{Assemble}$	$\Delta\text{Factorize}$	ΔSolve	ΔTime	<i>Optimize</i> [%]
$1 \cdot 10^1$	$-2.59 \cdot 10^1$	$-4.43 \cdot 10^{-2}$	$+4.97 \cdot 10^{-1}$	$-6.15 \cdot 10^1$	24.03
$1 \cdot 10^2$	$-2.59 \cdot 10^1$	$-4.43 \cdot 10^{-2}$	$+4.97 \cdot 10^0$	$-5.70 \cdot 10^1$	20.04
$1 \cdot 10^3$	$-2.59 \cdot 10^1$	$-4.43 \cdot 10^{-2}$	$+4.97 \cdot 10^1$	$-1.23 \cdot 10^1$	7.54
$1 \cdot 10^4$	$-2.59 \cdot 10^1$	$-4.43 \cdot 10^{-2}$	$+4.97 \cdot 10^2$	$+4.35 \cdot 10^2$	1.04
$1 \cdot 10^5$	$-2.59 \cdot 10^1$	$-4.43 \cdot 10^{-2}$	$+4.97 \cdot 10^3$	$+4.91 \cdot 10^3$	0.11

In Table 6 we report a comparison of the computational times required for the main operations involved in the process, for the case $k = 3$. As in the rest of the paper, the comparison is made subtracting the time relative to Cad_{20} from the time relative to Cad , hence a positive Δ means a gain in time when using the optimized mesh. Computations have been performed using a 3.60GHz Intel Core i7 processor with 16 cores and 32GB of RAM.

The first column contains the number of time steps considered, from 10 to 10^6 . The times for importing the mesh structure in the solver, computing the geometric properties, and assembling the linear system, are aggregated in *Assemble*, while *Factorize* indicates the time for factorizing the global matrix of the problem. These operations are performed only once, therefore these times are independent of the number of time steps. The differences $\Delta\text{Assemble}$ and $\Delta\text{Factorize}$ are negative because elements in optimized meshes generally have more complicated shapes than the simple tets of the original ones, despite being fewer. We indicate by *Solve* the time required for the solution of the

problem, resulting from the sum of *Steps* iterations. The solution time is faster on the optimized meshes because the problem itself becomes much smaller, and this gain gets multiplied by the number of time steps. *Time* indicates the total time, resulting from the sum of the previous ones and including also the optimization of the mesh. Note that $\Delta Time$ cannot be expressed in terms of (16), as each ΔT already contains the time for optimizing the mesh. This Δ is initially negative, but it grows with the number of time steps because the initial fixed costs (optimizing the mesh, assembling the system, factorizing the matrix) become negligible with respect to *Solve*. In particular, we report the impact of the optimization on the total time in the last column.

We could observe a difference in the *Factorize* cost with respect to *Tet* and *Hex* (Table 3 and Table 9). In those cases, the factorization step is significantly cheaper on the optimized mesh, resulting in a positive ΔT for $k > 1$ even if we were solving a single iteration of the problem. We associate this fact with the quality of the original mesh: *Tet* and *Hex* contain very bad-shaped elements, while *Cad* is made of good-quality tets. If the starting mesh is already good, there is less space for improvements, and our optimization becomes effective only after a certain number of time steps.

In particular, there exists a critical number of time steps \bar{t} such that for less than \bar{t} steps, computations are faster on the original mesh, while for more than \bar{t} steps the optimized mesh becomes advantageous. This number decreases as k grows: we have $\bar{t} = 2.63 \cdot 10^4$ for $k = 2$, $\bar{t} = 1.25 \cdot 10^3$ for $k = 3$, and $\bar{t} = 7.63 \cdot 10^2$ for $k = 4$. For $k = 1$ times are essentially equivalent, thus we could not identify a single \bar{t} value. However, the presented calculations were made using a code written and optimized for meshes with convex elements, therefore we believe further improvements can be made with solvers optimized for generic non-convex cells.

