Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 00 (2017) 000–000

Procedia Engineering

www.elsevier.com/locate/procedia

26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain

Solving the Maximum Weight Independent Set Problem: Application to Indirect Hexahedral Mesh Generation

Kilian Verhetsel[a,*], Jeanne Pellerin[a], Amaury Johnen[a], Jean-Francois Remacle[a]

[a] *Université catholique de Louvain, iMMC, Avenue Georges Lemaitre 4, bte L4.05.02, 1348 Louvain-la-Neuve, Belgium*

Abstract

Indirect hex-dominant meshing methods rely on the choice of a subset of compatible hexahedra among a large set of candidate hexahedra generated by combining tetrahedra. We propose a new parallel algorithm to choose this subset of compatible hexahedra. Our algorithm computes a near-optimal solution to the Maximum Weight Independent Set problem on the incompatibility graph of the candidate hexahedra. An initial solution computed with a greedy algorithm is iteratively improved by optimizing subgraphs containing up to a few hundred vertices. This procedure uses a branch and bound algorithm and is done in parallel on multiple disjoint subgraphs. First results are presented on large sets of candidate hexahedra and we show that meshes containing up to 10% more hexahedra than greedy methods can be computed within a few seconds.

Peer-review under responsibility of the scientific committee of the 26th International Meshing Roundtable.

Keywords: Indirect meshing; Hex-dominant meshes

1. Introduction

Recently, promising methods to automatically generate hex-dominant meshes on general domains have been proposed [1–5]. Their principle is to combine elements of a tetrahedral mesh T into hexahedra. These indirect methods first compute the largest set H of hexahedra that can be constructed by combining tetrahedra of T. This set H has a typical size of 10 to 40 times the number of vertices of T [1]. The second step is to choose the hexahedra of the final mesh, that is a subset $H_C \subset H$. To have a valid output mesh all selected hexahedra should be compatible, i.e. the intersection between any two hexahedra of H_C should be empty or equal to a facet, edge, or vertex shared by the two hexahedra. This selection procedure can be reformulated as a Maximum Weight Independent Set (MWIS) problem on the graph G that has one vertex per hexahedron and one edge linking each pair of incompatible hexahedra. An independent set is a subgraph of G where no vertex is connected to another vertex, in other words, where no hexahedron is incompatible with another hexahedron. The vertices of G are furthermore weighted by the quality of the corresponding hexahedron to select hexahedra of the best possible quality.

In the general case, the problem of the Maximum Weight Independent Set (MWIS) is NP-Hard. We introduce a new algorithm that is well suited to compute a good solution of this MWIS problem for incompatiblity graphs

* Corresponding author. Tel.: +32 10472355; fax: +32 10472999.
E-mail address: Kilian.Verhetsel@uclouvain.be

Peer-review under responsibility of the scientific committee of the 26th International Meshing Roundtable.

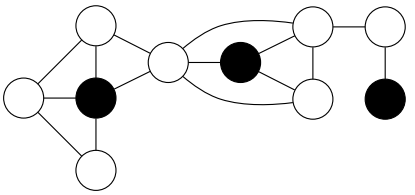
of potential hexahedra generated by combining elements of a tetrahedral mesh. Starting from an initial solution, our method iteratively improves the best known solution by optimizing small disjoint subgraphs. Solutions on the subgraphs are computed in parallel with an existing branch and bound algorithm [6]. First results are presented on large sets of candidate hexahedra generated with the method presented in [1] and we show that within a few seconds we can compute meshes that contain up to 10% more hexahedra than greedy methods.

