# Graph Partitioning for the Finite Element Method:

## Reducing Communication Volume with the Directed Sorted Heavy Edge Matching

**Dissertation**

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

**José Luis González García**
aus Zacatecas, Mexiko

Göttingen, 2019

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Stephan Waack
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Andrei Tchernykh
Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California (CICESE)

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH,
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Stephan Waack
Institut für Informatik, Georg-August-Universität Göttingen

2. Korreferent: Prof. Dr. Andrei Tchernykh
Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Dieter Hogrefe
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr.-Ing. Delphine Reinhardt
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr.-Ing. Marcus Baum
Institut für Informatik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 2 Mai, 2019

# Acknowledgments

Many people have been involved, in one way or another, during my PhD studies. They have made easier, or even possible, the achievement of this work and it is difficult to fully express my thankfulness. I am sincerely grateful to all of them.

I am very thankful to my supervisor, Prof. Dr. Ramin Yahyapour, for giving me the opportunity, support and advise to complete this work. It has been invaluable his leadership and supervision during my time in Dortmund and Göttingen. I would also like to thank Dr. Andrei Tchernykh for providing advice as well as supervising my work. They are the main source of ideas that guided my research work. Also Dr. Philipp Wieder has been important during my time in Germany helping me when I needed it.

I would like to express my deep gratitude to the *Consejo Nacional de Ciencia y Tecnología* (CONACYT) in Mexico for its financial support through the grant 309370. This work would not have been possible without its essential help. Also important is the *Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen* (GWDG) which gave me the opportunity to finish my research work.

To my spouse 小文 for her patience, understanding, support and love among the hard times. And, last but not least, to my family in Mexico who supported me during the time I have been away from home; they are my inspiration and strength in life. To my beloved parents, José Luis and Rebeca, for their guidance and teaching me how important life is.

# Abstract

A technique called the Finite Element Method is primarily utilized to numerically solve Partial Differential Equations, most commonly by the use iterative methods, over a compact domain. The partial differential equations domain is represented by a mesh of information which needs to be distributed among all available processors or cores in a parallel computer. Distributing the mesh, known as the mesh partitioning problem, is NP-complete. Much effort focuses on graph partitioning and parallelization to address it.

An increasing variety of general purpose techniques and libraries has been and is being developed in recent time, many of which provide great effectiveness. However, the load balancing of the mesh is still an open problem; newer and larger simulations bring new requirements into play. These techniques have to scale linearly on large clusters of hundreds of thousands of processors. They have to be resource aware and take into consideration the heterogeneity of current processors and network infrastructures in the partitioning process. Equal size meshes, provided by traditional partitioning methods, no longer fulfill the main goals.

New enhancements to existing libraries and algorithms are required to support even more complex applications and the constantly evolving hardware architectures. In this work, we give an overview of current graph partitioning techniques used on large-scale parallel machines for load balancing of finite element computations. We introduce a new vertex matching model called Directed Sorted Heavy Edge Matching to reduce the communication volume during FEM simulations and ensure efficient execution on a distributed system. Finally, we provide performance analysis of the proposed model and comment on its benefits.

# Contents

# Part I.

# Introduction

# Chapter 1.

# Background and Motivation

The Finite Element Method (FEM), or Finite Element Analysis[1] (FEA), is a technique used in different areas of scientific computation such as engineering and physics. It helps to analyze the behavior of real life objects, or physical phenomena, under different environmental conditions; i.e., heat, mechanical stress, vibrations, etc. To provide solutions with high accuracy, current FEM applications rely on large computational, memory and communication requirements; hence parallel systems are extensively used for FEM applications. The efficiency of parallel FEM applications is heavily affected by its dynamic nature and the load balancing problem must be addressed. One effective way to approach it is through graph partitioning; the main focus of the research presented in this thesis.

This chapter gives a brief overview of the FEM and its applications, the use of parallel systems to address the intractability of current problems, how dynamic problems affect the efficiency of parallel implementations, how the current load balancing methods motivated this work, and finally, our approach to address the problem.

## 1.1. General Overview

The FEM, or FEA, is widely used in engineering and physics, among other disciplines. For example, a civil engineer can analyze how a bridge reacts under load or specific physical conditions. Base on this analysis, problems in the design can be identified and corrected before the actual construction begins; it prevents catastrophic failures during the service life of the bridge. Many other examples of applications can be cited such as the design of automobiles, aircrafts, buildings, etc. The FEA is of vital importance whether it used by a private company, government contractor or a scientist; it brings important benefits such as lowering design and manufacture costs. For example, the production of a new aircraft can bring a company into bankruptcy if the design has flaws; most importantly, the cost in lives could be high. FEA is a cost-effective way to ensure that the design of a new product is ready for manufacture.

The accuracy of the numerical solutions, provided by the FEM applications, depends on the discretization of the model; and with higher accuracy, higher processing power is required. This situation makes sequential implementations useless in practice and parallel systems come into play, but they bring new challenges in terms of efficiency [1]. When parallel systems are employed, efficiency becomes an important concern; an efficient system reduces costs and time.

The load balancing problem needs to be addressed in order to improve the efficiency of a parallel system. One effective approach is through graph partitioning where the load is modeled as a graph. The

[1] It is commonly known as the practical application of the Finite Element Method.

graph is then partitioned and mapped into the processors; it defines the distribution of the load in the parallel system. It is a more difficult task with dynamic problems as redistribution must be performed regularly when the imbalance reaches a certain threshold.

With the availability of hundreds of thousands of processors becoming cheaper every day, the load balancing problem shifts its focus to communication costs. The new technologies bring new challenges in terms of efficiency and current solutions become obsolete. We address the load balancing problem through graph partitioning, with a particular emphasis on the reduction of communication volume, to improve the efficiency of parallel FEM applications.

### 1.1.1. FEA and FEM

In FEM applications, the Partial Differential Equations (PDEs) are used to describe the problem. Its domain is discretized resulting in a mesh of information (triangles or rectangles for two dimensional objects, tetrahedra or hexahedra for three dimensional objects). Based on the elements of the mesh, the PDEs are then converted into a system of linear equations [2]. In general, iterative methods are employed to solve that linear system [3], [4]. The accuracy of the discretization influences the subsequent solution and its quality; the elements of the mesh need to be small enough to produce precise approximations. An extremely fine discretization may incur in extra computation, communication and memory costs. Adaptive techniques were introduced to mitigate this issue by allowing the solution error to be within certain limits while the costs are minimized [5].

Usually the parallel FEM simulations are divided into three main steps which are then repeated several times until the end of the simulation. The PDEs are solved during the computational step. According to the results, the mesh is refined in areas where needed. The refinement produces an imbalance in the system which is then reduced in the next step. Once the load has been balanced, the system is ready to perform the next computational step.

### 1.1.2. Parallelization of Numerical Simulations

FEA is a typical example of an application for High Performance Computing (HPC) systems. To provide solutions with high accuracy, current FEM applications rely on large computational, memory and communication requirements; making sequential implementations generally useless in practice. The introduction of parallel systems helps overcome this limitation; though, efficiency concerns become important. The efficiency of parallel applications is defined by the distribution of the mesh (the load) and the communication overhead among all subdomains.

The PDEs domain is represented by a mesh of information which has to be distributed amongst all available processors in the parallel system. It generally employs an iterative approach to approximate the solution. Then, multiple processors execute the same code on different mesh elements to compute the final solution. With dynamic problems, the refinement step introduces an imbalance to the system and the mesh must be redistributed to keep the efficiency within an acceptable range. It is a difficult task to keep the workload in balance since it is not possible to know in advance what regions will be refined. The mesh partitioning problem is known to be NP-complete [6]–[8].

### 1.1.3. Load Balancing through Graph Partitioning

Load balancing is essential in parallel computations to improve its efficiency. Shivaratry et al. [9] describe and compares some common strategies. Due to the fact that the mesh of information can be characterized by a graph, as depicted in Figure 1.1, much effort focuses on graph partitioning algorithms to address the load balancing problem of parallel FEM simulations. Graph partitioning algorithms generate arrays of information containing the location for every graph vertex; i.e., what mesh element is assigned to which processor, see Figure 1.2. In addition, different types of graphs can be used according to the requirements of the problem.

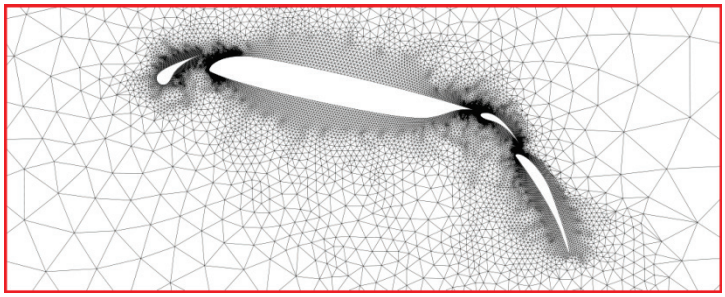

**Figure 1.1. Graph of an airfoil with flaps.**

An increasing variety of general purpose techniques and libraries has been, and is being, developed in recent time which provides great effectiveness; we refer the reader to the work by Buluç et al. [10] and Fjällström [11] for more information. However, the redistribution of the mesh is still an open problem; newer and larger simulations bring new requirements into play. These techniques have to scale on clusters of hundreds of thousands of processors. They have to be resource aware and take into consideration the heterogeneity of current processors and network infrastructures in the partitioning process.

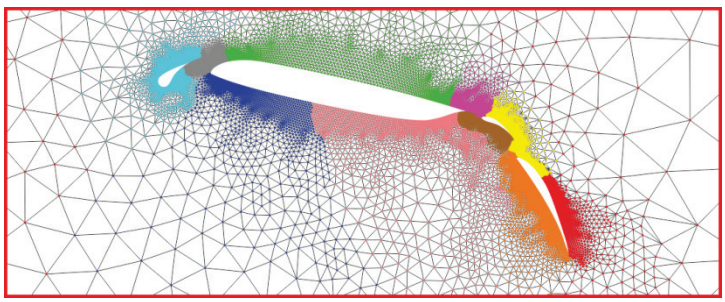

**Figure 1.2. Graph of an airfoil with flaps partitioned into 10 subdomains, each identified by a different color.**

### 1.1.4. Challenges

Emerging applications and hardware architectures become more complex and heterogeneity shall be considered to improve efficiency. It has been established that current methods which optimize only the size of subdomains and the edge cut do not fulfill current requirements and the efficiency can drop drastically [12]. New techniques are needed to reduce the time spend on FEM simulations [13], [14]. Enhancements to existing libraries and algorithms are required to support more complex applications and the constantly evolving hardware architectures. Thus, the advantages brought by new HPC technologies will never be fully exploited unless efficient load balancing techniques are applied.

The reduction of communication volume, while keeping the balance in the partition, will reduce the overall runtime of parallel FEM computation. The transmission of data over network links is considerable slower than data processing. A trade between communication and computation is required to increase the efficiency of parallel computations [15], [16]. As stated by Jensen [17], "*the most critical system control mechanisms in a distributed computer are clearly those involved with interprocess and interprocessor communication"*. Over the years, several techniques propose to overlap the computation and communication in an attempt to hide overhead brought by the *slow* network links [18]–[22]. Others consider the hardware design and architecture to reduce the communication during parallel computations [23]–[25], just to mention a few.

### 1.1.5. Current Trends

Often, FEM libraries are designed for relatively small systems. When hundreds of thousands of processors are available their design becomes an important limitation in order to scale accordingly. This situation leads to an important inequality between the software and hardware; which translates into a decrease of efficiency. Heister et al. [26] propose new parallel data structures and adapted algorithms to benefit from large clusters during immensely parallel computations. They have improved the library deal.II to deal with the problem by addressing the most important bottlenecks of scalability: handling the mesh of information, the numerical linear algebra, and finally the degrees of freedom, its global numbering and distribution. ALPS [27] is another example of a library developed with massively parallel simulations in mind. ALPS is based on an existing library called p4est [28], but with some drawbacks compared to deal.II: the lack of the comprehensive infrastructure support, and not being publicly available.

During the simulation, some regions of the mesh may be refined or coarsened between computational steps to increase the accuracy of the solution. Since the regions are not known beforehand, or can vary over the course of the simulation, the mesh changes unpredictable during the computations. After this refinement or coarsening process, the workload may become unbalanced and the imbalance has to be corrected. The load balancing step could incur in a large cost, it is performed only when the imbalance is high and its benefits overcomes its cost. Thus, it is of great importance to accurately determine the impact of the new imbalance on the simulation to decide if the mesh should be redistributed to increase the overall performance. Olas et al. [29] have introduced a dynamic load balancer to the existing library NuscaS [30] which includes a performance model of their own. The model accurately estimates the cost, measured in time, of every load balancing and computational step with or without a balanced workload.

Many parallel systems are made as a collection of shared memory multiprocessors with an intricate and heterogeneous interconnection. One typical example is Grid computing which is a geographically dispersed system working together to perform large tasks. This distributed system introduces new and important challenges in resource management due to the heterogeneous nature. To efficiently allocate the data on these systems, new generation of load balancers requires being aware of the available resources. In other words, they have to consider the heterogeneity of the hardware employed during the simulations. Some attempts to address this issue are presented in [31]–[35].

## 1.2. Motivation

Current FEM applications heavily depend on large computational, memory and communication costs to provide high accurate solutions. Parallel systems help overcome the limitation of sequential implementations by bringing together the computational power of hundreds of thousands of processors. Emerging applications and new hardware architectures define the new challenges in efficiency of parallel applications. There are still many open problems in load balancing for parallel applications; the study and development of new techniques is essential to fully exploit the new HPC technologies.

### 1.2.1. Opportunities

In FEM simulations, when the quality of the solution is not within the acceptable limits, the density of the mesh of information must increase. The size of the elements in the mesh heavily impacts the precision of the approximation. Although this approach improves the accuracy of the final solution; it is not appropriate for all scenarios as the computational power required to compute the solution grows accordingly. Adaptive dynamic meshes help overcome this problem by keeping the processing cost at a minimum while providing accurate solutions. In some cases, the efficiency of the parallel application drops drastically with this type of meshes. The new generation of load balancers must follow the same philosophy: being adaptive and dynamic. Static counterparts no longer fulfill the requirements for parallel FEM applications on current HPC systems.

Traditionally, the transmission of data over network interconnects is slow. Emerging technologies have been addressing this issue and have improved the speed. Nonetheless, communication continues to be slow with respect to the computing power. The new generation of load balancers must address this condition while making the decision on when and how the load should be distributed. It is not enough to have a perfect balance of the workload if it implies a large migration of elements between computational steps of the simulation; communication dependencies also define the overall efficiency of a parallel application. The load balancer must also take into account the underlying hardware architecture in order to better distribute the load along all available processors in the most efficient way. In the design of a comprehensive, adaptive and dynamic load balancer, all levels of the load balancing process should be considered. Another aspect to consider in the design of an effective load balancer is scalability. Many of the approaches are based on legacy methods which do not consider the current advances in HPC technologies. While adequate for small implementations, they do not scale well to systems of hundreds of thousands or processors in heterogeneous hardware architectures.

### 1.2.2. Problem Statement

The problem in question is simple, how to distribute the workload of parallel FEM simulations to improve the efficiency; and hence, to reduce the execution time. The solution, however, is not straight forward. Every use case is unique and the requirements differ from one to the other; one solution does not fit all. Why an appropriate balance of the workload is important in parallel applications? The overall efficiency of a parallel system during the execution of a simulation is closely related to monetary costs; the longer the execution time the higher the costs. With an adequate load balancer, the efficiency of the

system can be largely improved reducing costs. In an ideal word, where the load balancer generates perfect distributions of the load among the available processors, it would be possible to reduce the processor idle time to zero and obtain the results in the shortest time possible. In reality, the large number of conflicting requirements, and the nature of the problem, make impossible to solve the load balancing problem at all.

Traditional methods do not fulfill current requirements, but they are still in use. The reason is simple; they are *fast* and *good* enough. Several approaches have been proposed over the years; however they were conceived in a time when massive amounts of resources were not available. They do not scale well and generate poor distributions when hardware heterogeneity is part of the equation. This brings us to ask another question. If previous approaches do not work well with current systems, what should be considered in the design of a load balancer? And the answer is not simple. Different use cases have different requirements; a *perfect* load balancer for a particular parallel application may not be good in other situations. The nature of the parallel application, the input data, the hardware architecture, among many other aspects, should be considered.

It is not trivial to address the problem; several attempts have been made and they focus on particular issues. Many have as central objective to balance the workload of the parallel system, which deals with the original problem. Nonetheless, the redistribution of the load involves transmission of data. A *perfect* distribution of the workload is inefficient if it requires huge amounts of data to be migrated. In addition, communication dependencies among subdomains are important as they also affect the overall efficiency; partial solutions computed by a processor need to be transmitted to neighbors. The *perfect* distribution of the workload is also useless if it requires large amounts of time to compute. A trade between speed and accuracy is essential in the development of a load balancer. This leads us to ask some important questions:

- Can a generic load balancer fulfill the current requirements of FEM applications? If so, how can it be achieved?
- Many FEM applications rely on dynamic meshes to improve the accuracy of the results without incurring in extra computational cost. Can a similar approach be used in the design of a load balancer?

Recent efforts have contributed to develop new techniques designed to address the load balancing problem of parallel FEM applications; many of them by focusing on the graph partitioning problem. The mesh of information in FEM applications can be characterized by a graph and different types can be used for this purpose according to the requirements. We focus our efforts on this direction and identify some key questions that need to be addressed.

- Can a graph partition be improved and is it worth the effort?
- If communication costs are important while generating a distribution of the load, can it be included in the partitioning process of the graph that represents the mesh of information? And how this new objective will affect the partitioning process?

## 1.3. Proposed Approach to the Load Balancing Problem

The central efforts of this work focus on two key approaches: a low level generic strategy, and a multilevel hierarchical load balancer. Both combined can improve the overall efficiency of parallel FEM applications. The first approach is a graph partitioning algorithm with the aim of reducing the communication dependencies among the different subdomains, hence reducing the overall execution time. The second approach, in combination with the previous algorithm, is a comprehensive load balancer which uses idle processors to re-balance the load dynamically during the execution of the application.

### 1.3.1. Graph Partitioning

We propose a new vertex matching algorithm for the multilevel graph partitioning technique. Its aim is to reduce the communication volume by simulating a directed graph. The new model introduces information of the direction of the communication in the matching decisions to reduce the overall communication volume in the final partition. The algorithm takes advantage of the efficient data structures used to represent undirected graphs. Without incurring in extra memory requirements, the algorithm can simulate the direction of the communication within the graph. With the new directional information, the matching phase can improve the final partition of the graph.

### 1.3.2. Load Balancer

We propose a new multilevel hierarchical load balancer model. It is hardware aware and uses graph partitioning techniques. It improves the local load imbalance and reduces the overall execution time of parallel FEM applications. It uses a cost function that includes a comprehensive collection of information to better approximate the computation, communication and load balancing costs. A trade between speed and accuracy is fundamental in the development for the next generation of load balancers. The first level is responsible for the main load balancing steps over the entire system. It is performed within the main load balancing steps during the execution of the FEM application. The cost model defines the current imbalance and if the load balancing step should be performed; sometimes it is more efficient to keep a small imbalance and continue with the next computational step. The second level is in constant execution, monitoring the status of the current computational step. Processors that become available receive loads from neighbors to reduce their idle time. In this way, the workload is redistributed dynamically between the main load balancing steps.

## 1.4. Justification and Scope

It has been established that current methods do not fulfill the new requirements of FEM applications; the efficiency of parallel systems is not ideal. For instance, algorithms which only optimize the size of subdomains and the edge cut do not produce efficient partitions and the efficiency is degraded due to the communication dependencies. New techniques are needed to reduce the time spend on FEM simulations. Therefore, the advantages brought by new HPC technologies will never be fully exploited

unless effective load balancing techniques are applied.

Processing power is becoming cheaper by the day with the technology evolving continuously; and yet, communication costs are still considered relatively high. The transmission of data over network links is considerable slower compared to data processing. It is vital to reduce the communication costs of parallel FEM computations to reduce the overall runtime, and hence improve its efficiency. Over the years, several techniques propose to overlap the computation and communication in an attempt to hide overhead brought by the slow network links. However, this approach only hides the source of the problem. The real communication costs shall be reduced.