7 Conclusions

We presented an algorithm for optimizing the size of a mesh with respect to its quality, in the context of VEM simulations. The algorithm is able to significantly reduce the total number of mesh elements and the number of DOFs, in particular in high-order formulations, while preserving the VEM optimal convergence rates. These effects lead to a notable decrease in the computational effort required for obtaining the numerical solution. A scenario in which this decrease becomes particularly interesting is the class of time-dependent problems: in such cases, we optimize the mesh only once and then save time in each iteration. We also observed how, if the input mesh has a particularly low quality, the optimization can locally improve the shape of the elements. This improvement allows to control the stiffness matrix condition number and therefore to recover the optimal VEM convergence rate. The mesh optimization algorithm is therefore a powerful tool to achieve faster and more stable VEM simulations.

One potential limitation of the algorithm is that it only evaluates elements pairwise. When considering an element E with neighboring elements E' and E'' ,

the algorithm separately assesses the potential quality of $E \cup E'$ and $E \cup E''$, without considering the overall quality of $E \cup E' \cup E''$. Consequently, some small mesh elements may still persist in the optimized final mesh, particularly around small-scale shape features of the boundary. This issue can be controlled by subdividing the optimization into smaller steps (i.e., running the algorithm twice with $\mathcal{K} = 40$ and $\mathcal{K} = 25$ instead of once with $\mathcal{K} = 10$). However, the numerical findings presented in this study suggest that such action is generally not necessary for obtaining satisfactory results.

We believe that an interesting aspect of our algorithm is its generality, as it can be applied to any type of mesh. It is also modular: its key ingredients, the mesh quality indicator and the graph partitioning algorithm, may be easily replaced or adjusted for adapting the optimization to specific needs. With a different quality indicator, we could optimize the mesh to be used with numerical methods other than the VEM or include in the quality criterion features of the numerical problem (e.g., anisotropies). We refer the reader to [2] for more quality indicators. Other algorithms for graph partitioning may be more efficient, see the parallel version of METIS, ParMETIS [31]. In some situations instead, it may be useful to have a partitioning method that finds a local maximum of the global quality of the mesh and automatically decides the optimal number of partitions, as it is done in [21] using GraphCut [32]. Future research will investigate these aspects more deeply, as well as try to adapt a similar approach to mesh refinement strategies.

Acknowledgments

This work was carried out within the framework of the project “RAISE - Robotics and AI for Socio-economic Empowerment” and has been partially supported by European Union - NextGenerationEU. However, the views and opinions expressed are those of the authors alone and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

F. Vicini, S. Berrone, and G. Manzini are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM); this work has been partially supported by INdAM-GNCS Project CUP: E53C23001670001, by the MUR PRIN project 20204LN5N5_003, and by the European Union through the project Next Generation EU, PRIN 2022 PNRR project P2022BH5CB_001 CUP:E53D23017950001.

References

- [1] DSH Lo. *Finite element mesh generation*. CRC Press, New York, 2014.
- [2] T Sorgente, S Biasotti, G Manzini, and M Spagnuolo. A survey of indicators for mesh quality assessment. In *Computer Graphics Forum*, volume 42-2, pages 461–483. Wiley Online Library, 2023.