2. Algorithm Overview

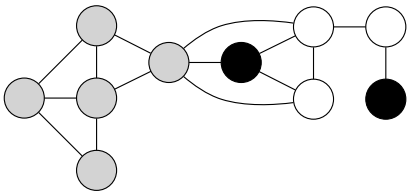
The input of our algorithm is the incompatibility graph $G = (V, E)$ of the set of hexahedra generated by the method proposed in [1]. This weighted graph has one vertex per hexahedron, the weight of the vertex being the quality of this hexahedron, and one edge linking each each pair of incompatible hexahedra. Our algorithm outputs an independent set S of G, *i.e.* a set of vertices such that no two vertices in S are adjacent in G, for which the sum of the weight of the vertices in S is maximal. Our strategy is to iteratively improve an initial solution by optimizing subgraphs containing up to a few hundred vertices.

The initial solution is computed using a greedy approach. First, a list of all vertices sorted by descending weight is built. Then, the initial solution is built by iteratively adding the best vertex compatible with all previously added vertices. The algorithm then iteratively improves this greedy solution by re-optimizing small subgraphs of G (this is a special case of the large neighborhood search proposed for other problems [7,8]). Each subgraph contains vertices which are close to each other. This strategy is motivated by the fact than the way one tetrahedron is combined with its neighbors into a hexahedra has almost no impact on the combination of far away tetrahedra. At each iteration the following steps are performed (Figure 1):

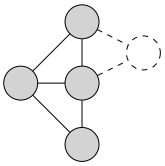
1. A subset X of the vertex set V, called fragment, is computed by selecting the n nearest neighbors of a randomly picked vertex (Figure 1b). The larger the size of the fragment, n, the better the quality of the solution, but the higher the cost of each iteration. The size n can thus be adjusted automatically to perform a target number of iterations N within a given time limit.
2. Remove all the vertices of the fragment that are adjacent to a vertex belonging to the current solution but not to the fragment (Figure 1c). These vertices are connected to one vertex of the solution and are not to be considered at this iteration.
3. Compute the optimal solution of the MWIS problem on the subgraph of G defined by the vertices of the fragment (Figure 1d). The computation of this set can be performed using any exact algorithm for the MWIS problem. Our implementation uses the one of [6].



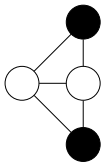
(a) Current solution



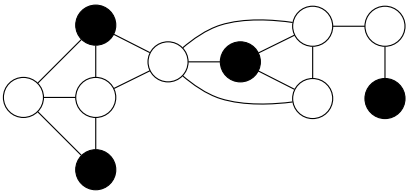
(b) Fragment selection



(c) Removal of neighbors of the solution



(d) Optimal solution on the fragment



(e) Solution update

Fig. 1: An iteration of the algorithm

4. Update the solution (Figure 1e). The state of all vertices that are not part of the fragment is left unchanged, while the state of the vertices of the fragment is updated to the exact solution computed during the previous step.

One advantage of this approach is that it is possible to adapt the algorithm for parallel architectures by performing the last steps in parallel on multiple fragments. In order to ensure the validity of the solution regardless of the fragment subset added to the solution, fragments explored in parallel should neither overlap nor be connected. This is ensured using one main thread to generate fragments and multiple worker threads to optimize fragments. A concurrent queue is used to allow communication between these threads. The algorithm guarantees that all fragments that are being explored by a worker thread or in the queue are not connected by any edge. This is managed by the main thread during fragment generation by ignoring conflicting vertices.

3. Results and Discussion

We evaluate our algorithm on different sets of candidate hexahedra generated with the method proposed in [1] with minimum quality of 0. An Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz with 16 GB of RAM was used to test our method. We compare the number of selected hexahedra obtained with our algorithm with the number of hexahedra obtained with the greedy algorithm (starting point of our method). The results obtained on 4 meshes are shown in Table 1. Compared to greedy methods, the number of tetrahedra combined into hexahedra is significantly increased. A hex-dominant mesh obtained by selecting the hexahedra with a greedy methods is compared on Figure 2 to a mesh obtained by selecting the hexahedra with our algorithm.

Table 1: Our method (time limit: 60 s) produces between 5 and 10% more hexahedra than the greedy algorithm. Models are the same as in [1].