### 1.4.1. Delimitations of Scope

One essential aspect of the research that needs to be defined in the first stages is its scope. This work is limited to the load balancing problem on FEM applications with an emphasis on graph partitioning. We use a new multi-level method to compute the partition and compare its performance with other well-known strategies. The analysis is limited to an experimental study to statistically evaluate the performance with the use of different input data. The Scientific Compute Cluster located at Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG) is used for the experimental evaluation. The analysis and comparison with other strategies are based on a collection of synthetic and real life graphs and a set of commonly used metrics. Some real life graphs have been converted from their original format to match that of METIS; however, no essential information is lost or modified. The coordinate information, included with the real life graphs, is only used to create the graphical representations included in this thesis. Although the focus of this work is improving the efficiency of FEM simulations, the proposed algorithm has been not evaluated with a FEM application. Nevertheless, its performance is evident with certain types of graphs and the efficiency of FEM applications is ensured.

### 1.4.2. Key Assumptions

The experimental results reflect the expected performance of the algorithm in real life scenarios; although the experiments are conducted with a limited set of synthetic and real life graphs. This claim arises from the fact that the graphs represent the different situations that may be encounter with diverse use cases. With new technologies developed continuously, processing power is cheaper every day; though, communication costs are still relatively high. Much work has been done to hide it by overlapping computations and communications in FEM applications. The partitions produced by the proposed algorithm reduce the communication volume among the subdomains, hence, reducing the overall execution time of FEM applications by increasing their efficiency. In addition, the comparison with other strategies is limited to those implemented in METIS. However, the improved partitions are an indication of the benefits of using the proposed strategy in different applications. It is also impossible to carry out a comprehensive and uniform evaluation of a wide range of algorithms to compare their performance.

# Chapter 2.

# Contributions and Outline of this Thesis

The main contributions are generic in the sense that the utility of the proposed algorithm encompass a wide variety of applications. It is a general algorithm used to create partitions whilst reducing the communication volume between the different subdomains. It is implemented in METIS [36], [37], a graph partitioning software that is broadly used; for instance, it could be applied to VLSI logic simulations [38]. In addition, the Multilevel Hierarchical Load Balancer presents a new idea on how to approach the load balancing problem in a more effective manner.

This chapter presents a short summary of the main contributions of this research work and outlines the rest of the thesis.

## 2.1. Contributions

As a summary, the contributions of this thesis are listed next:

- An optimized graph partitioning algorithm which reduces the communication volume between subdomains called DSHEM. It includes information of the direction of the communication in the matching decisions to reduce the overall communication volume; the idea behind is simple: to find the vertices which will reduce the amount of data to be transferred. DSHEM takes advantage of the efficient data structures to store the graphs in METIS and mimic the direction of the communication without incurring in extra memory or information from the user. Part of this research work has been submitted to the conference HPCS 2019 where DSHEM is described and evaluated [39].
- The implementation of DSHEM within the popular graph partitioning software METIS; this is of particular importance as described next. The software is free and broadly used around the globe; its efficiency has been established over the years. It is possible for the users to tweak or tune-up the behavior of DSHEM according to their own requirements because METIS is open source. The execution of DSHEM is also configurable by the use of execution parameters sent to METIS.
- A comprehensive experimental analysis of the performance of the new proposed algorithm. The results show that DSHEM improves the communication volume of partitions with specific graph geometries and does not degrade it with the rest. The analysis includes two different approaches of DSHEM: a full (normal) DSHEM and a nested DSHEM partitioning. The description and evaluation of both approaches of DSHEM have been submitted to the conferences HPCS 2019 and PPAM 2019 recently [39], [40].

- A new concept in load balancing for FEM applications: the Multilevel Hierarchical Load Balancer. In combination with DSHEM, the new model can successfully be used as a starting point for a more complex load balancing strategy. This line of research is presented by the author in [13], [14].

### 2.1.1. Publications

- J. L. González García, R. Yahyapour, and A. Tchernykh, "Load balancing for parallel computations with the finite element method," in *3rd International Supercomputing Conference in Mexico*, 2012, p. 9.
- J. L. González García, R. Yahyapour, and A. Tchernykh, "Load balancing for parallel computations with the finite element method," *Comput. y Sist.*, vol. 3, no. 17, pp. 299–316, Sep. 2013.
- J. L. González García, R. Yahyapour, and A. Tchernykh, "Graph Partitioning for FEM Applications: Reducing the Communication Volume with DSHEM (under review)," in *Submitted to HPCS 2019 - The 2019 International Conference on High Performance Computing & Simulation*, 2019.
- J. L. González García, R. Yahyapour, and A. Tchernykh, "Graph Partitioning for FEM Applications: Reducing the Communication Volume with Nested DSHEM (under review)," in *Submitted to PPAM 2019 - 13th International Conference on Parallel Processing and Applied Mathematics*, 2019.

## 2.2. Outline

This thesis is organized in four major parts which are divided in several chapters. Three appendices are also included to provide additional information to some chapters. The next paragraphs provide a short description of the different sections.

- Part I, Introduction, gives a short overview of this work and is divided in two chapters. Chapter 1 introduces the background and challenges. It identifies, as well, the focus of this work and the research problem to address. Chapter 2 presents the contributions of the thesis.
- Part II, Literature Review, provides a detailed background, formally defines the problem and is divided in two chapters. Chapter 3 presents the FEM, introduces the load balancing problem and describes how to address it. Next, Chapter 4 presents a list of the main FEM frameworks and simulators, dedicated load balancing libraries, and the available graph partitioning software.
- Part III, DSHEM and Evaluation, describes the proposed algorithm and presents its experimental evaluation; it is divided in five chapters. Chapter 5 introduces the central idea behind the algorithm DSHEM. Chapter 6 presents the algorithm in detail and its implementation. Chapter 7 describes the methodology used to evaluate DSHEM. Chapter 8 presents its experimental evaluation. Chapter 9 presents experimental evaluation of a variation of the algorithm.

- Part IV, Discussion, concludes the thesis and consists of one chapter. Chapter 10 presents the conclusions of the research questions and problem, a description of the limitations and implications of this research work, and outlines the future perspective introducing a new multilevel load balancer.
- Appendix A, HEM, SHEM and DSHEM, presents a detailed description of the original algorithms in METIS, as well as their evolution over time. It also describes the new algorithm DSHEM and highlights the differences with its predecessor. Appendix B, Graphs, gives a description of the different graphs utilized during the experimental evaluation. It also provides graphical representations and descriptions on how the synthetic graphs are created.

# Part II.

# Literature Review

# Chapter 3.

# The Load Balancing Problem in Parallel FEM Computations

The FEM is a powerful technique widely used to predict the comportment of real life objects subject to specific conditions such as mechanical stresses, heat, vibrations, among others. In practice, the increasing cost of FEM applications, in terms of memory, communication and computations, renders them useless in sequential implementations. To address this problem, parallel systems come into play [1], but they bring as well new challenges in terms of efficiency.

This chapter presents a detailed description of how the load balancing problem for parallel FEM applications is addressed and why it is not efficient. We mainly focus on load balancing through graph/mesh partitioning methods.

## 3.1. FEA and FEM

The FEM, or FEA, is widely used in engineering and physics, among other disciplines, because it is a cost-effective way to detect problems in designs or predict how real life objects react under certain circumstances. A model created with a finite number of discrete elements is adequate in many cases. In other circumstances infinitesimals are used to describe the problem leading to differential equations. To overcome its intractability, different discretization methods have been proposed to calculate a numerical solution. With the increasing processing power of current computers, the solutions become more accurate. Examples of typical areas of application are heat transfer, structural analysis, fluid flow, etc.

Physical phenomena, such as the dissemination of temperature on an object, the propagation of a crack, the air flow around a wing, are usually modeled by PDEs [3], [41]. Unfortunately, they generally do not have an explicit solution making them hard to solve. However, a widely used technique to solve the PDEs is to discretize them into a mesh of information; the FEM adds some degree of smoothness over the original domain. A collection of *cells* (elements or nodes) therefore model the complex geometry of the real object. This discretization generally produces large and sparse matrices suitable for iterative methods such as Multigrid (MG) and Conjugate Gradient (CG) [3], [4].

### 3.1.1. Preconditioners and Solvers

The FEM solver computes a collection of matrix equations to generate an approximation of the phenomena under analysis. The first introduced iterative methods were based on relaxation of the

coordinates like Jacobi, Gauss-Seidel, and SOR [4]. These methods are rarely used in our days. Other techniques utilize a projection process to approximate the solution. The Krylov subspaces methods are considered among the most important techniques. We can mention Arnoldi's Method, CG, Lanczos Biorthogonalization, and Transpose-Free Variants [4], among others. Initially, MG methods were introduced to solve discretized elliptic PDEs. Later, they were enhanced to handle other PDEs problems as well as those not defined by PDEs. The performance of MG methods is superior to that achieved by Krylov subspace methods. However, they differ in an important aspect: MG methods require specific implementations for each problem while Krylov subspace methods are for general purpose. The CG method is suitable for a particular system of linear equations: symmetric positive definite matrices, based on an orthogonal projection onto a Krylov subspace. Iterative methods such as MG and CG are considered among the fastest [42].

As described in [4], preconditioners reduce the complexity of the solution with a given iterative method by introducing implicit or explicit modifications. It conditions the problem in order to make it more suitable for a numerical solution. The convergence of iterative methods increases by reducing the condition number of the matrix. It is the reason why preconditioners are important for Krylov subspace methods.

### 3.1.2. Meshes

The accuracy of the discretization largely impacts the quality of the final solution. The size of the elements of a mesh heavily affects the precision of the approximation. Unfortunately, it is not possible to identify in advance the regions with large gradient. Hence, meshes can be unstructured and periodically refined or coarsened in areas where it is required during calculations; or they can be structured with equal connectivity for each node. Structured meshes are not efficient in all scenarios, but they are much easier to handle. Its problem lies in the unnecessary small elements in those regions where not needed, increasing considerably the cost of the simulation. Obviously, the first variant is preferred and used for FEM; the solution has the same quality but the time needed for the computations is drastically reduced. Adaptive techniques allow keeping the solution error under control while computational costs are minimized [5].

#### Density

The mesh density is an important topic in FEA because of the direct relationship with the solution and the computational costs associated. The accuracy of the solution improves with the increase of the number of elements in the mesh; i.e., its density. As the mesh elements get finer, the solution gets closer to the reality. However, the accuracy comes with a high cost; the computational power that is required to compute the solution increases exponentially with the mesh density, as well as the memory requirements. Several studies classify the mesh generation methods and the impact of its density and shape in the accuracy of the solution [43]–[49], among others. Ghavidel et al. [44] show that an inadequate mesh density affects the reliability of the model's prediction. The authors compared four models with different mesh densities. Ashford and Sitar [45] give an overview of the development of the FEM over the last 30 years.

#### Static and Dynamic Meshes

Static meshes are suitable for many Computational Fluid Dynamics (CFD) simulations where the physical geometry does not change over the course of the simulation. The mesh is generated prior to the computations and stays unchanged until the end of the simulation. However, static meshes are not suitable for all scenarios; certain problems would require a high density mesh, to keep the solution error within the limits, and would render it completely intractable. Adaptive Mesh Refinement (AMR) is a method to dynamically adapt the mesh, within certain regions, during the simulation. These regions are refined when higher precision is required, or coarsened to reduce computational costs. The numerical solution, given a desired accuracy, is calculated with the least amount of work. Different techniques have been proposed in literature over the years to dynamically refine and coarsen the mesh, allowing the solution error to be within certain limits while the costs are minimized [5], [28], [50]–[59]. The advantages of AMR are significant: increase of the accuracy of the solution, reduction in computational and memory costs.

### 3.1.3. Multiphase Problems

For certain applications the mesh elements may belong to more than one phase. Typically these applications arise from multiphysics or contact-impact modeling, and geometric partitioners are often preferred to compute the partitions. This kind of problems consists of various separate phases interrelated (e.g., crash simulations consist of two different phases: the calculation of forces then the contact detection). Frequently, each phase uses separate partitions and data communication is required between them [60].

As data needs to be communicated between phases, computing a single partition well with respect to all phases would reduce or, in the ideal scenario, remove communication. Each processor would have multiple workloads, which correspond to the different phases, making the computation of this single partition more complex. In short, the partitioning is done phase by phase and the computation of the current partition is affected by the results of previous phases [61].

## 3.2. The Problem of Load Balancing in Parallel Computations

The FEM is a powerful technique widely utilized to predict the comportment of real life objects subject to specific conditions such as mechanical stresses, heat, vibrations, etc. [2], [3], [41], [51], [62]. However, current applications have large computational, communication and memory costs to be useful in practice in the form of sequential implementations. Parallel systems allow to overcome this problem by making available a large amount of resources to FEM applications [1], but they bring, as well, new challenges regarding system efficiency.

The efficiency of parallel FEM applications is primarily defined by two important factors: the distribution of the data over all available processors and the communication overhead of the boundary mesh elements. When the mesh is refined and coarsened several times during the computations, the workload changes unpredictably and a redistribution of the mesh may be required. The FEM application has to be interrupted for a load balancing step. This interruption should be as short as possible and the new distribution of the mesh should keep to a minimum the number of elements changing their location.

Efficient load balancing techniques are required to maximize efficiency and fully profit from the potential of HPC systems.

As parallel simulations, as well as environments, change and become more complex, partitioning techniques must be enhanced to fit the emerging requirements. Partitioning algorithms need to be aware of computer architectures, memory and communication resources. Additionally, FEM simulations must scale linearly with the problem size and available resources.

### 3.2.1. Parallelization of Numerical Simulations

FEM is now a typical example of an application for HPC systems due to the number of mesh elements required to obtain accurate approximations. The mesh of information is partitioned and distributed among all processors following the paradigm Single-Program Multiple-Data (SPMD) [41], [63]; multiple processors simultaneously execute the same code on different mesh elements. The mesh partitioning problem is the distribution of the mesh of information amongst all available processors in a parallel system and it was shown to be NP-complete [6]–[8], [64]. As the mesh can be easily represented as a graph, considerably effort focuses on developing proper heuristics based on the graph partitioning problem [65]–[72].

With dynamic problems, some regions of the mesh are refined or coarsened between computational steps. Since these regions are not known in advance, or can vary over time, the mesh changes unpredictable during the computations. This is the main source of imbalance in parallel FEM simulations. Hence, efficient load balancing techniques are essential to reduce the impact of this refinement-coarsening process on the efficiency of computations. It is essential to find new balanced partitions with an additional restriction to prevent too many elements migrating to other processor. Moving mesh elements is usually an expensive operation since huge amounts of information have to travel over the network. A number of solutions have been proposed over the time [73], [74].

#### Factors Leading to Imbalance

The most important causes of load imbalance in parallel FEM applications are the dynamic nature of the problem through time (the cost of computation and communication), and the adaptive refinement or coarsening of the mesh during the simulation. Since it is not possible to know in advance what regions of the mesh will change, it is a difficult task to keep a balanced workload on all processors. The interference from other users in a shared system and the heterogeneity in either the hardware or in the solver can also affect the load balance and performance.

Numerous static and dynamic methods have been developed for load balancing. The dynamic problem has not been extensively studied as the static one. Devine et al. [31] provide ideas to address the dynamic problem. Willebeek-LeMair and Reeves [75] provide a comparison study of dynamic load balancing strategies. Chen and Taylor [12] achieved improvements up to 36% when heterogeneity is considered in distributed systems. Furthermore, speed is commonly the main objective in dynamic load balancing while the quality of the partition (its balance) comes in second place. A less balanced distribution of work does not necessarily mean an increase in computing time; it may allow other metrics to improve such as communication overhead.

#### Refinement of the Dynamic Mesh

The quality of the solution depends on the accuracy of the discretization of the mesh; however, a static mesh with high degree of refinement requires considerable computational power. Dynamic meshes address this problem and are used when the static counterparts do not fulfill the requirements. Over time, dynamic meshes are modified to increase, or decrease, the accuracy in certain regions while keeping the overall computational cost at a minimum. The refinement and coarsening process takes the original mesh and adds or subtracts vertices according to function $f$ producing a new mesh based on error estimates of the previous computational step. As the FEM application is govern by the size and distribution of the mesh, this process heavily affects its performance and efficiency in parallel implementations.

Several techniques have been proposed in literature, they are classified into two broad categories: dynamic and static algorithms. Some of them use the Delaunay refinement [76], [77], but in practice this approach faces important problems such as handling small input angles and the tendency to produce slivers in three dimensions. Others techniques use ball-packing methods, which also have some drawbacks such as generating large meshes. Various other techniques have been proposed in literature, Hudson, in his doctoral work [52], provides extensive information on the topic.

#### Element Migration

Once the dynamic mesh has been refined adaptively, a load imbalance is introduced to the system; a new distribution of the elements is necessary to restore the balance. A cost function categorizes elements in boundary regions. Next, appropriate elements are moved to neighboring subdomains until the balance is restored. The migration tends to be localized, when elements are not moved between nonadjacent subdomains, in order to keep the quality of the partition from being degraded; however, it may be more difficult to achieve a balance. One key aspect to consider is the cost of this migration; it may be more efficient to keep a small imbalance when the cost of the migration is high. Migrating elements could be costly as large amounts of data have to be sent through network links. We refer the reader to [74], [78]–[80] for more information.

### 3.2.2. Approaches to Balance the Load

Load balancing is important in parallel computations and an interesting area of research with a vast range of applications. It was first introduced by Shivaratry et al. [9] who described and compared some common strategies. It maximizes the overall performance of the application, in a parallel system, by reducing the processor idle time and communication. All the processors should have the same amount of work and the data dependencies between them should be reduced in order to minimize the overall computational cost. Hence, efficient load balancing techniques are required to fully exploit the advantages of HPC systems.

To date, simulations may require weeks, months or even years to be performed. Load balancing techniques are an important part of FEM simulations; the load balancer is responsible of the overall efficiency during the execution of the simulation. However, the time required by the load balancer must be kept to a minimum. It is often preferred a fast partition of the mesh than a completely balanced one.

The nature and size of the problem make it unfeasible to search for an exact solution as the search space increases exponentially with the size of the problem [72].

There exist several classifications proposed in literature; we mention only a few of them, but refer the reader to the work of Bichot [81] for more information. The next paragraphs describe different approaches to address the load imbalance in parallel computations.

### Heuristics and Metaheuristics

To address the load balancing problem, most of the efforts made in the past focus on heuristics and approximation algorithms due to the nature of the problem. These techniques provide fast and acceptable good solutions to keep the computational and communication cost under control [72], [74], [82]. Farhat et al. [83] have implemented a number of algorithms and demonstrated their efficiency in practical large-scale problems. They provide detailed descriptions with strong and week points of those algorithms. The list includes the greedy algorithm, the reverse Cuthill-McKee algorithm, the principal inertia algorithm, the recursive graph bisection algorithm, the ID topology frontal algorithm, and the recursive spectral bisection algorithm, just to mention some. Bichot [81], [84] proposed a method called fusion fission and compared its performance with several other algorithms such as simulated annealing and ant colony.

### Spectral Methods

Spectral methods were widely used due to the quality of the solutions they provide. Later, they were eventually replaced by hierarchical methods which are considerable less expensive while provide solutions with comparable quality [81]. Examples of spectral algorithms can be found in literature such as the work by Hendrickson and Leland [67], Barnard and Simon [85], Pothen et al. [69], among others. We refer the reader to the doctoral work of Bichot [81] and Diekmann et al. [74] where more information on spectral methods is provided.

### Diffusive Methods

Another approach to the load balancing problem uses diffusive methods such as the work by Walshaw et al. [86]. Their works is based on the algorithm proposed by Hu and Blake [87], which in turn is an improvement of previous methods studied by Cybenko [88].

### Kernighan-Lin Based Algorithms

The Kernighan-Lin (KL) algorithm [72], named after its authors, is not a partitioning algorithm, it refines an existing partition in a graph. The refinement is achieved by moving vertices between neighbor subdomains in order to improve the quality of the partition and stops when no further improvement is possible. Due to its complexity, $O(n^2 \log n)$, Fiduccia and Mattheyses [71] proposed a linear-time variant with similar results. Other approaches, such as [89], [90], use these concepts to address the load balancing and the graph partitioning problems. Abou-Rjeili and Karypis [91] and Walshaw et al. [89] provide more information on this topic.

### Graph Partitioning Techniques

Recent efforts have contributed to develop new techniques designed to address the mesh partitioning problem on parallel computers; many of them by focusing on the graph partitioning problem. These new techniques have been implemented in numerous frameworks and graph partitioning libraries. However, the graph partitioning problem is not recent [72].

The mesh of information of FEM applications can be characterized by a graph and different types can be used for this purpose according to the requirements. With this in mind, efforts to address the mesh partitioning problem in parallel FEM simulations have been focused on graph partitioning techniques. The graph partitioning algorithms generate an array which contains the location information for every graph vertex; it indicates to which processor the vertex should be migrated. This information is then translated to the mesh. When dual or nodal graphs are used, the output of the partition libraries contains only the new distribution of the mesh elements and a separated distribution needs to be computed for the nodes. Fjällström [11] describes the graph partitioning problem, a number of algorithms and their applications. Buluç et al. [10] provide a survey of the most recent graph partitioning algorithms and their applications.