- [3] P Knupp. Algebraic mesh quality metrics. *SIAM Journal on Scientific Computing*, 23(1):193–218, 2001.
- [4] CJ Stimpson, CD Ernst, P Knupp, PP Pébay, and D Thompson. The Verdict library reference manual. *Sandia National Laboratories Technical Report*, 9(6), 2007.
- [5] R Chalmers, F Hurtado, V Sacristán, and M Saumell. Measuring regularity of convex polygons. *Computer-Aided Design*, 45(2):93–104, 2013.
- [6] J Zunic and PL Rosin. A new convexity measure for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):923–934, 2004.
- [7] W Huang and Y Wang. Anisotropic mesh quality measures and adaptation for polygonal meshes. *Journal of Computational Physics*, 410:109368, 2020.
- [8] T Sorgente, S Biasotti, G Manzini, and M Spagnuolo. The role of mesh quality and mesh quality indicators in the virtual element method. *Advances in Computational Mathematics*, 48(1):3, 2021.
- [9] T Sorgente, S Biasotti, G Manzini, and M Spagnuolo. Polyhedral mesh quality indicator for the virtual element method. *Computers & Mathematics with Applications*, 114:151–160, 2022.
- [10] S Berrone and A D’Auria. A new quality preserving polygonal mesh refinement algorithm for polygonal element methods. *Finite Elements in Analysis and Design*, 207:103770, 2022.
- [11] L Beirão da Veiga, F Brezzi, A Cangiani, G Manzini, LD Marini, and A Russo. Basic principles of virtual element methods. *Mathematical Models and Methods in Applied Sciences*, 23(01):199–214, 2013.
- [12] F. Brezzi, G. Manzini, D. Marini, P. Pietra, and A. Russo. Discontinuous Galerkin approximations for elliptic problems. *Numerical Methods for Partial Differential Equations*, 16(4):365–378, 2000.
- [13] L Beirão da Veiga, K Lipnikov, and G Manzini. *The mimetic finite difference method*, volume 11 of *Modeling, Simulations and Applications*. Springer, San Diego, CA, USA, I edition, 2014.
- [14] Natarajan Sukumar and A21015991073 Tabarraei. Conforming polygonal finite elements. *International Journal for Numerical Methods in Engineering*, 61(12):2045–2066, 2004.
- [15] Daniele Antonio Di Pietro and Jérôme Droniou. The hybrid high-order method for polytopal meshes. *Number 19 in Modeling, Simulation and Application*, 2020.

- [16] Enrico Bertolazzi and Gianmarco Manzini. A cell-centered second-order accurate finite volume method for convection–diffusion problems on unstructured meshes. *Mathematical Models and Methods in Applied Sciences*, 14(08):1235–1260, 2004.
- [17] L Beirão da Veiga, F Brezzi, LD Marini, and A Russo. The hitchhiker’s guide to the virtual element method. *Mathematical Models and Methods in Applied Sciences*, 24(8):1541–1573, 2014.
- [18] Dong Ming Yan, Wenping Wang, Bruno Lévy, and Yang Liu. Efficient computation of clipped voronoi diagram for mesh generation. *Computer-Aided Design*, 45(4):843–852, 2013.
- [19] Paola F Antonietti and E Manuzzi. Refinement of polygonal grids using convolutional neural networks with applications to polygonal discontinuous galerkin and virtual element methods. *Journal of Computational Physics*, 452:110900, 2022.
- [20] Paola F Antonietti, Stefano Berrone, Martina Busetto, and Marco Verani. Agglomeration-based geometric multigrid schemes for the virtual element method. *SIAM Journal on Numerical Analysis*, 61(1):223–249, 2023.
- [21] T Sorgente, F Vicini, S Berrone, S Biasotti, G Manzini, and M Spagnuolo. Mesh quality agglomeration algorithm for the virtual element method applied to discrete fracture networks. *Calcolo*, 60(2):27, 2023.
- [22] R A Adams and J J F Fournier. *Sobolev spaces*. Pure and Applied Mathematics. Academic Press, Amsterdam, 2 edition, 2003.
- [23] L Ridgway Scott and SC Brenner. *The mathematical theory of finite element methods*. Texts in Applied Mathematics 15. Springer-Verlag, New York, 3 edition, 2008.
- [24] B. Ahmad, A. Alsaedi, F. Brezzi, L.D. Marini, and A. Russo. Equivalent projectors for virtual element methods. *Computers & Mathematics with Applications*, 66(3):376–391, 2013.
- [25] L Beirão da Veiga, F Brezzi, LD Marini, and A Russo. Virtual element method for general second-order elliptic problems on polygonal meshes. *Mathematical Models and Methods in Applied Sciences*, 26(4):729–750, 2016.
- [26] Marco Livesu. cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. *Transactions on Computational Science XXXIV*, pages 64–76, 2019.
- [27] George Karypis and Vipin Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota Digital Conservancy*, 1997.