Model	Input tetrahedral mesh			# hex in output mesh		# tet in output mesh	
	# vertices	# tet	# potential hex	Greedy	Our method	Greedy	Our method
CrankShaft	23,245	104,302	306,710	9,154	10,355	44,337	36,520
Fusee	11,975	50,750	145,918	5,225	5,707	16,819	13,768
Caliper	130,572	675,289	3,491,410	85,962	89,923	151,556	127,071
Fusee_1	71,947	349,893	1,697,116	47,501	49,510	60,980	48,484

In our algorithm, the size of fragments is dynamically adjusted in order to perform a given number of iterations within a given time. To show the influence of this size, we compare in Figure 3 the quality of the solution obtained when fixing the fragment size. We can see that there exists an optimal fragment size. This is induced by better solutions computed when the size of fragments is larger, but at the cost of an exponentially higher time to explore them. This optimal fragment size increases when the time limit is increased. The choice to dynamically adjust the fragment size was made because this optimal value varies for different input meshes and different time limits and cannot be easily predicted.

To study the strong scaling of our algorithm, we fix the fragments size at 380 and the number of iterations to 25,000. Figure 4 presents the strong scaling on CrankShaft mesh for a number of threads going from 1 to 24. Each time point is the median of 100 runs. For this test case, our algorithm scales linearly up to 12 threads. Since a single thread generates the fragments, the computation time reaches a plateau for a higher number of threads.

Since the optimal solution is always computed on the fragment subgraphs, the search is poorly diversified. An alternative could be to optimize the solution in fragments using local search algorithms [8]. This would also lower the cost of increasing the fragment size as the computation timings of the optimal solution increase exponentially with the fragment size. In terms of performances, the algorithm can still be improved in a number of ways in order to better make use of systems with a few dozen cores or more. As observed above the performances of the current algorithm are limited by how quickly one thread generates all fragments (Figure 4). This could be solved by allowing multiple producer threads to run in parallel. Such a change would require a strategy to ensure that all producers generate fragments that can be explored in parallel, or at least a way of dealing with conflicts.