The mesh of information is first converted into a weighted graph. The weights of vertices represent the calculation costs while the weights of edges the communication costs. A variety of graph types can be used for this purpose. The selection of the type is done according to the requirements of the application, the model to estimate the costs, and the desired precision to approximate the cost model. We refer the reader to the work published by Basermann et al. [92] for more details.

The different types of graphs than can be used are:

- *Dual graph* or *element graph*. In this type of graph, vertices represent mesh elements and their weight the computational costs of those elements. Edges, also weighted, symbolize the communication cost between the corresponding mesh elements. Vertices in the graph are connected by edges only when the respective mesh objects share and edge or face, in two dimensions or three dimensions respectively.

- *Extended dual graph*. Some types of meshes cannot be accurately characterized by a dual or element graph. This is the case when the mesh elements have different dimensions; it is not possible to represent the potential communication. Extended dual graphs solve this problem by connecting vertices only if the mesh elements share one or more nodes. This approach maintains all connections even between different dimension elements that otherwise would be lost in a dual or element graph. However, extended dual graphs are in general more complex and its associated cost superior, especially for 3 dimensional meshes.

- *Generalized dual graph*. This kind of graph is a compromise between the extended dual graph and the element graph making it also suitable for meshes with elements of different types. A key characteristic is that vertices are not always connected when the mesh elements share a node; they are connected by and edge depending on the local maximum number of shared nodes between those mesh elements.

- *Nodal graph*. This type of graph uses a different approach. Vertices represent the nodes in the mesh and they are linked only when they share a mesh element.

- *Combined graph*. Vertices represent both the elements and the nodes making it easier to describe the different calculation costs. It is due to the fact that FEM applications use nodes to describe inter-process communication and the graph edges for the potential communication between elements and nodes. All kinds of connections such as element to element, element to node and node to node would be included.

### 3.2.3. Load Balancing through Graph Partitioning

FEM applications use a mesh of information to describe the object under analysis. Often, graphs are used to represent this mesh in order to address the load partitioning problem when the data (work) needs to be reallocated amongst the processors. The vertices of the graph describe the mesh elements (data or work) to be partitioned whereas the edges represent the potential communication between those mesh elements. The amount of communication required during the computation of the solution is estimated by the boundary edges (edges connecting vertices located in different subdomains). The weights of vertices and edges reflect, to a degree, the associated costs of computation and communication respectively. Thus, the aim is to create a balanced partition and keeping the edge cut to a minimum. It is vital to use the correct type of graph to represent the mesh of information; the accuracy of that representation heavily affects the final result of the simulation. Basermann et al. [92] provide detailed information on this regard. The graph partitioning libraries provide an array that indicates for each vertex the location (processor) it should be migrated. Though, this is usually not a perfect balance of the load since the vertex weights represent only an approximation of the work. In fact, the local subdomain solutions define the computational cost.

Regardless the type of graph used for load balancing, they have limitations. There is always a compromise between its benefits and drawbacks when choosing the correct graph type for the problem. Graphs can only represent an approximation of the computation and communication costs and are limited to a type of system [93]. The graphs used to model the mesh of information are generally undirected. This implies symmetry in all relations between vertices, making them unsuitable for non-symmetric matrices. To address these problems, hypergraphs have been also used in FEM applications. As in a standard graph, hyper vertices also represent the data. However, the hyper edges represent sets of related vertices making the hyper edges in the cut a precise characterization of communication costs, not just an approximation [94]. Hypergraph partitioning has proven to produce high quality solutions in many areas of application such as sparse matrix decompositions [94], [95], database storage and data mining [96], [97], and Very-Large-Scale Integration (VLSI) design [98]. However, it has been demonstrated that hypergraph partitioning is considerably slower than graph partitioning [99]. It is confirmed by the generalized use of graph partitioning algorithms and libraries to balance the workload in parallel FEM computations.

Graph partitioning problems are important in various areas of engineering and computer science. Examples of applications are present in VLSI design, FEM, image segmentation, route planning, social networks, air traffic control, among others [81], [84], [100], [101]. Many of these applications use a graph, to represent the data, and employ a variety of techniques to dive it; with the final goal being the solution to the problem they treat. It is an essential part of FEM applications for the load balancing step; it balances the load while keeping the communication at a minimum in scientific simulations.

### The Graph Partitioning Problem

Briefly, the graph partitioning problem involves the creation of subdomains, or smaller groups, from a collection of vertices in a graph, according to some objectives such as the minimization of a cost function. The problem becomes more complex when the number of objectives increases or when they oppose to each other. We now formally define the graph partitioning problem and describe the most important metrics used to measure the quality of the partition. For the purposes of this work, and based on [79], [91], [102], [103], we define the *k*-way graph partitioning problem as follows:

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $|E|$ edges. Both vertices and edges are weighted with positive integer values. The vertices represent the computational load by the corresponding mesh nodes, whereas the edges represent the data dependencies between them. The weight of vertex $v_i \in V$ is denoted by $\|v_i\|$; similarly $\|e_i\|$ denotes the weight of edge $e_i \in E$.

Given that the graph has to be divided into $k$ different subdomains, a partition $\pi$ of $G$ is defined as the mapping of $V$ into $k$ disjoint subsets $S_1, S_2, \cdots, S_k$ such that $S_{ii} \cap S_{jj} = \emptyset$ for $ii \neq jj$, $\bigcup_{1 \leq ii \leq k} S_{ii} = V$ and $\|S_{ii}\| = \bar{S}$; where the weight of a subdomain is the sum of weights of its vertices, $\|S_{ii}\| = \sum_{v \in S_{ii}} \|v\|$, and the optimal subdomain weight is given by $\bar{S} = \lceil \sum_{v \in V} \|v\| / k \rceil$. The graph partitioning problem is to find a partition that balances the load whilst minimizes the communication costs. Note that a perfect balance is not always possible.

The set of edges denoted by $C$ is called the edge cut; it is formally defined as the set $C = \{(v_i, v_j) | v_i, v_j \in V \wedge (v_i, v_j) \in E \wedge v_i \in S_{ii} \wedge v_j \in S_{jj} \wedge ii \neq jj\}$ with the edges having their vertices in different subdomains. If there are vertices $v_i, v_j \in V$, and there is an edge such that $(v_i, v_j) \in E$ with $v_i \in S_{ii}$, $v_j \in S_{jj}$ for $ii \neq jj$, then it is said that the edge $(v_i, v_j) \in C$. The size of the edge cut is given by $\|C\| = \sum_{e \in C} \|e\|$; the sum of weights of the edges in the cut. The vertices which have an edge in $C$ are referred to as *boundary* vertices. The *imbalance* is defined as $\max_{1 \leq ii \leq k} \|S_{ii}\| / \bar{S}$, which is the maximum subdomain weight divided by the optimal weight. A more precise definition of the graph partitioning problem is therefore to find $\pi$ such that $\|S_{ii}\| \leq \bar{S}$ and $\|C\|$ is minimized. Note that the number of edges in the cut is minimized when the edges have unitary weights (i.e., $|C| = \|C\|$). A partition is represented by a vector $\pi$ of size $n$, where $\pi[v] = ii$ for all $v \in S_{ii}$.

### Metrics to Evaluate the Partition

Frequently, the edge cut is used by graph partitioning algorithms to evaluate the quality of the partition as it is much easier to optimize compared with others. However, this metric reflects only an approximation of the potential communication costs during the simulation and it is affected by the type of graph selected to model the mesh of information. It is well known, and has been demonstrated, that the edge cut is not the best metric to optimize when used to balance the workload of parallel FEM applications [104], [105]. Including information related to the solver during the partitioning process can reduce the potential communication volume [82]. Minimizing the boundary vertices can lead to better results, but unfortunately this metric is much harder to optimize [93].

Other metrics have been used to try to improve the quality of the partitions. Primarily, the characteristics and requirements of the FEM application dictate the best metric to optimize. Among the most used we can mention the amount of vertices to be migrated to a different subdomain, the volume of

information to be sent by each subdomain, the volume of information to be received by each subdomain, the diameter of the resulting subdomains.

#### The Problem is NP

Graph partitioning problems are typically NP-hard [64], [82], [101]; the complexity grows exponentially with the problem size. However, other graph partitioning problems are NP-complete, as it has been demonstrated by Gary et al. [6]–[8]. Consequently, solutions to these problems focus on heuristics and approximation algorithms to keep the computational cost under control [82]. As an example, the work by T. N. Bui and C. Jones [64] demonstrates that finding a good vertex and edge partition is NP-hard.

## 3.3. Graph Partitioning Algorithms

During the years, an increasing number of techniques and methods to deal with the load balancing problem of FEM applications have been proposed. These methods have been implemented in a number of graph partitioning libraries. Fjällström [11] describes the graph partitioning problem, a number of algorithms and their applications. Buluç et al. [10] provide a survey of the most recent graph partitioning algorithms and their applications. Schulz [106] presents a list of graph partitioning techniques in his PhD dissertation. Next, we summarize and describe the most common techniques.

### 3.3.1. Genetic Methods

Genetic algorithms are metaheuristics used to generate high quality solutions to optimization problems. They are inspired in the process of natural selection. While they may provide good solutions, often they require higher processing power to generate same quality solutions compared to other methods. In addition, genetic algorithms do not scale well with complexity. A survey of genetic approaches, presented by Kim et al. [107], has a much deeper analysis.

### 3.3.2. Diffusive Methods

As its name suggests, this technique mimics the physical process of diffusion. It is easy to visualize the similarity between them; the work is spreading among all processors such as the heat in a block of metal. These methods are simple, well studied with several examples in the literature, and designed for dynamic balancing. Much work has been done in this area [66], [88], [108]–[114].

### 3.3.3. Spectral Methods

More elaborate methods, called spectral methods, utilize eigenvalues of the Laplacian matrix of the graph. These methods are, in general, much more expensive [69], [115], but with the proper optimization they became state-of-the-art for the graph partitioning problem [116], [117]. The Multilevel Spectral Bisection (MSB) is an optimization of the original Spectral Bisection (SB). MSB produces partitions of the same quality as SB in a fraction of the time, one or two orders of magnitude [85]. Other approaches using spectral methods can be found in [78], [108], [110], [113].

### 3.3.4. Greedy Methods

Greedy approaches are considered among the fastest and easiest to implement. They use the graph connectivity to create the subdomains. In order to create a subdomain, an initial seed is taken then further adjacent vertices are added until the appropriate size for the subdomain is attained. New subdomains are generated following the same principle until all vertices are assigned to a subdomain. The way of choosing an adjacent vertex, in order to assign it to a subdomain, influence the quality of the partition. Several possibilities exist such as vertices that reduce edge cut [65], [70], following the breath-first method [66], etc. The initial subdomains are, in general, very compact, but the final subdomain is composed by the left-over vertices, reducing the quality of the partition. Different methods try to solve this problem [70], [89], [118], [119].

Bubble [108] is a well-known algorithm fast and easy to implement. Unfortunately there is no guarantee on the quality of the partition; its major drawback. Even though, after several iterations the seeds could be uniformly distributed, the resulting subdomains may not be of the same size. To address this problem, a number of optimizations have been suggested over the years.

### 3.3.5. Geometric Methods

These methods are known for their simplicity and speed, but can only be used when vertices have coordinate information. Furthermore, the quality of the partitions is worse compared with other more expensive methods such as spectral [120], [121]. They use the geometric locality to create subdomains; very important when this is the main goal. However, they may induce a high volume of communication between subdomains due to the lack of explicit control over this metric. Examples of this approach are presented in [31], [70], [83], [122], [123].

### 3.3.6. Multilevel Methods

Recently, researches have focused on an efficient method for graph partitioning. It provides higher quality partitions, even compared with spectral methods, with a reasonable computational complexity [68], [115], [124]. The idea behind this method is simple and consists of four phases called Coarsening, Matching, Initial Partition, and Uncoarsening. For more details, the reader should refer to [68].

First, the graph is reduced until a point where it is easy to handle. The reduction produces a series of progressively smaller graphs by matching vertices and collapsing them together. Then, the smallest, and coarsest, graph is divided creating an initial partition. It is then projected back to the original, and finer, graph. The projection constantly refines the partition as finer graphs offer more opportunities to improve its quality. The main matric used to define the quality of the partition is the edge cut; an estimate of the total communication induced by the partition.

According to the results in [68], [115], multilevel methods generate partitions of good quality with a variety of unstructured graphs. However, only experimental analysis has been presented, the theoretical analysis to explain the effectiveness of these methods is missing.

Next, we describe in detail the different phases of multilevel methods by using the definitions in [79], [90], [124], [125]:

- *Coarsening phase.* A series of progressively smaller and coarser graphs, $G_i = (V_i, E_i)$, is created from the original graph $G_0 = (V_0, E_0)$ such that $|V_i| > |V_{i+1}|$. The coarser graph $G_{i+1}$ is constructed from graph $G_i$ by finding a maximal matching $M_i \subseteq E_i$ of $G_i$ and collapsing together the vertices that are incident to each edge of the matching. Vertices that are not incident to any edge of the matching are simply copied to $G_{i+1}$. When vertices $v, u \in V_i$ are collapsed to form vertex $w \in V_{i+1}$, the weight of vertex $w$ is $|w| = |v| + |u|$, while the edges incident to $w$ is set equal to the union of the edges incident to $v$ and $u$ minus the edge $(v, u)$. In the case where vertex $z$ in $G_i$ contains edges to both $v$ and $u$, such that $(z, v)$ and $(z, u)$, then the weight of the resulting edge in $G_{i+1}$ is set to $|(z, v)| + |(z, u)|$. Thus, during this process, the weights of vertices and edges, in the successive smaller graphs, increase with every level.

- *Matching Phase.* Some authors do not consider the matching as a separate phase, but part of the coarsening one. There are different ways to generate the matchings during the coarsening phase [68], [79], [91]. They have an important effect on the quality of the final partition and the total execution time of the partitioning process [124], [125]. Here, we just mention some of them. Random matching (RM) uses a randomized algorithm to generate a very fast matching [102], [115], [124], [125]. Heavy Edge Matching (HEM) [68], [90], [102], [124], [125], computes a matching $M_i$, such that the weight of the edges in $M_i$ is high. The Modified Heavy Edge Matching (HEM*) [90] is an optimization of HEM to minimize the average degree of the graph on every subsequent coarsening level. Walshaw and Cross [79] implemented a variation of the method proposed in [115] by Hendrickson and Leland.

- *Initial Partitioning Phase.* A partition of the coarsest graph $G_k = (V_k, E_k)$ is computed. Diverse methods can be applied for this purpose due to the size of $G_k$. Expensive methods can even be used without incurring in significant increase of the execution time. An study and comparison of a number of algorithms is presented in [68].

- *Uncoarsening phase.* The initial partition of the coarsest graph $G_k$ is projected back towards the original graph $G_0$ by going through the graphs $G_{k-1}, G_{k-2}, \cdots, G_1$, refining the partition at each graph level. Even if the partition of $G_i$ is at a local minima, the partition of $G_{i-1}$, obtained by the projection, may not be at a local minima. Hence, local refinement heuristics must be applied to improve the partition of $G_{i-1}$. Several algorithms for this purpose are presented and analyzed in [68].

Within multilevel methods, the graph partitioning problem is generally addressed by the use of recursive bisection. The original graph is first bisected, and then each half is recursively bisected again. The process is followed until the desired amount of subdomains is attained. George Karypis uses this approach in METIS [37] to compute all the partitions. It is also possible to directly compute a *k*-way partition, but the coarsening phase may become more expensive to perform. Nevertheless, there are some advantages such as the fact that the coarsening phase is performed only once, and that recursive bisection can lead to much worse partitions in some cases [126].

## 3.4. Most Common Approach

The literature reveals that the most common approach to address the load balancing problem in parallel FEM applications is through graph partitioning techniques. They do not, however, provide solutions with the highest quality compared with other methods. Graphs have numerous limitations when used to represent the mesh of information of FEM applications. None the less, the trade between speed and accuracy makes graph partitioning the best option to address the load balancing problem. It is more important to have an acceptable solution in the fastest way possible.

The next chapter provides information on a selection of software and libraries which address the load balancing problem through graph partitioning. It also sets the grounds for the development of DSHEM.

# Chapter 4.

# Current Solutions

An increasing number of frameworks and libraries have been developed over the years to address the load balancing problem in FEM simulations. They provide different functionalities and performance for a variety of requirements. Some of them are free while others are only commercially available. Whether it is a FEM framework or a fully integrated simulator, they all require a load balancing module to improve their performance in parallel systems. However, the vast majority delegate this task to external libraries.

In this chapter we give an overview of the available software. We refer the reader to the website of swMATH [127] for more information on mathematical software, or the corresponding references of the software presented here.

## 4.1. FEM Frameworks

A large variety of FEM frameworks and tools have been introduced in the last years [26], [27], [29], [30], [53], [128]–[145]. While some provide effective results for particular problems, others are more suitable for general purposes. We mention the most relevant tools including their main features.

Charm++ [128] is a framework created at the University of Illinois. It gives scientists the opportunity to focus on modeling the problem and not on parallelization details. It is based on a data driven execution and multi-partition decomposition to increase the efficiency of the simulation. The application is divided into a set of small parts called objects. The objects are then distributed amongst all available processors and the communication is defined between them and not between processors. The framework separates the numerical algorithms from the parallel implementation. One example of FEM simulators based on the Charm++ framework is NAMD2 described in [146]. The authors give the analysis of its performance on some benchmark applications.

Recent effort is being focused on massively parallel programming due to the increasing use of clusters of thousands of processors. Heister et al. [26] focus on the design of efficient data structures and algorithms to fit these new requirements. They have enhanced the library deal.II [145] to profit from the large amount of processing power of clusters. The library utilizes new method in order to reduce, into smaller blocks, the finite element implementations by using data encapsulation and object oriented techniques. deal.II is a mature library with a large support of different applications including various scientific areas, application-specific algorithms, and programming methodologies.

Dolfin [131] employs new techniques in order to generate code automatically. First, mathematical notations express the Finite Element (FE) variational forms, and then low-level code is generated and

compiled. Afterwards, linear algebra and computational meshes are integrated with the code. Dolfin differs from many other projects such as Sundance [134] and Life [137], [138], among others, for the simple reason that it focuses on code generation. As a result, Dolfin supports an extensive range of FE since it may assemble FE variational forms on FE spaces supported by the FE backend and the *form compiler*. The latter automatically generates code according to the high level characterization of the FE variational form provided by the users [131].

FEAST [132] is a FE based solver for the computations of PDE problems and it is designed to fully exploit the advantages of the large processing power of parallel HPC systems. It was created at the Technische Universität Dortmund and it is the successor of the established FE packages FEAT and FEATFLOW [147]. The next version, FEAST2, is currently under development and will include new features such as 3D support.

SIERRA [129] is a framework developed at the Sandia National Laboratories aimed to help write parallel applications. SIERRA is a comprehensive heavyweight framework that is continuously evolving. It is designed for massive parallel simulations of multiphysics applications. It includes a set of parallel data structures and software services that facilitate code development. Developers of mechanics code focus on the mechanics algorithms instead of parallelism, data structures, load balancing, etc. The framework allows developers to use different programing languages such as FORTRAN and C.

Diffpack [133], [140], [148] is a framework for the numerical solution of PDEs. From the point of view of a programmer, Diffpack is a set of libraries with a collection of classes written in C++. First appeared in the early 1990s and still actively developed. It is currently a commercial software maintained by inuTech GmbH [149]. The main libraries of Diffpack are BasicTools, LaTools, DpKernel and DpUtil.

FreeFEM++ [136], [144] is another framework oriented to solve PDEs using the FEM. It is currently being developed and maintained by the Université Pierre et Marie Curie and the Laboratoire Jacques-Louis Lions. It is a multi-platform, free software, written in C++ that provides a high level programming language.

All frameworks focus on facilitating the creation of parallel FEM applications without the hassle of parallelization details; the application programmer can now focus all the attention on modeling problem. The parallelization, and inherent load balancing, of every framework is already integrated and hidden from the user. Frequently, the load balancing part is handled by external libraries such as METIS, Jostle and Chaco which are described in Section 4.4. Charm++, FreeFEM++ and deal.II rely on METIS for balancing the load in parallel systems. SIERRA uses the Charm++ load balancing module, which in turn relies on METIS. Diffpack does not include a specific load balancing module; it gives the option to use external partitioning libraries such as METIS and Jostle. It is not clear how Dolfin and FEAST handle parallel systems.

## 4.2. Simulators

While FEM frameworks focus on the low level layers of the development of applications, ready-to-use software is also available [41], [150]–[155]. Due to the scope of this work we just mention a few of them with a brief description.