- [28] Jonathan Richard Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Workshop on applied computational geometry*, pages 203–222. Springer, 1996.
- [29] Hang Si and A TetGen. A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany*, 81:12, 2006.
- [30] Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)*, 39(4):117–1, 2020.
- [31] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. *University of Minnesota Digital Conservancy*, 1997.
- [32] Y Boykov, O Veksler, and R Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

A Supplementary Material

We collect in Tables 7-8 the analogues of Table 2 for datasets *Hex*, *Tri*, *Quad*, and their optimizations with $\mathcal{K} = 40$ and $\mathcal{K} = 20$. In Tables 9-10 we report the analogues of Table 3 for datasets *Hex*₂₀, *Tri*₂₀, and *Quad*₂₀.

Table 7: Analogue of Table 2 for dataset *Hex*. Number of vertices, faces and elements of each mesh; number of internal DOFs for $k = 1, 2, 3$; mesh quality (9).

<i>Dataset</i>	<i>Mesh</i>	\mathcal{T}_h			#DOFs			$\varrho_3(\mathcal{T}_h)$
		$\#\mathcal{V}_h$	$\#\mathcal{F}_h$	$\#\mathcal{T}_h$	$k=1$	$k=2$	$k=3$	
<i>Hex</i>	1	343	756	216	125	1331	3509	0.768
<i>Hex</i>	2	2197	5616	1728	1331	12167	31211	0.773
<i>Hex</i>	3	6859	18468	5832	4913	42875	109025	0.776
<i>Hex</i>	4	15625	43200	13824	12167	103823	262871	0.778
<i>Hex</i>	5	29791	83700	27000	24389	205379	518669	0.780
<i>Hex</i> ₄₀	1	343	439	85	125	873	2102	0.681
<i>Hex</i> ₄₀	2	2191	3395	677	1325	8425	19766	0.677
<i>Hex</i> ₄₀	3	6825	11218	2296	4891	30035	69897	0.675
<i>Hex</i> ₄₀	4	15547	26814	5459	12111	73877	171388	0.681
<i>Hex</i> ₄₀	5	29690	52117	10586	24307	146663	339215	0.679
<i>Hex</i> ₂₀	1	314	285	49	114	639	1467	0.614
<i>Hex</i> ₂₀	2	2022	2266	381	1237	6395	14275	0.609
<i>Hex</i> ₂₀	3	6332	7382	1248	4561	22437	49466	0.605
<i>Hex</i> ₂₀	4	14573	17998	2949	11414	56209	123592	0.607
<i>Hex</i> ₂₀	5	27684	34891	5739	22737	111271	244181	0.605

Table 8: Analogue of Table 2 for datasets *Tri* and *Quad*. Number of vertices, edges and elements of each mesh; number of internal DOFs for $k = 1, 2, 3$; mesh quality (9).

<i>Dataset</i>	<i>Mesh</i>	$\#\mathcal{V}_h$	\mathcal{T}_h		#DOFs			$\varrho_2(\mathcal{T}_h)$
			$\#\mathcal{E}_h$	$\#\mathcal{T}_h$	$k=1$	$k=2$	$k=3$	
<i>Tri</i>	1	100	261	162	64	451	1000	0.900
<i>Tri</i>	2	324	901	578	256	1667	3656	0.897
<i>Tri</i>	3	1156	3333	2178	1024	6403	13960	0.899
<i>Tri</i>	4	4356	12805	8450	4096	25091	54536	0.900
<i>Tri</i>	5	16900	50181	33282	16384	99331	215560	0.900
<i>Tri</i> ₄₀	1	92	153	62	64	251	500	0.735
<i>Tri</i> ₄₀	2	303	515	213	253	931	1822	0.727
<i>Tri</i> ₄₀	3	1098	1905	808	1001	3617	7041	0.744
<i>Tri</i> ₄₀	4	4199	7366	3168	4006	14347	27856	0.750
<i>Tri</i> ₄₀	5	16433	28963	12531	16057	57175	110824	0.751
<i>Tri</i> ₂₀	1	70	101	32	50	163	308	0.621
<i>Tri</i> ₂₀	2	254	368	115	215	659	1218	0.601
<i>Tri</i> ₂₀	3	962	1396	435	883	2635	4822	0.593
<i>Tri</i> ₂₀	4	3717	5406	1690	3559	10497	19125	0.579
<i>Tri</i> ₂₀	5	14535	21190	6656	14241	41793	76001	0.585
<i>Quad</i>	1	81	144	64	49	225	465	0.870
<i>Quad</i>	2	289	544	256	225	961	1953	0.865
<i>Quad</i>	3	1089	2112	1024	961	3969	8001	0.867
<i>Quad</i>	4	4225	8320	4096	3969	16129	32385	0.870
<i>Quad</i>	5	16641	33024	16384	16129	65025	130305	0.870
<i>Quad</i> ₄₀	1	58	82	25	41	131	246	0.589
<i>Quad</i> ₄₀	2	204	302	99	163	523	982	0.650
<i>Quad</i> ₄₀	3	773	1168	396	693	2177	4057	0.663
<i>Quad</i> ₄₀	4	3018	4603	1586	2835	8841	16433	0.667
<i>Quad</i> ₄₀	5	11709	18138	6430	11346	35551	66186	0.680
<i>Quad</i> ₂₀	1	44	55	12	31	85	151	0.311
<i>Quad</i> ₂₀	2	151	201	51	125	351	628	0.442
<i>Quad</i> ₂₀	3	551	754	204	489	1385	2485	0.506
<i>Quad</i> ₂₀	4	2144	2962	819	2021	5679	10156	0.505
<i>Quad</i> ₂₀	5	8503	11778	3276	8256	23063	41146	0.500