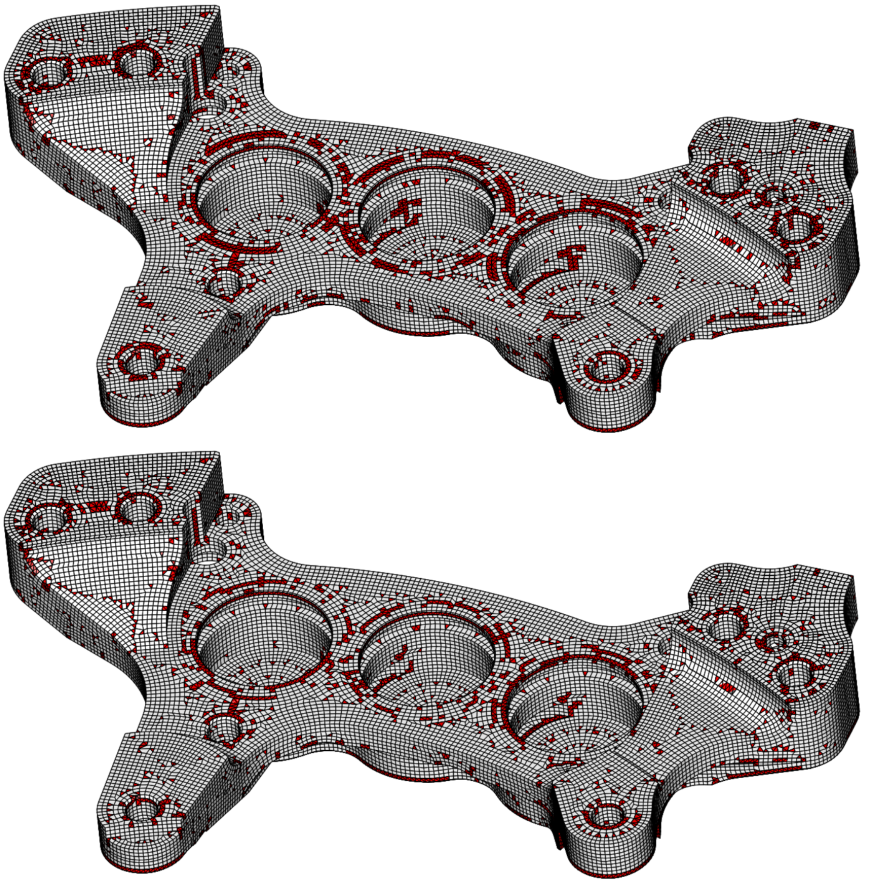


Fig. 2: Meshes produced for the Caliper model using a greedy method (top) and our algorithm (bottom). We obtain 4.6% more hexahedra in 60 s.

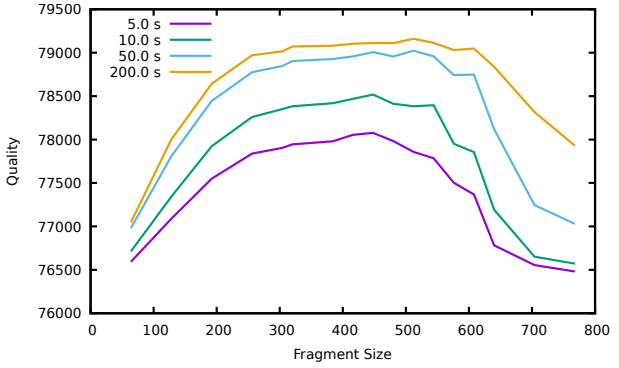


Fig. 3: Quality of the solution for the Caliper model with variable fragment sizes and time limits

References

[1] J. Pellerin, A. Johnen, J.-F. Remacle, Combining tetrahedra into hexahedra: a vertex based strategy, ArXiv e-prints (2017).
[2] D. Sokolov, N. Ray, L. Untereiner, B. Lévy, Hexahedral-dominant meshing, ACM Transactions on Graphics (TOG) 35 (2016) 157.
[3] A. Botella, B. Lévy, G. Caumon, Indirect unstructured hex-dominant mesh generation using tetrahedra recombination, Computational Geosciences 20 (2016) 437–451.
[4] S. Meshkat, D. Talmor, Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh, International

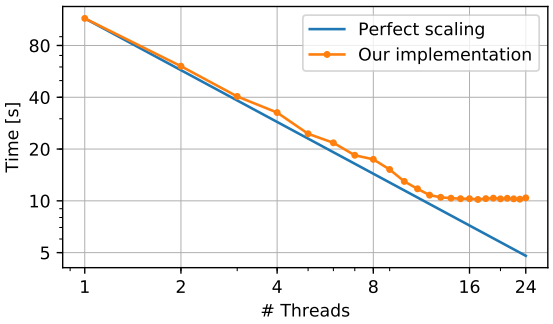


Fig. 4: The scaling of our current parallel implementation is limited by the single thread generation of the fragments.

Journal for Numerical Methods in Engineering 49 (2000) 17–30.

[5] S. Yamakawa, K. Shimada, Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells, International journal for numerical methods in engineering 57 (2003) 2099–2129.

[6] D. Kumlander, A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search, in: Proc. 5th Intl Conf. on Modelling, Computation and Optimization in Information Systems and Management Sciences, Citeseer, 2004, pp. 202–208.

[7] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in: International Conference on Principles and Practice of Constraint Programming, Springer, 1998, pp. 417–431.

[8] D. Pisinger, S. Ropke, Large neighborhood search, in: Handbook of metaheuristics, Springer, 2010, pp. 399–419.