Analysis3D [153] is a FEA software designed for structural engineering of steel construction of frame

and truss design. It comes with a Microsoft Windows user interface to facilitate the creation of models, as well as a library with the most commonly used sections such as AISC, British steel, DIN and Euronorm. A free educational version is available.

PadFEM [41] is an easy to use interface for parallel FEM simulations on distributed memory systems. It combines different methods to support scalable massively parallel numerical simulations with a modular structure.

ESI Group [151] develops a wide selection of software for different applications such as casting, electromagnetic, crash, biomechanics, and fluid dynamics, among others.

ANSYS, Inc. [150] provides a set of commercial solutions for a wide variety of applications. ANSYS multiphysics applications include thermal analysis, electronics cooling, fluid structure interaction, heat transfer, among others. ANSYS CFD is designed to simulate the behavior of fluids flows. ANSYS structural mechanics focuses on impact analysis, strength analysis, vibration, curability, etc. ANSYS electronic solutions focus on the analysis of electric motors, low-frequency electromagnetics, radio frequency and microwave, signal integrity, among others.

Commercial software is difficult to evaluate due to the fact that the source code is not available. ESI Group and ANSYS, Inc. do not share any information on how parallel applications are handled to improve the efficiency of the system. Analysis3D provides only a sequential implementation, rending it useless for HPC systems. PadFEM uses direct partitioning heuristics.

## 4.3. Load Balancing Libraries

A number of load balancing libraries have been proposed over the years; each of them with a particular problem in mind to address. This section presents the most representative of them, namely DRAMA [92], [156], Zoltan [157], [158], DRUM [159], [160], and UMPAL [161]. We refer the reader to the respective literature for more details on each of them.

The Dynamic Re-Allocation of Meshes for parallel finite element Applications (DRAMA) [92], [156] is a library designed to address the load balancing problem in FEM applications. It is the result of a European Commission project involving several countries. It is intended to be a general purpose library, but much emphasis was made to fit the requirements of two commercial FEM frameworks: PAM-CRASH/PAM-STAMP by ESI Group [151] and FORGE3 by Transvalor [152]. DRAMA is a library designed to be used in parallel MPI applications. It can directly reallocate the mesh elements or call ParMETIS and Jostle, two graph partitioning libraries, to accomplish the task. It uses the DRAMA cost model [156] to reallocate the mesh elements during the load balancing steps of FEM computations. The model is well suited for the inherent dynamic changes of computations and communications. It fulfills additional requirements due to its dynamic nature: parallel execution, the current partition is taken into account, interacts with the FEM application. A study by Basermann et al. [92] shows that FORGE3 and PAM-CRASH/PAM-STAMP benefit from the dynamic load balancing provided by DRAMA.

The Zoltan Parallel Data Services Toolkit [157], [158] is another library created by Center for Computing Research (CCR) at the Sandia National Laboratories. It is a collection of utilities to address load balancing, data migration, matrix ordering, and graph coloring among others. Zoltan has a unique design which allows it to support a wide range of applications; it uses a neutral data structure approach. FEM applications are not forced to implement a certain type of data structure in order to use Zoltan

during the load balancing process. Zoltan offers a callback interface to interact with the applications which provide the required data. DRAMA, in contrast, can only be called by mesh based applications. However, DRAMA performs better when data migration is critical due to the knowledge of the data structures used by the FEM application. Another advantage of Zoltan is the ability to add new algorithms to the toolkit; its source code is available on the website. It is easy to test new heuristics and compare them with the existing ones.

Dynamic Resource Utilization Model (DRUM) [159], [160] is the result of the PhD dissertation by Jamal Faik. It makes use of the execution environment in order to improve the load balancing process. The library is not publicly available as it is not yet ready for general use. DRUM is a parallel library developed in C and MPI. It requires information of the execution environment; part inferred at run time and the rest provided manually. In combination with Zoltan, it achieves good results during the load balancing process. There is a graphical user interface called DrumHead that helps during the initial configuration of DRUM. DRUM is not a load balancer per se; it encapsulates the hardware and network topologies and uses that information in conjunction with existing load balancers to improve their results. A model is first created to describe the execution environment and includes network and nodes capabilities; LINPACK [162], [163] is used to assess the hardware capabilities.

UMPAL [161] is a library that includes five different tools: a web interface to use UMPAL, a visualization tool to view the mesh once partitioned, a simulator, a load balancer with different algorithms, and a partitioner that uses Jostle, METIS and PARTY. It is an effort from the TungHai University in Taiwan. The partitioner within UMPAL uses three libraries: Jostle, METIS and PARTY. The characteristics of the graphs influence the results of those libraries. In order to obtain better results, all three libraries are executed independently and the best partition is chosen. That partition is then optimized with the Directed Diffusion Method (DD) [87], the Dynamic Diffusion Method (DDM) [111], or the Multilevel Diffusion Method (MD) [164]. The load balancer includes two different algorithms, the Prefix Code Matching Parallel Load Balancing (PCMPLB) and the Binomial Tree Based Parallel Load Balancing (BINOTPLB), both by Liao [111].

With a wide variety of load balancing libraries, it is curious that few of the FEM frameworks or simulators make use of them; perhaps an interesting detail contributes. The load balancing libraries rely on external graph partition libraries to distribute the load in a parallel system. DRAMA offers several mesh partitioners based on mesh migration, coordinate partitioning and graph partitioning. The first two are custom design whilst the last uses parallel versions of METIS and Jostle. Zoltan uses a similar approach delegating the graph partitioning to METIS and Jostle. DRUM interacts with Zoltan for the load balancing part adding support for heterogeneity. UMPAL relies on METIS, Jostle and PARTY for the partitioner.

## 4.4. Graph Partitioning Software

To date, there is a wide variety of libraries designed to improve the efficiency of parallel systems during FEM simulations. A number of them are free of charge for academic use, such as METIS [36], [37], Chaco [116], [117], and SCOTCH [165]–[167]. We refer the reader to [168] which provides more details on each of them. Over the years, a number of studies compares the software [74], [79], [90], [169]. However, it is difficult to reach a clear conclusion due to the different execution parameters and

input graphs available.

Jostle [164], [170], [171] is a multilevel graph partitioning library for undirected unstructured graphs in parallel computers. It was first introduced in 1995 and recently became NetWorks [172], a commercial software. Jostle provides superior results in successive repartitions (balance existing partitions) compared to METIS [36], [37]. It employs a modified version of the KL algorithm [72], with a balance flow, for the refinement process in order to optimize the partitions. The coarsening phase reduces the original graph until $k$ vertices remain, where $k$ equals the number of desired subdomains. This is possible because the refinement is performed during both the coarsening and uncoarsening phases. Jostle is suitable for a dynamic load balance of the constantly changing mesh during the FEM simulations. It also includes a number of execution parameters to fit a variety of requirements.

Chaco [116], [117] was first released in 1993 and it is being maintained by the CCR at the Sandia National Laboratories. It includes a variety of methods to produce high quality partitions. Chaco provides five types of partitioning heuristics: linear, inertial, spectral, KL, and multilevel-KL. Some of them were created by the authors while the others are optimization of existing methods; we refer the reader to [117] for more details. The Chaco development focuses on three main tasks: graph partitioning with a variety of methods, matching the partitions to the hardware topology, and sequencing graphs to preserve locality by the use of spectral methods. It includes several features such as the ability to read an existing partition from a file to optimize it. It may be difficult to use for a beginner as it contains a large number of parameters designed to optimize the partitions. However, many of those parameters are already set to reasonable defaults values.

PARTY [173]–[175] is an effort, from the Institut für Informatik of the Universität Paderborn, to provide a free, and easy to use, library with a number of different methods for partitioning graphs. It was first available in 1996 with version 1.1. PARTY is a versatile library able to be embedded into an existing application or used as a stand-alone tool. It can also interact with the Chaco library by providing interfaces that allow the user to call Chaco's methods within PARTY. In addition, a default execution configuration is provided to enable new users a quick start. The PARTY library implements several methods to generate the initial partition, among them we can mention random, linear, and Farhat techniques. The refinement is done with KL or Helpful Sets (HS) [65], [176], a method by their own. The local refinement in PARTY differs from other implementations; it uses an approach based on a theoretical analysis to improve the partitions [177], [178]. The HS heuristic moves sets of vertices instead of single vertices during the refinement process. It is only possible to apply HS to bisections; however, the common way to address the $k$-way partitioning problem is by the use of recursive bisection. More information is available on the PARTY website [175].

SCOTCH [165]–[167] is developed at the Laboratoire Bordelais de Recherche en Informatique (LaBRI) of the Université Bordeaux I in France. Its development started in 1992 and currently version 6.0 is available. SCOTCH is a library for graph and mesh partitioning, reordering of sparse matrices, and static mapping. It uses the mapping algorithm called Dual Recursive Bipartitioning (DRB) and also includes other heuristics [179]. Recently, developers have introduced hypergraph partitioning algorithms extending some of SCOTCH capabilities such as support for native mesh structures. They also introduced parallel heuristics for graph ordering which are available in Parallel Threaded SCOTCH (PT-SCOTCH).

METIS [36], [37], [68], [99] is a library for partitioning graphs developed by George Karypis at the

Department of Computer Science & Engineering of the University of Minnesota. Current available version is 5.1.0 and its development goes back more than 20 years. It is a mature and wide used library for load balancing FEM applications. METIS is extremely fast creating partitions of graphs with millions of vertices in only seconds. It takes advantage of the multilevel paradigm [68], [90], [115], [180] to reduce the graph and create fast and high quality partitions. It consistently produces superior partitions, up to 50% better, compared to spectral methods. Furthermore, there is a parallel version of the library called ParMETIS based on the Message Passing Interface (MPI). METIS is a collection of various serial programs for partitioning graphs and meshes as well as reordering sparse matrices. Several algorithms have been implemented like recursive bisection [68], multilevel *k*-way [103], and multi-constraint methods. METIS, as well as Jostle, uses the algorithm by Hu et al. [181] to determine the best balancing flow for shifting elements. METIS is able to partition meshes by converting them into graphs. It currently supports four mesh element types: triangles, tetrahedra, hexahedra, and quadrilaterals.

As mentioned before, a concrete and concise conclusion comparing the different graph partitioning libraries cannot be established. Many comparisons between them have been published [74], [79], [90], [169] with different results. Karypis and Kumar [90] pointed out some differences on the refinement phase between Chaco and METIS, leading to a more expensive partitioning process in the former. Diekmann et al. [74] showed that Jostle and METIS are not suitable for use in long periods of time without a complete repartitioning from time to time, when AR is a metric of importance. The experimental results by Walshaw and Cross [79] show a degraded performance of METIS compared to Jostle due the coarsening phase. METIS coarsens to 2000 vertices while Jostle coarsens until the number of vertices is the number of required subdomains. Usually, Jostle generates better partitions than METIS but it takes longer to compute them. Furthermore, some solvers have constraints, such as straight partition boundaries and connectivity, which cannot be addressed by Jostle and METIS.

## 4.5. Hypergraph Partitioning Software

Graph partitioning libraries are widely used to address the load balancing problem in parallel FEM applications. It has been shown that hypergraphs can represent, more accurately, FEM meshes than regular graphs providing better results; however, they are more expensive in terms of computational time. Nevertheless, hypergraph partitioning libraries have been available to fulfill stricter demands when execution time is not an issue. Among the most popular libraries we can mention hMETIS [182], [183], PaToH [184], [185], and Mondriaan [186]; which are all serial implementations. Zoltan [157], [158], a load balancing library, also includes a hypergraph partitioner.

hMETIS [99], [182], [183] is a library for partitioning hypergraphs also developed by George Karypis at the Department of Computer Science & Engineering of the University of Minnesota. The latest stable version is 1.5.3, version 2.0pre1 still experimental, and its development goes back more than 20 years. First introduced in 1997 by Karypis et al. [187], hMETIS is a collection of various serial programs for partitioning hypergraphs. It is a library suited for partitioning hypergraphs modeling VLSI circuits, data mining, among other applications. Similarly to METIS, it uses the multilevel paradigm to generate the bisections. It is fast and robust while creating bisections of hypergraphs; and with multiple runs on the same hypergraph, it outperforms the previously known algorithms and benchmarks. According to the author, hMETIS produces bisections of much higher quality compared to other widely used algorithms;

up to 30% on average.

The Partitioning Tools for Hypergraphs (PaToH) [184], [185] is a library developed by Çatalyürek at Bilkent University to address the hypergraph partitioning problem; the development started in 1994 during his doctoral studies. The current stable version is 3.2. It is distributed for multiple platforms such as Linux, macOS, Solaris, and AIX, but it can also be used via Zoltan [158] and Mondriaan [186]; it is a very small library compared to hMETIS. PaToH is fast and stable with some important features: multi-constraint and fixed cells partitioning. It uses the multilevel paradigm to create the partitions, as well as recursive bisection to generate a *k*-way partition. It provides an interface with functions for initialization purposes, memory management and the actual partitioning.

Mondriaan [95], [186] is being developed by the Department of Mathematics of Utrecht University in the Netherlands. The current version is 4.2, released in September of 2017; its development goes back to 2002. Mondriaan employs the multilevel paradigm with recursive bisection to generate a *k*-way partition of a hypergraph. It generates horizontal and vertical divisions similar to the paintings by Mondriaan, according to an author's analogy. It is a sequential library written in C with the main focus on matrix partitioning. However, it is possible to use it with hypergraphs.

UMPa is being developed at the Ohio State University by Çatalyürek et al. [188]; however, the library is not publicly available. Similar to the libraries previously mentioned, UMPa is based on the multilevel paradigm and recursive bisection. It provides improved partitions with a directed hypergraph model and a novel *k*-way refinement heuristic that handles multiple communication metrics. It minimizes the communication that causes excessive network use and bottlenecks.

## 4.6. Limitations of Current Approaches

It becomes clear that the load balancing problem in parallel FEM applications is addressed by the use of graph partitioning methods. Graph partitioning libraries are widely used by FEM applications to improve the efficiency of parallel systems. Hence, all limitations of graphs and graph partitioning techniques are inhered by the FEM applications; those limitations are described earlier in Section 3.2.3, Load Balancing through Graph Partitioning. It is then important to study and improve current graph partitioning techniques if we expect efficient parallel FEM applications. With that goal in mind, we propose a novel idea to address the load balancing limitations of current parallel FEM applications. Chapter 5 introduces the general concept and how new information is incorporated to the graph in order to improve the quality of partitions when the communication volume is considered. Chapter 6 presents the new vertex matching algorithm called DSHEM as a result of the previous analysis; a novel idea that entirely changes the paradigm.

Based on the information presented in this chapter, we focus our effort on METIS. It is used by a majority of the available software making it the perfect candidate. It is open source and free for educational purposes too. Improving the partitions generated by METIS brings benefits not only to the users of this library but to all parallel FEM applications that rely on it.

# Part III.

# DSHEM and Evaluation

# Chapter 5.

# Analysis and Concept

Undirected graphs are widely used to represent FE meshes. Efficient data structures have been proposed to store the graphs as well as many partitioning libraries. We propose a new matching model for the multilevel graph partitioning technique which aims to reduce the communication volume by simulating a directed graph. The main work of this research focuses on METIS, primarily because it is an open source library that allows a clear and fair comparison of results.

This chapter examines the problem of current graph partitioning techniques. Not considering the communication dependencies of the subdomains leads to poor partitions that degrades the efficiency of parallel FEM applications.

## 5.1. Analysis

One of the fist metrics to optimize in graph partitioning techniques was the edge cut; it is fast and easy to implement. Unfortunately, undirected graphs used to characterize FE meshes do not represent the communication costs properly leading to inefficient partitions. This can be depicted easily with an example; Figure 5.1 shows two different partitions for the same graph, both of them with three edges in the cut. It could be assumed that both partitions have the same communication cost due the number of edges in the cut, however this is far from being true. Assuming that the weight for each edge is 1, the edge cut is 3 in both cases. But the communication volume differs due to the fact that each boundary vertex, represented by dotted lines, has to send and receive data.

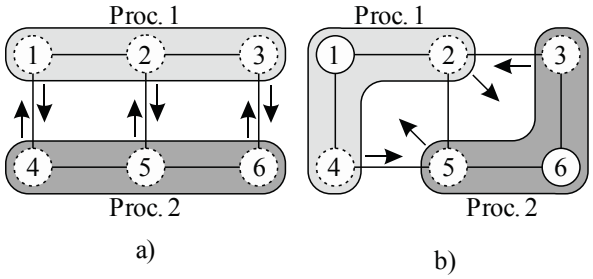


**Figure 5.1. Reduction of communication volume with an edge cut of the same size; a) has a communication volume of 6 whereas b) a communication volume of 4.**

It can be seen in Figure 5.1 that vertices 1, 2 and 3 in a) send data from $Processor\ 1$ to $Processor\ 2$; the same applies for vertices 4, 5 and 6. A different scenario is shown in b) where the communication volume is reduced to only 4 with the same edge cut value. Vertex 2 sends data from $Processor\ 1$ to $Processor\ 2$ only once, regardless of its two edges in the cut. When vertex 2 sends data

to vertex 3, there is no need to send the same information twice; the data is already in *Processor* 2 and vertex 5 can just fetch it locally. The same logic is valid when vertex 5 sends its data to *Processor* 1 only once. This particular case reduces the communication volume to only 4 units.

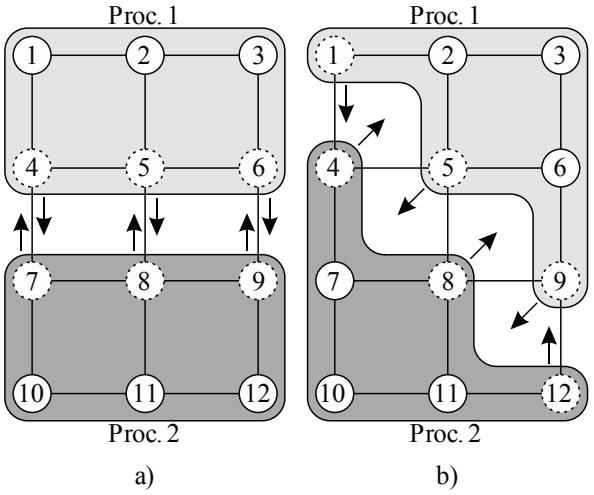


**Figure 5.2. Communication volume with the same number of boundary vertices. a) has an edge cut of 3 while b) an edge cut of 5; however, both a) and b) have a communication volume of 6.**

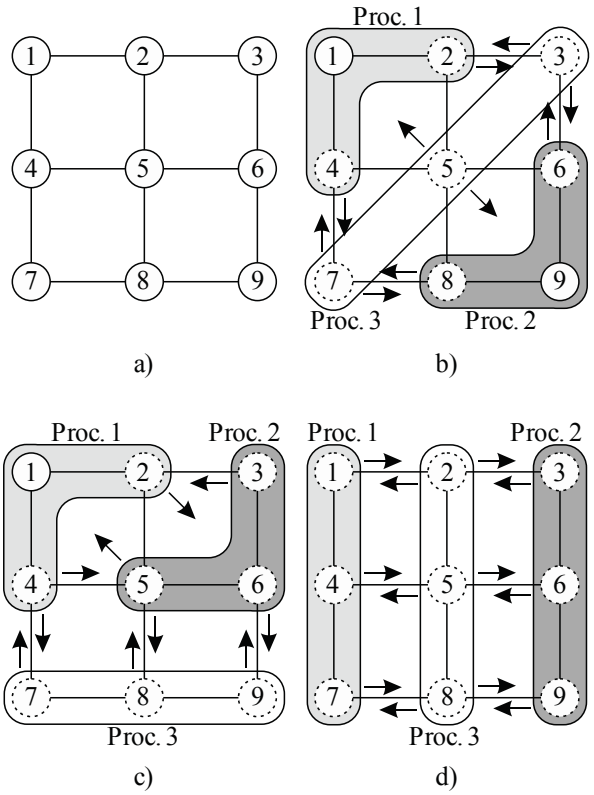


**Figure 5.3. Three different graph partitions and their characteristics. a) shows a small regular graph. A balanced partition with 7 boundary vertices, an edge cut of 8 and a communication volume of 10 is depicted in b). A second partition in c) with 8 boundary vertices, and edge cut of 6 and a communication volume of 10. The third partition consisting of 9 boundary vertices, and edge cut of 6 and a communication volume of 12 in d).**

The previous example leads us to think that the reduction of boundary vertices also reduces the communication volume and Figure 5.2 supports the hypothesis. Having the same number of boundary vertices, but different number of edges in the cut, the communication volume remains the same. An increment of 30% in the edge cut does not affect the amount of data to be transferred. However,

reducing the number of boundary vertices does not always reduce the communication volume as depicted next. Figure 5.3 shows some examples with different metrics. In b) the number of boundary vertices is 7 and the communication volume is 10 as in c), but c) has 8 boundary vertices. c) and d) have the best edge cut with only 6 edges; however, d) has the highest communication volume of all.