Table 9: Analogue of Table 3 for dataset *Hex*. Difference between the number of DOFs (13), the \mathcal{E}_{L^2} error (14), the \mathcal{E}_{H^1} error (15), and the computational time (16) w.r.t. the original mesh.

<i>Dataset</i>	<i>Mesh</i>	<i>k</i>	ΔDOFs	$\Delta\mathcal{E}_{L^2}$	$\Delta\mathcal{E}_{H^1}$	ΔT
<i>Hex</i> ₂₀	1	1	$1.10 \cdot 10^1$	$-9.19 \cdot 10^{-3}$	$-5.01 \cdot 10^{-1}$	$-1.67 \cdot 10^{-1}$
<i>Hex</i> ₂₀	2	1	$9.40 \cdot 10^1$	$-4.87 \cdot 10^{-3}$	$-2.19 \cdot 10^{-1}$	$-6.66 \cdot 10^{-1}$
<i>Hex</i> ₂₀	3	1	$3.52 \cdot 10^2$	$-2.95 \cdot 10^{-3}$	$-1.36 \cdot 10^{-1}$	$-2.47 \cdot 10^0$
<i>Hex</i> ₂₀	4	1	$7.53 \cdot 10^2$	$-1.85 \cdot 10^{-3}$	$-9.20 \cdot 10^{-2}$	$-6.18 \cdot 10^0$
<i>Hex</i> ₂₀	5	1	$1.65 \cdot 10^3$	$-1.27 \cdot 10^{-3}$	$-6.88 \cdot 10^{-2}$	$-1.65 \cdot 10^1$
<i>Hex</i> ₂₀	1	2	$6.92 \cdot 10^2$	$-4.12 \cdot 10^{-3}$	$-1.89 \cdot 10^{-1}$	$-2.15 \cdot 10^{-1}$
<i>Hex</i> ₂₀	2	2	$5.77 \cdot 10^3$	$-3.81 \cdot 10^{-4}$	$-3.64 \cdot 10^{-2}$	$-6.14 \cdot 10^{-1}$
<i>Hex</i> ₂₀	3	2	$2.04 \cdot 10^4$	$-8.92 \cdot 10^{-5}$	$-1.63 \cdot 10^{-2}$	$+3.32 \cdot 10^0$
<i>Hex</i> ₂₀	4	2	$4.76 \cdot 10^4$	$-3.22 \cdot 10^{-5}$	$-9.26 \cdot 10^{-3}$	$+6.95 \cdot 10^1$
<i>Hex</i> ₂₀	5	2	$9.41 \cdot 10^4$	$-1.57 \cdot 10^{-5}$	$-6.04 \cdot 10^{-3}$	$+1.52 \cdot 10^2$
<i>Hex</i> ₂₀	1	3	$2.04 \cdot 10^3$	$-4.70 \cdot 10^{-4}$	$-4.71 \cdot 10^{-2}$	$-3.44 \cdot 10^{-1}$
<i>Hex</i> ₂₀	2	3	$1.69 \cdot 10^4$	$-2.13 \cdot 10^{-5}$	$-5.66 \cdot 10^{-3}$	$+1.29 \cdot 10^0$
<i>Hex</i> ₂₀	3	3	$5.96 \cdot 10^4$	$-4.48 \cdot 10^{-6}$	$-1.82 \cdot 10^{-3}$	$+6.16 \cdot 10^1$
<i>Hex</i> ₂₀	4	3	$1.39 \cdot 10^5$	$-1.37 \cdot 10^{-6}$	$-7.66 \cdot 10^{-4}$	$+1.86 \cdot 10^2$
<i>Hex</i> ₂₀	5	3	$2.74 \cdot 10^5$	$-5.75 \cdot 10^{-7}$	$-4.03 \cdot 10^{-4}$	$+7.98 \cdot 10^2$

Table 10: Analogue of Table 3 for datasets *Tri* and *Quad*. Difference between the number of DOFs (13), the \mathcal{E}_{L^2} error (14), the \mathcal{E}_{H^1} error (15), and the computational time (16) w.r.t. the original mesh.

<i>Dataset</i>	<i>Mesh</i>	<i>k</i>	ΔDOFs	$\Delta\mathcal{E}_{L^2}$	$\Delta\mathcal{E}_{H^1}$	ΔT
<i>Tri</i> ₂₀	1	1	$1.40 \cdot 10^1$	$-2.76 \cdot 10^{-2}$	$-1.07 \cdot 10^{-1}$	$-1.42 \cdot 10^{-2}$
<i>Tri</i> ₂₀	2	1	$4.10 \cdot 10^1$	$-6.57 \cdot 10^{-3}$	$-4.86 \cdot 10^{-2}$	$-4.25 \cdot 10^{-2}$
<i>Tri</i> ₂₀	3	1	$1.41 \cdot 10^2$	$-2.10 \cdot 10^{-3}$	$-2.82 \cdot 10^{-2}$	$-2.66 \cdot 10^{-1}$
<i>Tri</i> ₂₀	4	1	$5.37 \cdot 10^2$	$-4.85 \cdot 10^{-4}$	$-1.35 \cdot 10^{-2}$	$-1.56 \cdot 10^0$
<i>Tri</i> ₂₀	5	1	$2.14 \cdot 10^3$	$-1.22 \cdot 10^{-4}$	$-6.66 \cdot 10^{-3}$	$-6.45 \cdot 10^0$
<i>Tri</i> ₂₀	1	2	$2.88 \cdot 10^2$	$-1.91 \cdot 10^{-3}$	$-2.93 \cdot 10^{-2}$	$+7.50 \cdot 10^{-4}$
<i>Tri</i> ₂₀	2	2	$1.01 \cdot 10^3$	$-2.76 \cdot 10^{-4}$	$-8.19 \cdot 10^{-3}$	$+4.90 \cdot 10^{-3}$
<i>Tri</i> ₂₀	3	2	$3.77 \cdot 10^3$	$-4.72 \cdot 10^{-5}$	$-2.52 \cdot 10^{-3}$	$-1.69 \cdot 10^{-1}$
<i>Tri</i> ₂₀	4	2	$1.46 \cdot 10^4$	$-7.76 \cdot 10^{-6}$	$-6.73 \cdot 10^{-4}$	$-1.55 \cdot 10^0$
<i>Tri</i> ₂₀	5	2	$5.75 \cdot 10^4$	$-7.70 \cdot 10^{-7}$	$-1.56 \cdot 10^{-4}$	$-6.41 \cdot 10^0$
<i>Tri</i> ₂₀	1	3	$6.92 \cdot 10^2$	$-2.55 \cdot 10^{-4}$	$-4.09 \cdot 10^{-3}$	$+2.44 \cdot 10^{-2}$
<i>Tri</i> ₂₀	2	3	$2.44 \cdot 10^3$	$+1.30 \cdot 10^{-3}$	$-6.45 \cdot 10^{-4}$	$+8.22 \cdot 10^{-2}$
<i>Tri</i> ₂₀	3	3	$9.14 \cdot 10^3$	$+6.13 \cdot 10^{-6}$	$-8.70 \cdot 10^{-5}$	$-3.59 \cdot 10^{-1}$