The right vertices and edges must be in the boundary in order to reduce the communication volume. The edge cut is not an accurate metric to reduce the communication volume. A novel idea is proposed to address the limitations of undirected graphs based on the previous analysis. The next section describes the main concept of the proposed approach.

## 5.2. Concept

The idea is simple: to find the correct vertices which reduce the amount of data to be transferred. In order to achieve this goal, it is necessary to use a new approach; the communication dependencies must be included into the partition process.

Using the data structures in METIS, which are designed for undirected graphs, we can simulate directed versions of them. This is possible due to the fact that each edge and its weight are stored twice; i.e., $(u, v)$ is stored independently of $(v, u)$. We can take advantage of this situation to mimic the direction and source of the communication between every pair of vertices connected by an edge during the coarsening process. By using independent values for edges $(u, v)$ and $(v, u)$, we can designate the amount and direction of the communication. This is more evident in the coarsest graph, and affects the initial partition. Later, when the partition is projected to the original graph, the benefits of this bidirectional graph are evident.

We follow the original process to partition a graph in METIS with some key differences. When two vertices are matched, the communication dependencies are considered in the decision. This new condition is implemented in combination with the coarsening process, where the weights of the edges and the direction of communication are calculated for the coarser graph. The values for the new edges, in the coarser graph, are calculated based on the communication dependencies. In the next coarsening level, these values are used to create the matching and the new coarser graph. A full description of the process is presented in the next chapter.

## 5.3. Challenges

There exist common challenges in the design, implementation and use of new graph partitioning methods; those are not the scope of this thesis. We, however, focus on the particular challenges of the concept mentioned earlier. An undirected graph, by nature, cannot represent directional communication. This brings an important question. How can directional communication be introduced while partitioning an undirected graph? An option is the use of a new data structure to store the information, but this comes with a cost: higher memory requirements. With a deep analysis of the data structures in METIS, and how they are used, it is possible to device a method to embed the new information within the original data structures.

Being able to store information related to directional communication within the same data structure is only part of the challenge. The next question is about the type of information that should be stored.

There is a limited amount of data that can be included without incurring in extra memory cost. The most basic, yet significant communication dependencies information shall be selected.

It is always possible to include as much information as possible for the partitioning process; better partitions can be obtained if more information is available in the decision process. As stated before, the complexity and time to generate a partition is largely increased. The inclusion of the new information should not reduce the performance of the partitioning process, and yet, improve the quality of the result.

## 5.4. Proposed Model

To address the limitations of METIS, and the inadequate undirected graphs, we propose a novel matching model called DSHEM; the main contribution of this thesis. It is designed around the idea presented in Section 5.2 and addresses the challenges in Section 5.3. A brief description is provided next; we refer the reader to Chapter 6 for a full description.

The data structures used by METIS store each edge twice, but with the exact same weight; i.e., $(u, v)$ is stored independently of $(v, u)$. DSHEM takes advantage of this situation and processes each edge in an independent manner. This new method allows the possibility of different values for edge $(u, v)$ and $(v, u)$, adding information of directional communication to the undirected graph. In addition, the origin of the communication can also be identified during the coarsening process. With the new information, the utility function in DSHEM can generate more accurate decisions and improve the quality of the partition. The rest of the partitioning process remains untouched.

The next chapters provide a detailed description of DSHEM and a comprehensive analysis of its performance.

# Chapter 6.

# Directed Sorted Heavy Edge Matching

This chapter proposes a new vertex matching model for the multilevel graph partitioning technique which aims to reduce the communication volume by simulating a directed graph. Based on the SHEM strategy implemented in METIS, the new model includes information of the direction of the communication in the matching decisions to reduce the overall communication volume. It is the result of a comprehensive analysis of current solutions and their limitations presented in Part II, Literature Review. It is a novel approach as it entirely changes the paradigm in METIS by introducing bidirectional communication to an undirected graph. By doing so, more information is available to improve the quality of partitions. It, however, does not increase the memory requirements of METIS.

Section 6.1 illustrates, with a small example, how HEM[2] and SHEM create the series of coarser graphs during the matching and coarsening phases. Next, in Section 6.2, the idea behind DSHEM and how it simulates a directed graph to produce more efficient partitions, in terms of communication volume, is presented; it addresses the limitations of SHEM. It is then compared to SHEM following all the steps of the coarsening and matching phases with another small graph. The rest of the sections provide implementation details as well as an analysis of the algorithm.

## 6.1. Sorted Heavy Edge Matching

First introduced in [124], [125], HEM removes the heaviest edges during the matching and subsequent creation of the coarser graphs; this helps ensure a smaller edge cut in the final partition. SHEM is a variation of HEM with the key difference that the vertices are visited in a sorted order based on their degrees instead of doing it randomly. Studies [68], [90], [91], [102], [103], [124], [125] have shown the superiority of SHEM over HEM and eventually HEM was dropped from METIS since version 5.0. SHEM has been improved over the time and became the de facto algorithm in METIS. The complete description of these algorithms can be found in [68] and in Appendix A where HEM and SHEM are presented in detail.

### 6.1.1. Description

Karypis and Kumar [124], [125] have demonstrated that removing the heaviest edges, while generating the series of coarser graphs, decreases considerably the edge cut of the initial partition in the coarsest

[2] Heavy Edge Matching (HEM) is an earlier implementation, later replaced by SHEM. The key difference is that SHEM visits the vertices in a sorted order based on their degree while HEM does it in a random manner.

graph. HEM visits the vertices in random order. Vertex $u$ is matched with an unmatched vertex $v$ such that the weight of the edge $(u, v)$ is maximum over all valid incident edges. Since the coarser graph has smaller edge weight, it has a smaller edge cut. The reduction of boundary vertices is performed during the uncoarsening phase in METIS. METIS uses efficient data structures to store the graph and the series of coarser graphs for the partitioning process; we refer the reader to [36], [37] for more details. These data structures are designed for undirected graphs with each edge and its weight being stored twice; i.e., $(u, v)$ is stored independently of $(v, u)$, but always with the same value.

An example of the matching process of HEM is depicted in Figure 6.1. The original graph in a) contains 7 vertices and 11 edges; the vertices are numbered and the values next to every edge represent their weight. If vertex 4 is randomly visited, after analysis, the heaviest incident edge is $(4,7)$; hence, HEM matches vertex 4 to vertex 7, as depicted in b). The matched vertices are enclosed by doted lines. Assuming vertex 6 is visited next, its heaviest incident edge is $(6,4)$, however, vertex 4 is already matched to vertex 7. In fact, only vertex 1 is available, therefore vertex 6 is matched to it; as depicted in c). Finally, when vertex 3 is visited the heaviest incident edge is $(3,5)$, producing the matching in d). The remaining vertex 2 cannot be matched due to the lack of available unmatched vertices.

After the maximal matching has been computed, METIS collapses the edges and creates a coarser graph; with the new smaller graph, the matching process starts all over again. Once a threshold is reached and the coarsest graph is generated, METIS creates the initial partition.

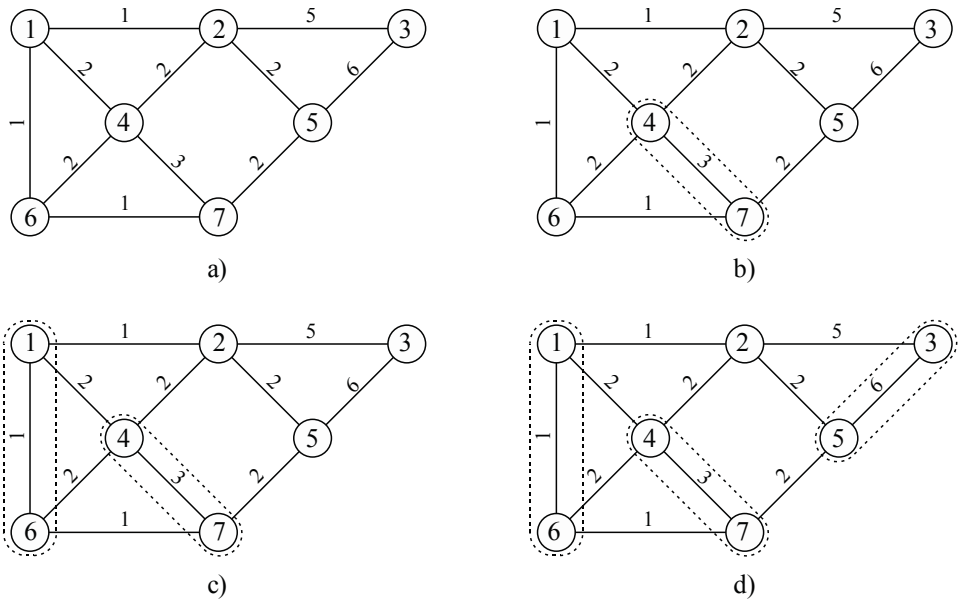


**Figure 6.1. Matching process with HEM. A sample graph with weighted edges is shown in a). When vertex 4 is visited, its heaviest edge produces the match with vertex 7 as shown in b). Next, in c), vertex 6 is visited and matched to the only available adjacent vertex, 1. Finally in d), vertex 3 is matched to vertex 5 due to its heaviest incident edge. Vertex 2 remains unmatched.**

SHEM follows the same principle visiting the vertices and matching them with the adjacent vertex having the heaviest edge. The difference, between them, is the order in which the vertices are visited. Hence, a description on how SHEM works is not necessary; though, it is important to understand the complete coarsening process. Figure 6.2 presents a suitable example to explain the full matching and coarsening process in METIS. a) shows the original graph which is, as a matter of fact, stored by METIS as depicted in c) using the data structures for undirected graphs. The values in parenthesis represent the weights of the edges; the weight of edge $(1,2)$ is 1, as well as that of edge $(2,1)$. With this

representation, the graph can also be seen as a *directed graph* as shown in b), with edges $(u, v)$ and $(v, u)$ always having similar weights. One possible matching produced by SHEM is shown in d); the matched vertices are enclosed by doted lines. Once the maximal matching has been computed, the matched vertices are collapsed together to form a coarser graph. Vertices 1 and 2 are collapsed to form a new coarse vertex, called super vertex $A$ in this case. All incident edges to both vertices are preserved in e), only edge $(1,2)$ is removed. The same process applies to vertices 5 and 6 to obtain the coarse graph in f). It can be seen that super vertex $A$ in g) has three edges; one from the original vertex 1 and two from the original vertex 2 in d). Super vertex $B$ has similar characteristics. Now, the matching process can continue with the new coarser graph generating the matching in h). This leads to the final, and coarsest, graph in i). The new edge $(C, D)$ has a weight of 3 due to the combine weights of edges $(A, 3)$, $(A, B)$ and $(4, B)$ in f). With the same logic, edge $(D, C)$ has a weight of 3; consistent with an undirected graph.

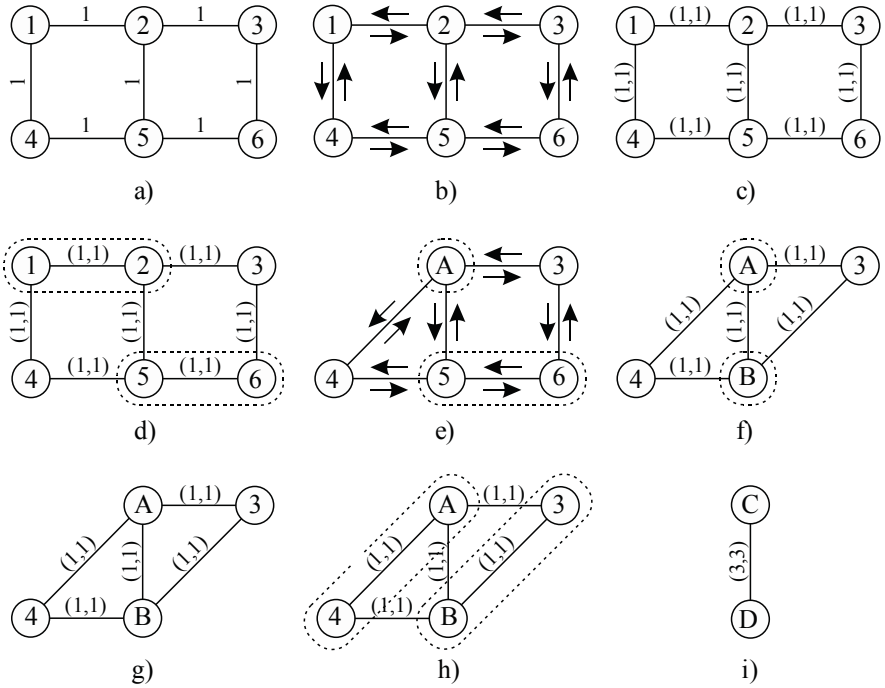


**Figure 6.2. Coarsening process with SHEM. A sample graph with weighted edges is shown in a). The actual way of storing the graph is depicted in b) and c). The first matching can be seen in d). The first coarsening phase of the graph, with the directional weights, is shown in e) and f). The coarsening process continues in g). h) depicts a SHEM matching. i) shows the coarsest, and final, graph with an edge cut of 3 (communication volume of 6).**

Once the coarsest graph is generated, METIS can calculate the initial partition of the graph. If the coarsest graph in i) is bisected, the resulting edge cut will be 3 as the coarse edge represents 3 individual edges in the original graph in a). The next step performed by METIS is the refinement process and the projection of the initial partition back to the original graph.

### 6.1.2. Algorithm

HEM, shown in Table 6.1, is one of the first algorithms implemented in METIS. It is simple and fast at creating partitions with a small edge cut. Improvements to the algorithm have resulted in SHEM, the de facto algorithm in METIS. The information required by HEM is the structure of the graph and the maximum weight that is allowed for a vertex during the coarsening process.

HEM first initializes the array $perm$ with a random permutation of the vertices; line 2 of the algorithm. Next, the for loop of lines 3 to 15 visits all vertices according to the order of the random permutation in $perm$. If the current vertex $i$ from $perm$ is $UNMATCHED$ then a match has to be found; lines 4 to 14. The first step is to match vertex $i$ to itself; it is done in line 5, bearing in mind the possibility that no appropriate match could be found. All adjacent vertices are evaluated with the purpose of finding the best match for vertex $i$ in lines 6 to 12. The decision to match vertex $i$ to adjacent vertex $j$ is defined by the conditional in lines 7 to 11. If adjacent vertex $j$ is $UNMATCHED$, and the weight of edge $(i, j)$ is maximum, and the combined weight of vertices $i$ and $j$ does not exceed the maximum weight of a coarse vertex, then vertex $i$ is matched to vertex $j$. After all adjacent vertices have been evaluated and vertex $i$ is matched, the number of coarse vertices is updated in line 13. Finally, when all vertices have been matched, the algorithm returns the maximal matching in line 16.

**Table 6.1: HEM simplified algorithm**

| **Algorithm** | Heavy Edge Matching |
|---|---|
| **Input:** | Maximum weight allowed for a vertex<br>Structure with information of the graph |
| **Output:** | Array with maximal matching of the graph<br>Number of new coarse vertices |

```
1:  procedure MATCH_HEM
2:    initialize perm with a random permutation of the vertices
3:    for each vertex i in perm do
4:      if vertex i is UNMATCHED then
5:        match vertex i to itself
6:        for each adjacent vertex j of vertex i do
7:          if adjacent vertex j is UNMATCHED and
8:          weight of adjacent edge is increased and
9:          maximum weight of a coarse vertex is not exceeded then
10:           match vertex i to adjacent vertex j
11:         end if
12:       end for
13:       increase the number of coarse vertices
14:     end if
15:   end for
16:   return
17: end procedure
```

SHEM is a modified version of HEM, which has been improved over the time on each release of METIS. SHEM visits the vertices in a sorted manner according to their degree, which is calculated in advance. The first release of SHEM visited the vertices with highest degree first, however recent implementations do it in reverse order: vertices with the lowest degree first.

It is worth mentioning that the *original version* of SHEM, depicted in Table 6.2, is actually a newer version of the algorithm. It is, however, more adequate for the purpose of describing and comparing

SHEM and DSHEM. This version is referred as the original version of the SHEM algorithm throughout this work. The computation of vertex degrees, the subsequent ordering of the vertices, and the matching of islands[3] are the core differences with HEM. Contrary to HEM, these islands are now properly matched leading to better partitions. The information required by SHEM is similar to HEM, the structure of the graph and the maximum weight that is allowed for a vertex during the coarsening process. Two vertices will only be matched if their combined weight does not exceed the specified maximum weight; nonetheless, the islands are excluded from this restriction with the aim of improving the final maximal matching by not leaving isolated vertices. The light gray text in Table 6.2 is the shared code between HEM, in Table 6.1, and SHEM, and the black text represents the new or modified code present in SHEM.

**Table 6.2: SHEM simplified algorithm (original version)**

| **Algorithm** | Sorted Heavy Edge Matching (original version) |
|---|---|
| **Input:** | Maximum weight allowed for a vertex |
| | Structure with information of the graph |
| **Output:** | Array with maximal matching of the graph |
| | Number of new coarse vertices |

```
1:  procedure MATCH_SHEM
2:    initialize perm with a sorted permutation of the vertices according to degree
3:    initialize rperm with a reversed permutation of perm
4:    for each vertex i in perm do
5:      if vertex i is UNMATCHED then
6:        if vertex i is not ISLAND then
7:          break
8:        end if
9:        match vertex i to itself
10:       for each vertex j in rperm do
11:         if vertex j is UNMATCHED and
12:         vertex j is not ISLAND then
13:           match vertex i to vertex j
14:           break
15:         end if
16:       end for
17:       increase the number of coarse vertices
18:     end if
19:   end for
20:   for each vertex i in perm do
21:     if vertex i is UNMATCHED then
22:       match vertex i to itself
23:       for each adjacent vertex j of vertex i do
```

[3] Islands are vertices with a degree equal to zero (0). They do not have incident edges or adjacent vertices.

```
24:            if adjacent vertex j is UNMATCHED and
25:            weight of adjacent edge is increased and
26:            maximum weight of a coarse vertex is not exceeded then
27:              match vertex i to adjacent vertex j
28:            end if
29:          end for
30:          increase the number of coarse vertices
31:        end if
32:      end for
33:      return
34:    end procedure
```

The first step performed by SHEM, in line 2, is to calculate the degree of each vertex. The degree values are limited to a maximum of 70% of the actual average degree in the graph; it translates into equal treatment for all vertices with high degree. Then the array $perm$ is populated with a sorted permutation of the vertices according to the degrees previously computed. Line 3 of the algorithm reverses the permutation and stores it in $rperm$; it is required during the matching of islands. The for loop of lines 4 to 19 treats the islands if they exist. All vertices are visited according to the permutation in $perm$; if vertex $i$ is $UNMATCHED$ then a match is needed. The conditional of lines 6 to 8 guarantees that the current vertex $i$ is an island, otherwise the loop is terminated. The array $perm$ is sorted and, if islands exist, they are located in the first positions due to their degree being zero. Once it is confirmed that vertex $i$ is an island and $UNMATCHED$, it is first matched to itself, in line 9, in case no other option is available. The vertices are then visited, using the for loop of lines 10 to 16, in order to find a proper match for vertex $i$, the island. Vertices with the highest degrees are located in the first positions of array $prerm$. The first $UNMATCHED$ and non island vertex $j$ is selected to match with vertex $i$ in the conditional of lines 11 to 15. There is no restriction of the combined weight of vertices $i$ and $j$. Once vertex $i$ has been matched, the number of coarse vertices is updated in line 17.

The rest of the algorithm follows the same logic as HEM with the only difference being the permutation order in $perm$. All the vertices are visited, if vertex $i$ is $UNMATCHED$ then all adjacent vertices $j$ are evaluated to find a match according to the conditions in lines 24 to 28.

The release of version 5 of METIS brought important enhancements to SHEM; the entire partitioning library was redesigned. The logic behind SHEM remains, nonetheless the majority of the code is new. The method to match islands was optimized; multi-constraint support and 2-hop matching were added as well. It is referred as multi-constraint when a vertex has multiple weights. 2-hop matching allows vertex $i$ to match with a non-adjacent vertex $k$, as long as vertex $i$ is adjacent to vertex $j$ and vertex $j$ is adjacent to vertex $k$.

The light gray text in Table 6.3 is the shared code between the original and the enhanced version of SHEM, the black text represents the new or modified code existing in the enhanced version of SHEM. Lines 2 to 6 calculate the 2-hop keys for the eligible vertices. Lines 12 to 14 find suitable matches for the islands. The rest of the code is divided in two main sections, one for a single constraint and the other for multiple constraints; the code of these two sections is almost identical. The next paragraphs describe, in detail, the enhanced version of the SHEM algorithm.

**Table 6.3: SHEM simplified algorithm (enhanced version)**

| **Algorithm** | Sorted Heavy Edge Matching (enhanced version) |
|---|---|
| **Input:** | Maximum weight allowed for a vertex |
| | Structure with information of the graph |
| **Output:** | Array with maximal matching of the graph |
| | Number of new coarse vertices |

```
 1:  procedure MATCH_SHEM
 2:    for each vertex i in the graph do
 3:      if maximum degree for 2-hop matching is not exceeded then
 4:        compute 2-hop key for vertex i
 5:      end if
 6:    end for
 7:    initialize perm with a sorted permutation of the vertices according to degree
 8:    for each vertex i in perm do
 9:      if vertex i is UNMATCHED then
10:        match vertex i to itself
11:      if maximum weight of a coarse vertex is not exceeded then
12:        if vertex i is ISLAND then
13:          match vertex i to next available UNMATCHED vertex j
14:        else /*Current vertex is not island*/
15:          if number of constraints is 1 then
16:            for each adjacent vertex j of vertex i do
17:              if adjacent vertex j is UNMATCHED and
18:              weight of adjacent edge is increased and
19:              maximum weight of a coarse vertex is not exceeded then
20:                match vertex i to adjacent vertex j
21:              end if
22:            end for /*Goes through all adjacent vertices*/
23:            if vertex i is matched to itself and
24:            maximum degree for 2-hop matching is not exceeded then
25:              for each adjacent vertex j of vertex i do
26:                for each adjacent vertex k of vertex j do
27:                  if vertex k is not vertex i and
28:                  adjacent vertex k is UNMATCHED and
29:                  both 2-hop keys are similar and
30:                  both degrees are similar and
31:                  maximum weight of a coarse vertex is not exceeded then
32:                    match vertex i to adjacent vertex k
33:                    break
34:                  end if
35:                end for
```

```
36:                 if vertex i is not matched to itself then
37:                   break
38:                 end if
39:               end for /*Goes through all adjacent vertices*/
40:             end if /*2-hop matching*/
41:           else /*Multiple constraints*/
42:             for each adjacent vertex j of vertex i do
43:               if adjacent vertex j is UNMATCHED and
44:               (weight of adjacent edge is increased or
45:               (weight of adjacent edge is preserved and
46:               partition balance is improved)) and
47:               maximum weight of a coarse vertex is not exceeded then
48:                 match vertex i to adjacent vertex j
49:               end if
50:             end for /*Goes through all adjacent vertices*/
51:             if vertex i is matched to itself and
52:             maximum degree for 2-hop matching is not exceeded then
53:               for each adjacent vertex j of vertex i do
54:                 for each adjacent vertex k of vertex j do
55:                   if vertex k is not vertex i and
56:                   adjacent vertex k is UNMATCHED and
57:                   both 2-hop keys are similar and
58:                   both degrees are similar and
59:                   maximum weight of a coarse vertex is not exceeded then
60:                     match vertex i to adjacent vertex k
61:                     break
62:                   end if
63:                 end for
64:                 if vertex i is not matched to itself then
65:                   break
66:                 end if
67:               end for /*Goes through all adjacent vertices*/
68:             end if /*2-hop matching*/
69:           end if /*One constraint*/
70:         end if /*Current vertex is island*/
71:       end if /*Current vertex weight not big*/
72:       increase the number of coarse vertices
73:     end if /*Current vertex not matched*/
74:   end for /*Goes through all vertices*/
75:   return
76: end procedure
```

The for loop of lines 2 to 6 computes the 2-hop keys for each vertex if their degree does not exceed a predefined value, i.e., if the vertices meet the requirements for 2-hop matching. Then the array $perm$ is populated with a sorted permutation of the vertices according to their degrees in line 7. The for loop, lines 8 to 74, visits the vertices of the graph according to the sorted permutation in $perm$. If vertex $i$ is $UNMATCHED$, in line 9, then it is first matched to itself in line 10; this contemplates the possibility that no other vertex could be available for the match. If the weight of vertex $i$ does not exceed the maximum allowed for a coarse vertex, in line 11, then the process to find a suitable match for vertex $i$ continues. To prevent oversized coarse vertices, vertex $i$ stays matched to itself when the maximum allowed weight is exceeded. This verification handles vertices with one or multiple constraints. The lines 12 to 14 match the islands, if they exist, to the first available vertex $j$; there is no restriction of the combined weight of vertices $i$ and $j$. The code from lines 15 to 69 is divided into two main sections: lines 16 to 40 for single constraint and 42 to 68 for multiple constraints. The two sections follow the same logic and share most of the code. The single constraint section is described next.

The adjacent vertices of vertex $i$ are then visited, using the for loop in lines 16 to 22, in order to select the adjacent vertex $j$ with the heaviest edge to match with vertex $i$. If adjacent vertex $j$ is $UNMATCHED$, and the weight of the edge $(i, j)$ is maximum, and the combined weight of vertices $i$ and $j$ does not exceed the maximum allowed for a coarse vertex then vertex $i$ is matched to vertex $j$; as defined in lines 17 to 21. If no match was found with the normal process, then 2-hop matching is employed when vertex $i$ meets the requirements. The verification is done in lines 23 and 24, vertex $i$ is matched to itself and it is suitable for 2-hop matching. The adjacent vertices of vertex $i$ are then visited again, using the for loop in lines 25 to 39, in order to select a vertex $k$ suited to match with vertex $i$. Vertex $k$ is adjacent to vertex $j$, which in turn is adjacent to vertex $i$; this is the idea behind 2-hop matching. The adjacent vertices of vertex $j$ are then visited, using the for loop in lines 26 to 35, in order to select a vertex $k$ to match with vertex $i$. If vertex $k$ is not vertex $i$, and vertex $k$ is $UNMATCHED$, and both vertices have the same 2-hop key, and they have the same degree, and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed then vertex $i$ is matched to vertex $k$ and the search ends; lines 27 to 34. Finally, in lines 36 to 38, the 2-hop matching process is terminated if vertex $i$ is not matched to itself anymore.

The multi-constraint section is similar to that of a single constraint described earlier. The difference lies in the two conditions of lines 43 to 47 and 55 to 59. The first condition matches vertex $j$ to vertex $i$ if adjacent vertex $j$ is $UNMATCHED$, and the weight of the edge $(i, j)$ is maximum, or the edge has the same weight but it leads to a better balance, and the combined weight of vertices $i$ and $j$ does not exceed the maximum allowed for a coarse vertex for all constraints. The second condition is practically similar; it just considers the multiple constraints of vertices.

The final and definitive matching of vertex $i$ is done after all its adjacent vertices have been examined and the 2-hop matching process completed; the number of coarse vertices is updated in line 72. Finally, line 75 returns the array with the matching information and the number of coarse vertices in the resulting coarse graph.

### 6.1.3. Limitations

The initial design of SHEM optimizes the edge cut, its only purpose. The subsequent refinement process

improves the initial partition and keeps its quality throughout the projection toward to the original graph. It has been demonstrated that the edge cut does not accurately represent the communication dependencies between subdomains. To address this issue, later releases of METIS introduce a new partitioning objective adding the ability to optimize the total communication volume. However, it has important limitations derived from the fact that it optimizes an edge-cut-based partition.

Based on the inherent limitations of SHEM, we propose a new matching model that better emulates the communications dependencies. It follows the ideas presented in Chapter 5 to address the limitations of METIS and produce higher quality partitions. The next sections describe the proposed model and an overview of its implementation and performance.

## 6.2. Directed Sorted Heavy Edge Matching

DSHEM is based on SHEM, and therefore on HEM, with a few important additions. DSHEM introduces the concept of *bidirectional communication* to the matching phase. The aim is to represent more accurately the communication between the subdomains while creating the final partition of the graph. The utility function uses this new data to create a more efficient matching which leads, in the end, to a better partition. DSHEM includes a mechanism to further improve the results by changing a few parameters during the execution of METIS. The complete description of the algorithm can be found in Appendix A where HEM and SHEM are also presented in detail.

### 6.2.1. Description

DSHEM is designed to improve the communication volume in partitions of undirected graphs. The idea behind is simple: to find the vertices which will reduce the amount of data to be transferred, as described earlier. It is based on SHEM and implemented in METIS. As previously stated, the data structures in METIS are designed for undirected graphs, but they can be used to simulate directed versions of them. Each edge and its weight are stored twice; i.e., $(u, v)$ is stored independently of $(v, u)$. DSHEM takes advantage of this situation to mimic the direction and source of the communication between every pair of vertices connected by an edge.

As in SHEM, all vertices are visited in a sorter manner in DSHEM. Vertex $u$ is matched with an unmatched vertex $v$ such that the weight of the edge $(u, v)$ is maximum over all valid incident edges and the communication volume is reduced. This new condition is implemented in combination with the coarsening process, where the weights of the edges and the direction of communication are calculated for the coarser graph.

An example of the matching and coarsening process of DSHEM is depicted in Figure 6.3. The original graph in a) can be represented as in c) based on the real bidirectional communication shown in b). The values in parenthesis represent the weights of the edges. Vertex 1 sends a communication volume of 1 to vertex 2, and vertex 2 sends a volume of 1 to vertex 1. METIS stores the weight of each edge twice; therefore, we do not incur in extra memory usage or extra computation for this new representation. Having the simulated directed graph, the matching and coarsening process can be performed. The first matching produced by DSHEM, in d), would be similar to that produced by SHEM. The matched vertices are enclosed by doted lines. At this point, the simulated graph does not have full

direction or source information of communications. Vertices 1 and 2 are collapsed to form a new coarse vertex, called super vertex $A$. All incident edges to both vertices are preserved to form the coarser graph in e), only edge $(1,2)$ is removed; similar process is done to collapse vertex 5 and 6 as shown in f). Now, the coarsening process updates the weight of all edges in the coarser graph by adding information concerning to direction and source. To clarify how this new information is calculated, the arrows originated from each super vertex are grouped into two categories according to their real origin. In e), observing the arrows originated from super vertex $A$, one can appreciate that some are solid black and others are white. Solid black arrows are those belonging to the original vertex 1, while white arrows belong to vertex 2. This grouping is translated into f); note that some edges have *negative* values. This minus sign (-) is used to identify the real source of the edge, and not to indicate a negative value. DSHEM collapses two vertices, and only two, to form a new super vertex in the coarser graph. This makes it easy to indicate the source without using extra memory during the coarsening process. The edge $(1,4)$ incident to vertex 1 in d) is preserved as edge $(A,4)$ in e); its communication volume remains the same. To differentiate the edges incident to vertex 2 in d) which are preserved in e), the minus sign is added to their communication volume. The resulting values are then stored in the coarser graph f) as edges $(A,3)$ and $(A,B)$. To collapse vertices 5 and 6 in e) the same process is followed. The new coarser graph g) includes all the necessary information to reduce the communication volume in the next step of the partitioning process. It can be seen that super vertex $A$ in g) has three edges; one from the original vertex 1 and two from the original vertex 2 in d) as it is indicated by the minus signs. In fact, the real source cannot be established, only the fact that one edge comes from one vertex and the other two edges from the other vertex in the finer graph. Vertex $B$ in f) has similar characteristics.

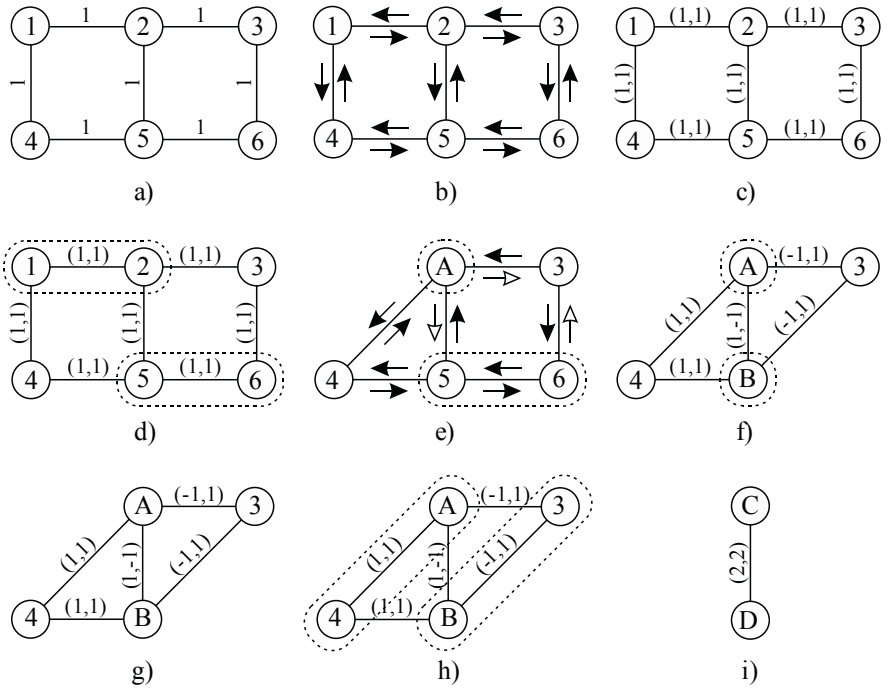


**Figure 6.3. Coarsening process with DSHEM. A sample graph with weighted edges is shown in a). A bidirectional communication is adapted to the graph in b) and c). The first matching can be seen in d). The first coarsening phase of the graph, with the new directional weights according to the source, is shown in e) and f). The coarsening process continues in g), but now with directional and source information. h) depicts a DSHEM matching reducing the communication volume. i) shows the coarsest, and final, graph with a communication volume of 4.**

The final matching depicted in h) is important as it describes the idea behind DSHEM. With the

complete information of direction and source, we can improve the partition reducing the volume of information to be transferred. If SHEM were used then all edges would have a weight of 1; their current weight is $(1,1)$ with source information (the minus signs). If we visit vertex $A$ to find a match, in h), all three neighbors are inspected (vertices 3, 4 and $B$). If vertex $A$ is matched with vertex 3 or $B$, it can be seen that the resulting edges incident to the collapsed vertex will have three or four different sources (two sources from vertex $A$, and one from vertex 3 or two from vertex $B$), but if vertex $A$ is matched to vertex 4 the resulting edges will have only two sources (one source from vertex $A$ and one from vertex 4). DSHEM will match vertex $A$ to vertex 4 reducing the number of sources in the resulting coarser vertex; SHEM would match vertex $A$ with any vertex because it only considers the weight of the edge that will be removed.

If we take vertex 4 in h) as reference, DSHEM will match it with vertex $A$ while SHEM would choose any of the neighbors resulting in a degradation of the communication volume. The next coarsening step produces the coarsest graph shown in i) with a communication volume of 4. Vertices $A$ and 4 in h) are collapsed to form super vertex $C$ in i). The edges incident to vertex $A$ are also collapsed and their weights added. Due to the fact that both edges incident to vertex $A$ have the same source, we just count them as one edge going out from it. The resulting weight for that edge $(C, D)$ is 2 (1 from the edge incident to vertex $A$ and 1 from the edge incident to vertex 4). The same process is done for the contraction of vertices 3 and $B$.

### 6.2.2. Algorithm

DSHEM shares the majority of the code with the enhanced version of SHEM. The fundamental difference lies on the condition that decides when the adjacent vertex $j$ is matched to vertex $i$. Most of the new code in DSHEM gathers information necessary to the conditional mentioned above.

The algorithm in Table 6.4 distinguishes the new or modified code in DSHEM, the black text, from the inherited code from SHEM, the light gray text. We refer the reader to the previous section for the description of this share code. The information required by DSHEM is the structure of the graph, the maximum weight that is allowed for a vertex during the coarsening process, and three percentage values. These percentages are used to fine tune how much the weight of the edge, combined with the number of sources, affects the decision to match vertex $j$ to vertex $i$. The algorithm returns the array with the matching information and the number of coarse vertices.

The for loop in lines 2 to 5 computes the number of positive and negative edges incident to every vertex in the graph. This information is later required to determine the number of sources that would be generated as a result of vertex $i$ being matched to adjacent vertex $j$; with fewer sources the total amount of communication is also reduced. Further in the code, the same two sections from SHEM are present: the first one for the single constraint case, lines 20 to 51, and the second one for the multi-constraint case, lines 53 to 87. These two sections share most of the code and only the single constraint is described here; the multi-constraint section uses a similar approach. It is important to note that DSHEM keeps the same code for islands and 2-hop matching from SHEM.

The line 21 finds the edge $(j, i)$, i.e., edge $(i, j)$ in the opposite direction. These edges have different values in DSHEM, as opposed to SHEM where their values are always identical. Line 22 computes the number of sources for the case of vertex $j$ matching vertex $i$. The conditional in lines 23 to 32 decides

whether vertex $j$ is matched to vertex $i$ taking into consideration the new collected information. With the data acquired previously, DSHEM can make a more informed decision whether vertices $i$ and $j$ should be matched together. It considers the three possible scenarios: the number of sources is reduced, preserved or increased. The weight of the edge, in combination with the percentages, affects the decision whether increasing the number of sources is preferred over reducing them. It could be more beneficial to remove an over weighted edge, at the cost of increasing the sources, rather than removing a very light edge. Removing heavy edges in the coarsening process also reduces the overall communication in the coarsest graph. The conditional in lines 23 to 30 is that of SHEM with the addition of the three different scenarios described earlier. Instead of considering only the weight of the edge, the number or sources and percentage are included.

**Table 6.4: DSHEM simplified algorithm**

| **Algorithm** | Directed Sorted Heavy Edge Matching |
|---|---|
| **Input:** | Maximum weight allowed for a vertex |
| | Structure with information of the graph |
| | Structure with matching percentages for DSHEM |
| **Output:** | Array with maximal matching of the graph |
| | Number of new coarse vertices |

```
1:  procedure MATCH_DSHEM
2:    for each vertex i in the graph do
3:      count all positive edges of vertex i
4:      count all negative edges of vertex i
5:    end for
6:    for each vertex i in the graph do
7:      if maximum degree for 2-hop matching is not exceeded then
8:        compute 2-hop key for vertex i
9:      end if
10:   end for
11:   initialize perm with a sorted permutation of the vertices according to degree
12:   for each vertex i in perm do
13:     if vertex i is UNMATCHED then
14:       match vertex i to itself
15:       if maximum weight of a coarse vertex is not exceeded then
16:         if vertex i is ISLAND then
17:           match vertex i to next available UNMATCHED vertex j
18:         else /*Current vertex is not island*/
19:           if number of constraints is 1 then
20:             for each adjacent vertex j of vertex i do
21:               find edges (i, j) and (j, i)
22:               compute the number of sources for vertices i with j
23:               if adjacent vertex j is UNMATCHED and
```

```
24:             ((number of sources is reduced and
25:             weight of adjacent edge is increased by percentage 1) or
26:             (number of sources is preserved and
27:             weight of adjacent edge is increased by percentage 2) or
28:             (number of sources is increased and
29:             weight of adjacent edge is increased by percentage 3)) and
30:             maximum weight of a coarse vertex is not exceeded then
31:               match vertex i to adjacent vertex j
32:             end if
33:           end for /*Goes through all adjacent vertices*/
34:           if vertex i is matched to itself and
35:           maximum degree for 2-hop matching is not exceeded then
36:             for each adjacent vertex j of vertex i do
37:               for each adjacent vertex k of vertex j do
38:                 if vertex k is not vertex i and
39:                 adjacent vertex k is UNMATCHED and
40:                 both 2-hop keys are similar and
41:                 both degrees are similar and
42:                 maximum weight of a coarse vertex is not exceeded then
43:                   match vertex i to adjacent vertex k
44:                   break
45:                 end if
46:               end for
47:               if vertex i is not matched to itself then
48:                 break
49:               end if
50:             end for /*Goes through all adjacent vertices*/
51:           end if /*2-hop matching*/
52:         else /*Multiple constraints*/
53:           for each adjacent vertex j of vertex i do
54:             find edges (i, j) and (j, i)
55:             compute the number of sources for vertices i with j
56:             if adjacent vertex j is UNMATCHED and
57:             (((number of sources is reduced and
58:             weight of adjacent edge is increased by percentage 1) or
59:             (number of sources is preserved and
60:             weight of adjacent edge is increased by percentage 2) or
61:             (number of sources is increased and
62:             weight of adjacent edge is increased by percentage 3)) or
63:             ((number of sources is preserved and
64:             weight of adjacent edge is preserved) and
```

```
65:                 partition balance is improved)) and
66:                 maximum weight of a coarse vertex is not exceeded then
67:                   match vertex i to adjacent vertex j
68:                 end if
69:               end for /*Goes through all adjacent vertices*/
70:               if vertex i is matched to itself and
71:               maximum degree for 2-hop matching is not exceeded then
72:                 for each adjacent vertex j of vertex i do
73:                   for each adjacent vertex k of vertex j do
74:                     if vertex k is not vertex i and
75:                     adjacent vertex k is UNMATCHED and
76:                     both 2-hop keys are similar and
77:                     both degrees are similar and
78:                     maximum weight of a coarse vertex is not exceeded then
79:                       match vertex i to adjacent vertex k
80:                       break
81:                     end if
82:                   end for
83:                   if vertex i is not matched to itself then
84:                     break
85:                   end if
86:                 end for /*Goes through all adjacent vertices*/
87:               end if /*2-hop matching*/
88:             end if /*One constraint*/
89:           end if /*Current vertex is island*/
90:         end if /*Current vertex weight not big*/
91:         increase the number of coarse vertices
92:       end if /*Current vertex not matched*/
93:     end for /*Goes through all vertices*/
94:     return
95:   end procedure
```

The multi-constraint section is similar to that of a single constraint described earlier. The difference lies in the conditional of lines 56 to 66 which considers the multiple constraints of vertices.