<i>Tri</i> ₂₀	4	3	$3.54 \cdot 10^4$	$+3.04 \cdot 10^{-3}$	$-1.18 \cdot 10^{-5}$	$-1.55 \cdot 10^0$
<i>Tri</i> ₂₀	5	3	$1.40 \cdot 10^5$	$+1.98 \cdot 10^{-5}$	$-1.40 \cdot 10^{-6}$	$-6.26 \cdot 10^0$
<i>Quad</i> ₂₀	1	1	$2.00 \cdot 10^1$	$-6.14 \cdot 10^{-2}$	$-2.80 \cdot 10^{-1}$	$-8.80 \cdot 10^{-3}$
<i>Quad</i> ₂₀	2	1	$1.00 \cdot 10^2$	$-1.47 \cdot 10^{-2}$	$-1.15 \cdot 10^{-1}$	$-5.68 \cdot 10^{-2}$
<i>Quad</i> ₂₀	3	1	$4.72 \cdot 10^2$	$-4.68 \cdot 10^{-3}$	$-6.43 \cdot 10^{-2}$	$-1.25 \cdot 10^{-1}$
<i>Quad</i> ₂₀	4	1	$1.95 \cdot 10^3$	$-1.11 \cdot 10^{-3}$	$-3.17 \cdot 10^{-2}$	$-5.09 \cdot 10^{-1}$
<i>Quad</i> ₂₀	5	1	$7.87 \cdot 10^3$	$-2.89 \cdot 10^{-4}$	$-1.64 \cdot 10^{-2}$	$-3.54 \cdot 10^0$
<i>Quad</i> ₂₀	1	2	$1.44 \cdot 10^2$	$-6.03 \cdot 10^{-3}$	$-8.03 \cdot 10^{-2}$	$-1.82 \cdot 10^{-3}$
<i>Quad</i> ₂₀	2	2	$6.10 \cdot 10^2$	$-8.00 \cdot 10^{-4}$	$-1.95 \cdot 10^{-2}$	$-9.10 \cdot 10^{-3}$
<i>Quad</i> ₂₀	3	2	$2.58 \cdot 10^3$	$-8.81 \cdot 10^{-5}$	$-4.70 \cdot 10^{-3}$	$+1.66 \cdot 10^{-3}$
<i>Quad</i> ₂₀	4	2	$1.05 \cdot 10^4$	$-1.13 \cdot 10^{-5}$	$-1.20 \cdot 10^{-3}$	$-8.68 \cdot 10^{-1}$
<i>Quad</i> ₂₀	5	2	$4.20 \cdot 10^4$	$-1.61 \cdot 10^{-6}$	$-3.16 \cdot 10^{-4}$	$-3.47 \cdot 10^0$
<i>Quad</i> ₂₀	1	3	$3.20 \cdot 10^2$	$-8.94 \cdot 10^{-4}$	$-1.22 \cdot 10^{-2}$	$+8.12 \cdot 10^{-3}$
<i>Quad</i> ₂₀	2	3	$1.33 \cdot 10^3$	$-5.00 \cdot 10^{-5}$	$-1.52 \cdot 10^{-3}$	$+6.39 \cdot 10^{-2}$
<i>Quad</i> ₂₀	3	3	$5.52 \cdot 10^3$	$-2.78 \cdot 10^{-6}$	$-1.85 \cdot 10^{-4}$	$+2.24 \cdot 10^{-1}$
<i>Quad</i> ₂₀	4	3	$2.22 \cdot 10^4$	$-1.90 \cdot 10^{-7}$	$-2.36 \cdot 10^{-5}$	$-8.42 \cdot 10^{-1}$
<i>Quad</i> ₂₀	5	3	$8.92 \cdot 10^4$	$-1.51 \cdot 10^{-8}$	$-3.03 \cdot 10^{-6}$	$-3.23 \cdot 10^0$