## 6.3. Implementation Overview

METIS employs a multilevel approach to generate the partitions; Table 6.5 presents a simplified version of the partitioning process. Once the initial configuration of the environment is done, the graph is contracted until a point where it is easy to handle; it involves matching the vertices that will collapse together to generate the next level: a coarser graph. This method generates a series of gradually smaller graphs. Then, the smallest, and coarsest, graph is divided by recursive bisection to create the initial

partition. The partition is refined, to improve its quality, and projected back to the previous level. The refinement and projection process continues until it reaches the original graph; it offers more opportunities to improve the quality by constantly refining the partition on finer graphs.

**Table 6.5: Partitioning process overview**

| **METIS** | Partitioning process |
|---|---|
| 1: | **procedure** Partition graph *k*-way |
| 2: | initializes environment |
| 3: | initializes graph |
| 4: | coarsens graph to produce a series of smaller graphs |
| 5: | computes initial partition on coarsest graph by recursive bisection |
| 6: | refine initial partition on coarsest graph and projects it back to original graph |
| 7: | **end procedure** |

### 6.3.1. DSHEM and METIS

As stated before, DSHEM produces a different set of values for the edges $(u, v)$ and $(v, u)$ in the matching and coarsening process; it is, by far, the major change within METIS. Though DSHEM is only executed in step 4: (coarsens graph to produce a series of smaller graphs) of Table 6.5, it impacts the entire process. This means that all steps require changes to be able to handle the new values produced by DSHEM. By using independent values for edges $(u, v)$ and $(v, u)$, DSHEM changes the paradigm within METIS and complicates its implementation. The code is adapted to handle new values for edges $(u, v)$ and $(v, u)$ and new parameters are added to control the execution of METIS.

The cost function of DSHEM uses three values to modify its performance. They are percentages that affect the matching according to the number of sources and the weight of the edges in three different scenarios: number of sources is reduced/maintain/increased and the weight of adjacent edge is increased.

Debug code is also included in the implementation to track the progress of the partitioning process. In some cases a new compilation is required to disable some of the code. This approach is less practical but more efficient as the executable does not contain the unwanted code making it faster.

Currently, the refinement follows the original method throughout the projection back to the original graph. It has been, however, adapted to handle the new weights of edges in the coarser graphs produced by DSHEM. The impact of the refinement process on DSHEM and its performance is an interesting focus for future research. The refinement type can be selected by the means of a new parameter. The initial purpose of this implementation is to analyze and understand the effect of DSHEM, and only DSHEM, in the partitioning process.

#### Methods Optimized for DSHEM

New changes have also been implemented on METIS to enhance, configure, and analyze the impact of DSHEM on every step of the partitioning process. Among the modifications we can mention the selection of refinement type, the coarsening process limit, and the nested partitioning.

The coarsening limit brings a new idea to METIS: to contract the graph until $n$ coarse vertices are

available. Contracting the graph up to this point *eliminates* the need to compute the initial partition as every vertex is assigned to a different subdomain. It is similar to JOSTLE, where the contraction always reaches $nparts$ vertices. Some conditions are relaxed to efficiently attain the coarsest level and not fall into a point were more than $nparts$ vertices are still available but cannot be matched.

The nested partition involves tracking two graphs in the partitioning process: one with the original SHEM values and a second one with DSHEM values. One graph is used for the coarsening process and the other for the initial partitioning and refinement. More information is provided in Section 6.3.3 where Nested DSHEM Partitioning is described.

A new parameter controls the maximum vertex weight; originally hardcoded in METIS. During the coarsening process the vertices are matched and a condition limits their weight preventing super heavy vertices. It is used to relax the conditions when the coarsening process generates a graph with $nparts$ vertices.

### 6.3.2. Full DSHEM Partitioning

It is the implementation of the original idea behind DSHEM as described in Appendix A, where HEM and SHEM are also presented in detail. DSHEM shares the majority of the code with the enhanced version of SHEM and follows the original process to partition a graph in METIS. The fundamental difference is the condition that determines if the adjacent vertex $j$ is matched to vertex $i$. Most of the new code in DSHEM gathers information necessary to the conditional mentioned above. Hence, the matching strategy is added to METIS and the rest of the code adapted to cope with the new *directed* graph generated by DSHEM.

This implementation neither increases the memory requirements nor execution time of METIS. It takes full advantage of the data structures to describe the communication within the graph without performance degradation.

Three main arguments are received by DSHEM; specifically $maxvwgt$, $graph$ and $percentages$. $maxvwgt$ specifies the maximum weight allowed for a coarse vertex. $graph$ is the structure containing all information related to the graph. $percentages$ contains three percentage values employed to fine tune the algorithm. The algorithm returns the array $match$ with the matching information and $cnvtxs$ with the number of coarse vertices.

### 6.3.3. Nested DSHEM Partitioning

Nested DSHEM is a variation of the original algorithm where a joint effort by SHEM and DSHEM is used to generate the partitions. It uses independent SHEM or DSHEM values for the coarsening process and for the initial partition. Four different combinations are possible:

- DSHEM+DSHEM: During the coarsening process, DSHEM creates the matching used to contract the graph. Once the contraction is done and a coarser graph is created, the weights of the edges on the coarser graph have the new independent values produced by DSHEM. A second coarse graph is created based on the one produced by DSHEM; with the same adjacencies but different edge weights. This new graph is an undirected version of the original one: think of a graph produced by SHEM but with DSHEM utility function. At this point two series of “equal” coarser graphs are

produced with the only difference that one is directed and the other undirected. During the initial partitioning, as well as the refinement, the directed graph produced by DSHEM is used until the end of the partitioning process. In this execution the undirected version of the graph is never used and the final partition is that of a full DSHEM partitioning.

- DSHEM+SHEM: During the coarsening process, DSHEM creates the matching used to contract the graph. Once the contraction is done and a coarser graph is created, the weights of the edges on the coarser graph have the new independent values produced by DSHEM. A second coarse graph is created based on the one produced by DSHEM; with the same adjacencies but different edge weights. This new graph is an undirected version of the original one: think of a graph produced by SHEM but with DSHEM utility function. At this point two series of "equal" coarser graphs are produced with the only difference that one is directed and the other undirected. During the initial partitioning, as well as the refinement, the undirected graph "produced" by SHEM is used until the end of the partitioning process. The first half of the partitioning process uses the directed graph while the second half uses the undirected version of the graph.
- SHEM+DSHEM: During the coarsening process, SHEM creates the matching used to contract the graph. Once the contraction is done and a coarser graph is created, the weights of the edges on the coarser graph have the normal values produced by SHEM. A second coarse graph is created based on the one produced by SHEM; with the same adjacencies but different edge weights. This new graph is the directed version of the original one: think of a graph produced by DSHEM but with SHEM utility function. At this point two series of "equal" coarser graphs are produced with the only difference that one is directed and the other undirected. During the initial partitioning, as well as the refinement, the directed graph "produced" by DSHEM is used until the end of the partitioning process. The first half of the partitioning process uses the undirected graph while the second half uses the directed version of the graph.
- SHEM+SHEM: During the coarsening process, SHEM creates the matching used to contract the graph. Once the contraction is done and a coarser graph is created, the weights of the edges on the coarser graph have the normal values produced by SHEM. A second coarse graph is created based on the one produced by SHEM; with the same adjacencies but different edge weights. This new graph is the directed version of the original one: think of a graph produced by DSHEM but with SHEM utility function. At this point two series of "equal" coarser graphs are produced with the only difference that one is directed and the other undirected. During the initial partitioning, as well as the refinement, the undirected graph produced by SHEM is used until the end of the partitioning process. In this execution the directed version of the graph is never used and the final partition is that of a full SHEM partitioning

This implementation has an impact in the memory requirements as a second graph is kept in memory during the complete process. The execution time is not noticeable impacted as the matching, coarsening and refinement steps use only one graph at a time.

## 6.4. Expected Performance

It is possible to deduce the performance of DSHEM through the analysis of its design. The quality of the partition is improved or kept intact as DSHEM includes the cost function used by SHEM; this produces a partition with a minimal edge cut, but with a possible reduction in communication volume. Refinement techniques remain intact in METIS and the inherent reduction in boundary vertices. DSHEM improves efficiency in parallel FEM computations without incurring in extra computations or memory requirements. The reduction of communications leads to faster parallel FEM computations.

DSHEM takes advantage of the original data structures that are used in METIS and no extra memory is required. By using independent values for edges $(u, v)$ and $(v, u)$, it is possible to better emulate the amount and direction of the communication dependencies. The fundamental difference is the data stored in such data structures. The nested version of DSHEM, a variation of the original algorithm, has higher memory requirements. It keeps track of two versions of the same graph during the coarsening and uncoarsening phases. However, the second graph requires a smaller amount of memory; only the weights of the edges are stored and the rest of the information is shared with the first graph.

DSHEM is based on the original SHEM and inherits its basic behavior. A different cost function is added to reduce the communication in the resulting partition. The vertices are visited in the same manner as in SHEM, but the data to analyze are different. The modifications to the coarsening phase are minimal due the introduction of the direction and source of communication. Only a few extra computations are added to deal with the new information. The resulting running time should not increase more than 5% and the memory requirements are unchanged.

DSHEM and the modified coarsening process produce a better partition if the communication volume is considered. The extra computations are minimal and do not affect the overall running time because the uncoarsening and refining process has a better partition to work with. The direction and source of communication are only tracked between two levels in the coarsening process without incurring extra memory usage. The real weights of vertices and edges are maintained in the finest graph.

To confirm the initial assessment of the performance of DSHEM, Chapter 8 and Chapter 9 provide detailed information on the experimental evaluation of the new model. It assesses its performance in controlled and real use cases based on the methodology described in Chapter 7.

# Chapter 7.

# Evaluation Methodology

The experimental environment, and its correct design, is important during the evaluation of any algorithm. It is of extreme importance to correctly select the correct setup to assess the performance and comparison with other methods. This chapter presents in detail the metrics, the real life and synthetic graphs, as well as the hardware and software setup used to evaluate DSHEM.

## 7.1. Evaluation Metrics

A number of metrics exist to evaluate the quality of a partition; the edge cut is traditional among them. Depending on the application, some of the metrics might be more important than others. In practice, it is common to find a partition that minimizes or maximizes only one metric or objective. Among the most studied in literature is the edge cut; however, it has been demonstrated that it is not adequate for certain types of problems. It is well known that the edge cut it is only an approximation of the communication costs. The communication volume metrics measure more accurately the performance of DSHEM.

This section introduces the different metrics used to evaluate the performance of DSHEM. We refer the reader to The Graph Partitioning Problem in Section 3.2.3, Load Balancing through Graph Partitioning, to understand the definitions in this section.

### 7.1.1. Total Edge Cut

Among the most employed and studied metrics in literature is the edge cut; relatively easy to optimize and with many applications. The edge cut is useful for some types of problems such as VLSI, where the reduction of the number of edges in the cut is desired. However, it is not optimal for FEM problems due to the inaccurate approximation of the communication costs. Karypis and Kumar [124], [125], among others, present an analysis of the edge cut for multilevel graph partitioning.

The total edge cut is given by $\|C\| = \sum_{e \in C} \|e\|$. It is the sum of the weights of the edges in $C$. If the edges have unitary weights (i.e., $|C| = \|C\|$) then the number of edges in the cut is minimized. The partition is optimal if there is no partition with smaller edge cut.

### 7.1.2. Total Communication Volume

As stated before in Section 5.1 Analysis, and by studies such as [100], [189], the edge cut is not the best choice for certain problems. For example, it does not model the real communication costs in FEM

applications; the actual communication can differ significantly [93]. Studies such as [104], [188]–[190] focuses on this metric. This metric is important due to the fact that often the bottleneck of parallel applications is the communication. Hence, it is related to network bandwidth usage and total execution time.

Let $B_{v_i} = \{jj | (v_i, v_j) \in C \wedge v_j \in S_{jj}\}$ the boundary subdomains of vertex $v_i$. The amount of data to be transferred by vertex $v_i$ to a neighboring subdomain is given by $|v_i|$, its size. Then, the communication volume coming from vertex $v_i$, going into all its neighboring subdomains, is given by $CommVol_{v_i} = |v_i| \times |B_{v_i}|$. Hence, the total communication volume induced by the partition $\pi$ is defined as $CommVol_\pi = \sum(|v_i| \times |B_{v_i}|)$, the sum of the communication from all vertices. Note that $|B_{v_i}| = 0$ for non boundary vertices.

### 7.1.3. Maximum Communication Volume of all Subdomains

The reduction of the total communication volume is not necessarily the best approach in FEM applications; the different subdomains may have an unbalanced volume of communication. If a subdomain holds a large percentage, the efficiency decreases as the rest of the subdomains have larger idle times. We use the maximum communication volume of all subdomains to determine that imbalance.

Let $B_{S_{ii}} = \{v_i | (v_i, v_j) \in C \wedge v_i \in S_{ii}\}$ be the boundary vertices in $S_{ii}$. Then the communication volume produced by subdomain $S_{ii}$ is given by $CommVol_{S_{ii}} = \sum_{v \in B_{S_{ii}}} CommVol_v$, the sum of the communication of all its boundary vertices, and the maximum of the communication volume of all subdomains is then given by $\max_{1 \le ii \le k} CommVol_{S_{ii}}$.

### 7.1.4. Minimum Communication Volume of all Subdomains

Less important, but a good indicator of the efficiency of the algorithm, is the minimum of the communication volume of all subdomain. Even though a small amount of communication coming from a subdomain has less impact in the overall execution time of the application, it is an indicator that the algorithm used to create the partition is not optimal. This metric is defined as $\min_{1 \le ii \le k} CommVol_{S_{ii}}$.

### 7.1.5. Average (Desired) Communication Volume of a Subdomain

It is the optimal value to achieve, a balanced communication per subdomain. The desired communication volume per subdomain is given by the average of all subdomains $\overline{CommVol} = \lceil \sum_{1 \le ii \le k} CommVol_{S_{ii}} / k \rceil$ or by the total communication volume divided by the number of subdomains $\overline{CommVol} = \lceil CommVol_\pi / k \rceil$.

### 7.1.6. Total Communication Cost

The total communication cost is obtained based on the communication volume when the weights of the edges are also taken into account. Let $C_{v_i} = \{(v_i, v_j) | (v_i, v_j) \in C\}$ be the set of edges incident to $v_i$ that

belong to the cut, and $C_{v_i S_{jj}} = \{(v_i, v_j) | (v_i, v_j) \in C_{v_i} \wedge v_j \in S_{jj}\}$ a subset of $C_{v_i}$ with vertex $v_j$ in subdomain $S_{jj}$.

Then, the total communication cost by vertex $v_i$ is given by $CommCost_{v_i} = \sum_{jj \in B_{v_i}} \left( |v_i| \times \min_{e \in C_{v_i S_{jj}}} |e| \right)$. When the edges have unitary weights, the communication cost is then reduced to $CommCost_{v_i} = \sum_{jj \in B_{v_i}} (|v_i| \times 1)$, and $CommCost_{v_i} = |v_i| \times |B_{v_i}|$. Which in turn leads to $CommCost_{v_i} = CommVol_{v_i}$. Hence, the total communication cost induced by the partition $\pi$ is defined as $CommCost_\pi = \sum CommCost_{v_i}$, the sum of the communication cost of all vertices. Note that $|B_{v_i}| = 0$ for non boundary vertices.

### 7.1.7. Maximum Communication Cost of all Subdomains

As other maximum values, it is used calculate the imbalance among the different subdomains. The communication cost of subdomain $S_{ii}$ is given by $CommCost_{S_{ii}} = \sum_{v \in B_{S_{ii}}} CommCost_v$, the sum of the communication cost of all its boundary vertices, and the maximum of the communication cost of all subdomains is $\max_{1 \le ii \le k} CommCost_{S_{ii}}$.

### 7.1.8. Minimum Communication Cost of all Subdomains

A less important, but still necessary metric, is the minimum communication cost of all subdomains. This metric is defined as $\min_{1 \le ii \le k} CommCost_{S_{ii}}$.

### 7.1.9. Maximum Weight of all Subdomains

The weight of a vertex represents the processing requirements of the respective mesh element. The most over weighted subdomain is used to calculate the imbalance of the load induced by the partition. The maximum weight of all subdomains is given by $\max_{1 \le ii \le k} \|S_{ii}\|$.

### 7.1.10. Minimum Weight of all Subdomains

It is also a good indicator of the efficiency of the algorithm used to create the partition. The minimum weight of all subdomains is given by $\min_{1 \le ii \le k} \|S_{ii}\|$.

### 7.1.11. Average (Desired) Weight of a Subdomain

It is the optimal value to achieve, a perfect distribution of the load among the processors. The desired weight of a subdomain is given by the total sum of the weights of all vertices divided by the number of subdomains $\bar{S} = \lceil \sum_{v \in V} \|v\| / k \rceil$, or by the average weight of all subdomains $\bar{S} = \lceil \sum_{1 \le ii \le k} \|S_{ii}\| / k \rceil$.

## 7.2. Graphs and Statistics

The selection of data for experimentation is of high importance and has to consider multiple factors; among them, the design and purpose of the algorithm. The wrong data could lead to a false evaluation of the algorithm under investigation. All possible scenarios need to be incorporated in order to correctly assess its performance. This is the reason why a collection of real life, as well as synthetic, graphs have been used for the experimental analysis of DSHEM and comparison with SHEM.

The set of real life graphs comes from The SuiteSparse matrix collection of the Texas A&M University [191], the 10th DIMACS Implementation Challenge [192] contributed by Siew Yin Chan et al. [193], the METIS distribution [36], the Internet parallel computing archive [194], and the Graph partitioning archive [195]. It includes a wide variety of graphs used in FEM applications, including 2D and 3D models.

Synthetic graphs are used, as well, to provide a more controlled output. They are valuable to categorize the performance of DSHEM with the different types of graphs. They are divided into regular and irregular, 2D and 3D instances. The next sections describe the different graphs used for the experimental analysis of DSHEM.

A detailed description of the graphs can be found in Appendix B, where the real life graphs are presented and the construction of the synthetic graphs is also described.

### 7.2.1. Real Life Graphs

The SuiteSparse matrix collection[4] has a wide variety of spare matrices that arise from real life applications. It is maintained in a regular basis and new matrices are constantly incorporated. The collection is frequently used by the community to evaluate the performance of new algorithms. Examples of graphs for structural engineering, fluid dynamics, thermodynamics, among others, can be found. The three real life graphs used in this work are available at the SuiteSparse matrix collection; however, they originally come from other sources such as the Internet parallel computing archive.

#### 2D Graphs

One 2D graph has been selected for the experimental evaluation of DSHEM. The graph contains the most common geometry utilized in FE computations. The size is classified as small with thousands of vertices. The next paragraphs give a brief description; the reader should refer to the appendix and the source of the graph for more information.

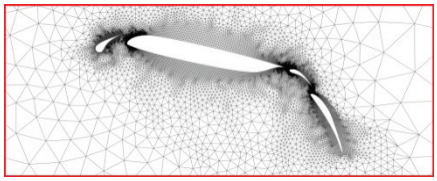

**Figure 7.1. Area of interest of the graph of an airfoil with flaps.**

Figure 7.1 shows a section of the graph of an airfoil with front slat and rear flaps; this is a small

[4] Originally known as the University of Florida sparse matrix collection [204], which is now available at the Texas A&M University web site [191].

triangular graph. It contains around 15 thousand vertices and 45 thousand edges. The dark areas represent a concentration of vertices and edges in the graph. More details are presented in the appendix.

### 3D Graphs

Two 3D graphs have been used for the experimental evaluation of DSHEM. The graphs have a quadrangular and a triangular geometry; medium and small size respectively. The next paragraphs give a brief description.

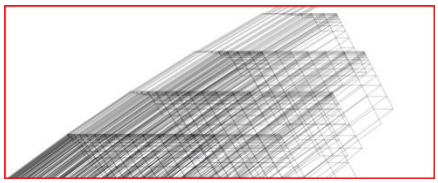

**Figure 7.2. Some area of the graph of the ocean.**

Figure 7.2 shows a mall section of the graph of the oceans; a medium size quadrangular graph. It contains over 140 thousand vertices and 400 thousand edges. More details are presented in the appendix.

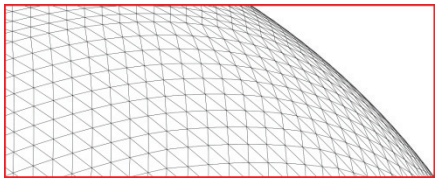

**Figure 7.3. Some area of the graph of a sphere.**

Figure 7.3 presents a mall section of the graph of a sphere; a small triangular graph used for the analysis of DSHEM. It has around 16 thousand vertices and 49 thousand edges. More details are presented in the appendix.

## 7.2.2. Synthetic Graphs

Several methods designed to generate synthetic graphs have been proposed in literature. They produce graphs with different properties, depending on its intended use. Some of the most important methods are described next. Gilbert [196] propose a method to create a *Random plane network* by employing a Poisson process to place vertices with a density $D$ in a given area. Then those vertices are connected together in pairs when their distance does not exceed a given range $R$. Barabási and Albert [197], [198] propose a method to build random networks. It is based on the observation that diverse networks expand continuously by adding new vertices and those new vertices are most likely attached to well connected vertices. The Yule process, or Preferential Attachment, [199], [200] can also be used to generate random graphs. The idea behind this method is that new edges will likely be placed with vertices with high degrees. Newman et al. [201] propose several models to create social network graphs such as unipartite networks and bipartite networks with very good approximations. The Erdős-Rényi model has two variants $G(n,p)$ [202] and $G(n,M)$ [203], The first variant creates a graph connecting the $n$ vertices randomly with a probability $p$. The second variant chooses a random graph from the set of graphs with $n$ nodes and $M$ edges. The Delaunay, or Delone, triangulation [76], [77] creates a triangular graph by

connecting a set $P$ of vertices such as no other vertex is located inside the circumcircle generated by any triangle in the graph.

The experimental evaluation of DSHEM does not include any synthetic random graph generated by the methods presented above. They are not suited for the evaluation of DSHEM because it is essential to have full control over the graph geometry to investigate how it influences the quality of the partition and performance of the algorithm. The previous methods generate synthetic random graphs which emulate their real word counterparts. They are generally used when no real data, or a limited amount of it, is available. In this case, a big collection of real life graphs is publicly available making the synthetic random graphs redundant. The decision to not use this type of graphs was driven by these factors.

Thus, to have a controlled environment to assess the performance of DSHEM, several synthetic graphs with specific characteristics have been used. The synthetic graphs used for this work are based on the most common geometries found in FEA. Triangular and quadrangular geometry graphs are created in several ways to evaluate the performance of DSHEM. They are divided into regular and irregular, as well as 2D and 3D graphs. It is possible to control the degree of randomness with these synthetic graphs and evaluate how it affects the partitions generated by DSHEM. The range of sizes varies from small to large, as with the real life graphs. A description is presented next.

### Regular

The regular synthetic graphs are used to perform a controlled evaluation of the comportment of DSHEM under several conditions. It is more evident how the geometry and size of the graph may affect its behavior, as well as the execution time. With that in mind, three different types of graphs have been selected to evaluate DSHEM; one quadrangular and two triangular geometries. The first type is a square graph, composed of regular squares as depicted in Figure 7.4. It size is measured by the number of vertices per side and it is given by $n \times n$, in this example $3 \times 3$.

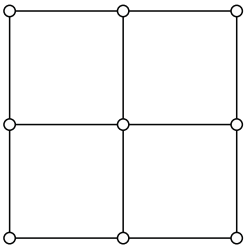

**Figure 7.4. Square graph with 3 vertices per side.**

Figure 7.5 depicts the second type of synthetic graph with a triangular geometry. Its size is also measured by the number of vertices per side, i.e., $n \times n$.

A second triangular graph with a modified geometry is also employed for the experimental analysis. Figure 7.6 presents a $3 \times 3$ dense triangular square graphs; $(n-1) \times (n-1)$ extra vertices have been added, and connected with edges, to the triangular square mesh to create the dense graph. Its size is given by the number of external vertices per side: $n \times n$.

3D versions of all three regular graphs are also created to evaluate DSHEM. Figure 7.7 shows the process to create a 3D square graph with a size of $2 \times 2$. It starts with the original graph as shown in a). Next, in b), the graph is replicated to have $n$ copies. Finally, edges are generated between the vertices of the different copies of the graph as depicted in c).

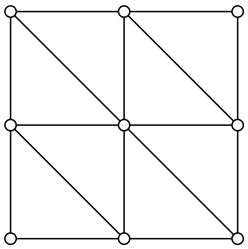

**Figure 7.5. Triangular square graph with 3 vertices per side.**

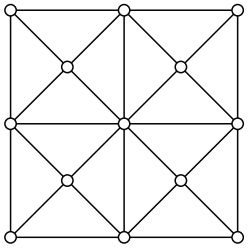

**Figure 7.6. Dense triangular square graph with 3 vertices per side.**

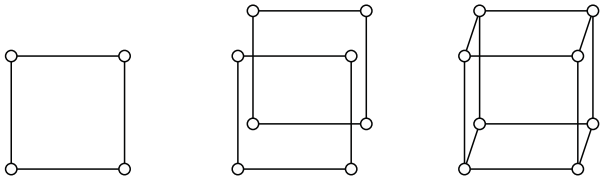


**Figure 7.7. Creation of a 3D square graph with 2 vertices per side.**

The creation of the 3D versions of the triangular square and dense triangular square graphs is described in detail in the appendix. It is important to note that the coordinate information is not used during the partitioning process; it is useful only for visualization purposes.

## Irregular

Introducing irregularity to the graphs helps understanding the robustness of the algorithm during the partitioning process. The irregular synthetic graphs are based on the regular synthetic versions. All the edges in the graph are visited and removed with a given uniform probability. Several probabilities have been used to build different irregular graphs, giving them a degree or randomness. The resulting graph is then used for the experimental analysis to assess the performance of DSHEM and compare it with SHEM. It is important to understand how the edges are removed from the graph. Once a regular graph is created, all edges are duplicated; it is the normal way the graph is stored. All edges are visited in a sequential order and removed with a given probability. This method, visits edges $(u, v)$ and $(v, u)$ independently; one edge could be removed while the other remains. After visiting all edges, the graph may be corrupted due to the fact that some of the edges are no longer *duplicated*; the graph is then corrected by duplicating those edges. The final result is a correct graph with some of the edges being removed. If the *initial* probability to remove an edge was 10%, the resulting graph will have about 5% (*final* probability) of the edges removed.

Throughout this work, the probability values for irregular graphs refer to the *initial* probability unless it is otherwise stated. Figure 7.8 shows an irregular dense triangular square graphs. It is built after the regular version by removing edges with a *final* probability of 25%. The same procedure is applied to the

other two types of graphs: square and triangular square. The 3D graphs have also undergone the same process to create an irregular version of them.

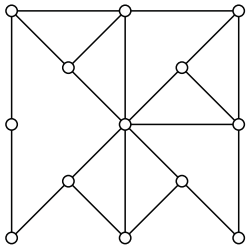

**Figure 7.8. Dense triangular square graph with 3 vertices per side. Edges were removed with a *final* probability of 25%.**

### 7.2.3. Statistics

Table 7.1 shows the properties of the real life graphs used for the experimental evaluation in this thesis. A medium size graph with quadrangular geometry is included and two small graphs with triangular geometries are also part of the experiments.

**Table 7.1: Properties of the real life graphs.**

| *Graph* | *Vertices* | *Edges* | *Description* | *Source* |
|---|---|---|---|---|
| ef_4elt | 15,606 | 45,878 | A 2D triangular mesh around an airfoil with flaps | [36], [191], [194], [195] |
| ef_ocean | 143,437 | 409,593 | A 3D quadrangular mesh of the ocean | [36], [191], [194], [195] |
| ef_sphere | 16,386 | 49,152 | A 3D triangular mesh of a sphere | [36], [191], [194], [195] |

Due to the regularity of the synthetic graphs used in this work, Table 7.2 shows their general properties. The number of vertices and edges can be calculated according to the size of the graph. In the case of the irregular graphs, the number of edges is an approximation due to the probability $p$ used to remove edges from the base regular graph. 2D and 3D graphs with triangular and quadrangular geometries are part of the experiments, with their size ranging from small to large.

**Table 7.2: Properties of the synthetic graphs.**

| *Graph* | *Vertices* | *Edges* | *Description* |
|---|---|---|---|
| sm_2d | $n^2$ | $2\big(n(n-1)\big)$ | A synthetic regular 2D quadrangular mesh |
| tsm_2d | $n^2$ | $2\big(n(n-1)\big)+(n-1)^2$ | A synthetic regular 2D triangular mesh |
| dtsm_2d | $n^2+(n-1)^2$ | $2\big(n(n-1)\big)+4(n-1)^2$ | A synthetic regular 2D triangular mesh |
| sm_3d | $n^3$ | $2\big(n(n-1)\big)n+n^2(n-1)$ | A synthetic regular 3D |

| | | | |
|---|---|---|---|
| | | | quadrangular mesh |
| tsm_3d | $n^3$ | $\big(2\big(n(n-1)\big)+(n-1)^2\big)n+$ $n^2(n-1)+(n-1)^3$ | A synthetic regular 3D triangular mesh |
| dtsm_3d | $n^3+$ $(n-1)^3$ | $\big(2\big(n(n-1)\big)+4(n-1)^2\big)(n-1)+$ $4(n-1)^3+2\big(n(n-1)\big)+n^2(n-1)$ | A synthetic regular 3D triangular mesh |
| sm_2d _perc | $n^2$ | $2\big(n(n-1)\big)$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 2D quadrangular mesh |
| tsm_2d _perc | $n^2$ | $2\big(n(n-1)\big)+(n-1)^2$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 2D triangular mesh |
| dtsm_2d _perc | $n^2+$ $(n-1)^2$ | $2\big(n(n-1)\big)+4(n-1)^2$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 2D triangular mesh |
| sm_3d _perc | $n^3$ | $2\big(n(n-1)\big)n+n^2(n-1)$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 3D quadrangular mesh |
| tsm_3d _perc | $n^3$ | $\big(2\big(n(n-1)\big)+(n-1)^2\big)n+$ $n^2(n-1)+(n-1)^3$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 3D triangular mesh |
| dtsm_3d _perc | $n^3+$ $(n-1)^3$ | $\big(2\big(n(n-1)\big)+4(n-1)^2\big)(n-1)+$ $4(n-1)^3+2\big(n(n-1)\big)+n^2(n-1)$ Minus approximately $(100-p/2)$% of the edges | A synthetic irregular 3D triangular mesh |

All real life graphs are strongly connected, symmetric undirected graphs; as can be appreciated in Table 7.3. Based on Euler's formula[5] for planar (2D) graphs, the average degree of a vertex is strictly less than 6. With a triangular geometry, the average degree of a vertex tend to be 6, the vertices in the external limits of the graph reduce that average. This situation is evident with the graph *ef_sphere* having an average of almost 6, as it does not have outer limit vertices due to its three dimensions. The graph *ef_ocean*, not planar, has the lowermost average degree due to the large amount of vertices in the external limit of the graph. It is evident that the geometry of the graph does not have big influence in the average degree of this collection of graphs. The most important factor is the ratio of vertices in the external limits of the graph and its total.

**Table 7.3: Detailed statistics of the real life graphs.**

| *Graph* | *Average degree* | *Strongly connected components* | *Symmetric* | *Kind* |
|---|---|---|---|---|
| ef_4elt | 5.880 | 1 | Yes | Undirected graph |
| ef_ocean | 5.711 | 1 | Yes | Undirected graph |
| ef_sphere | 5.999 | 1 | Yes | Undirected graph |

The same analysis for the synthetic graphs is presented in Table 7.4. The average degree of all 2D

[5] Planar graphs obey the inequality $2e \geq 3f$ based on $v-e+f=2$, Euler's formula.

graphs does not exceed 6; their 3D counterparts have an average degree of up to 11. That average is affected by the number of edges removed in the irregular instances, as well as the number of strongly connected components.

**Table 7.4: Detailed statistics of the synthetic graphs.**

| *Graph* | *Average degree* | *Strongly connected components* | *Symmetric* | *Kind* |
|---|---|---|---|---|
| sm_2d | 2.000 to 4.000 | 1 | Yes | Undirected graph |
| tsm_2d | 2.500 to 6.000 | 1 | Yes | Undirected graph |
| dtsm_2d | 3.200 to 6.000 | 1 | Yes | Undirected graph |
| sm_3d | 3.000 to 6.000 | 1 | Yes | Undirected graph |
| tsm_3d | 3.750 to 10.000 | 1 | Yes | Undirected graph |
| dtsm_3d | 4.444 to 11.000 | 1 | Yes | Undirected graph |
| sm_2d _perc | Up to 4.000 | 1 or more | Yes | Undirected graph |
| tsm_2d _perc | Up to 6.000 | 1 or more | Yes | Undirected graph |
| dtsm_2d _perc | Up to 6.000 | 1 or more | Yes | Undirected graph |
| sm_3d_perc | Up to 6.000 | 1 or more | Yes | Undirected graph |
| tsm_3d_perc | Up to 10.000 | 1 or more | Yes | Undirected graph |
| dtsm_3d_perc | Up to 11.000 | 1 or more | Yes | Undirected graph |

None of the previous graphs have either vertex or edge weights and it is assumed that all weights are unitary; this kind of information is not widely available.

## 7.3. Environment

A full overview of the experimental environment is presented in the next subsections. They detail the software, hardware and input data used to evaluate DSHEM and compare its performance with SHEM and Random.

### 7.3.1. Graph Partitioning Software

METIS version 5.0.2 is the base for the implementation of DSHEM; the latest release at the moment of the coding. The current version, at the moment of writing, is 5.1.0, however the new changes are minimal and without impact to DSHEM. The 2-hop matching strategy is extended and a few bugs are corrected; yet, it only affects graphs with variable degree distribution, not graphs derived from FE meshes.

The standalone serial METIS program is *gpmetis* which receives two obligatory parameters, the graph and number of parts, and a set of noncompulsory options. Table 7.5 lists the relevant available options and the values used for the experimentation.

**Table 7.5: List of execution options for METIS.**

| *Option* | *Description and possible values* | | *Values used* |
|---|---|---|---|
| -ptype | Scheme to compute the partition | | kway |
| | rb | Multilevel recursive bisection | |
| | kway | Multilevel *k*-way partitioning | |
| -iptype | Scheme to compute the initial partition in the coarsest graph | | grow |
| | random | Random based | |
| | grow | Greedy based | |
| -ctype | Matching algorithm for the coarsening phase | | rm<br>shem<br>dshem |
| | rm | Random | |
| | shem | Sorted heavy edge | |
| | **dshem** | Directed sorted heavy edge | |
| **-cltype** | Coarsening limit type. | | normal<br>nparts |
| | normal | Normal coarsening limit | |
| | **nparts** | The coarsening process stops only when *nparts* vertices remain | |
| -objtype | Objective function to minimize | | cut<br>vol |
| | cut | Edge cut | |
| | vol | Total communication volume | |
| **-rtype** | Algorithm used for refinement | | norefinement<br>greedy |
| | **norefinement** | No refinement at all | |
| | fm | FM based edge cut refinement | |
| | greedy | Greedy based edge cut and volume refinement | |

It is important to note that new options were added to METIS in order to fine tune its execution; shown in bold in Table 7.5.

#### Justification

Being open source, METIS is an important candidate for the implementation of new algorithms based on the multilevel paradigm. New matching algorithms, as well as refinement strategies, can be added with relative ease instead of being implemented from scratch. In addition, METIS uses efficient data structures and it is a mature library with years of optimization and improvement. DSHEM takes advantage of those data structures to emulate the directional communication and improve the quality of the partition. The obvious choice to implement the algorithm, and test its performance, is METIS; the comparison between SHEM and DSHEM is also simplified.

### 7.3.2. Graphs

Section 7.2 presents detailed information of the input graphs. Each graph has been partitioned with the options described in Table 7.5 and Table 7.11. The small 2D synthetic graphs have 30 to 40 vertices per

side and the 3D counterparts, 9 to 13. This variation in size produces 3D graphs with approximately the same amount of vertices than the 2D graphs. Table 7.6 presents the list of small synthetic graphs, their size, their edge percentage and the number of subdomains for the first set of experiments; a total of 240 graphs are part of this initial set.

**Table 7.6: Partitioning information for the first set of small synthetic graphs.**

| *Graph* | *Size* | *Percentage of edges* | *Subdomains* | *Objective* |
|---|---|---|---|---|
| sm_2d[_perc] | 30 to 40 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| tsm_2d[_perc] | 30 to 40 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| dtsm_2d[_perc] | 30 to 40 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| sm_3d[_perc] | 9 to 13 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| tsm_3d[_perc] | 9 to 13 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| dtsm_3d[_perc] | 9 to 13 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |

The second set of graphs is shown in Table 7.7; the graphs are slightly bigger than the previous set and it is composed of 150 graphs. This set is used to focus the analysis of DSHEM with more specific ranges of values for the parameters.

**Table 7.7: Partitioning information for the second set of small synthetic graphs.**

| *Graph* | *Size* | *Percentage of edges* | *Subdomains* | *Objective* |
|---|---|---|---|---|
| sm_2d[_perc] | 40 to 50, step 2 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| tsm_2d[_perc] | 40 to 50, step 2 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| dtsm_2d[_perc] | 40 to 50, step 2 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| sm_3d[_perc] | 11 to 14 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| tsm_3d[_perc] | 11 to 14 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |
| dtsm_3d[_perc] | 11 to 14 | 75, 90, 95, 98, 100 | 2 to 32 | cut, vol |

Table 7.8 lists the set of medium size synthetic graphs. They have around 1 million vertices and up to 5.5 million edges. This set is used to measure the scalability of DSHEM with larger input graphs.

**Table 7.8: Partitioning information for the set of medium size synthetic graphs.**

| *Graph* | *Size* | *Percentage of edges* | *Subdomains* | *Objective* |
|---|---|---|---|---|
| sm_2d[_perc] | 1000 | 95 | 64 | cut, vol |
| tsm_2d[_perc] | 1000 | 95 | 64 | cut, vol |
| dtsm_2d[_perc] | 710 | 95 | 64 | cut, vol |
| sm_3d[_perc] | 100 | 95 | 64 | cut, vol |
| tsm_3d[_perc] | 100 | 95 | 64 | cut, vol |
| dtsm_3d[_perc] | 80 | 95 | 64 | cut, vol |

The set of real graphs, in Table 7.9, includes 3 instances with different geometries and sizes. The

graphs are used to assess the performance of DSHEM with real world data. Due to the time limitations, some graphs have been partitioned with a limited amount of subdomains.

**Table 7.9: Partitioning information for the set of real life graphs.**

| *Graph* | *Vertices* | *Edges* | *Subdomains* | *Objective* |
|---|---|---|---|---|
| ef_4elt | 15,606 | 45,878 | 2 to 32 | cut, vol |
| ef_ocean | 143,437 | 409,593 | 2 to 32 | cut, vol |
| ef_sphere | 16,386 | 49,152 | 2 to 32 | cut, vol |

#### Justification

The first set of experiments is designed to limit the amount of processing power, storage and time that is required for the execution. Nevertheless, its purpose is also to give an overview of the performance of DSHEM with an extended range of values for the main parameters. With these extended experiments, it is possible to design a more accurate analysis for bigger instances of the input graphs and confirm, or refute, the initial results. The second set of small graphs is used to focus the study of DSHEM with more specific ranges of values used for the parameters. These new values are the product of the analysis of the experimental results from the first set. The set of medium size synthetic graphs is designed to measure the scalability of DSHEM with much bigger input graphs. The number of subdomains used during experimentation is reduced due to time constraints. Finally, DSHEM is evaluated with a set of real world graphs, from small to medium sizes and different geometries. The four sets of graphs provide accurate information on the performance of DSHEM.

### 7.3.3. Hardware Setup

The Scientific Compute Cluster located at GWDG offers a comprehensive range of resources. It uses the Platform Load Sharing Facility (Platform LSF) as job scheduler to distribute the jobs into several queues. Table 7.10 details the hardware resources available for the general purpose queue *mpi*. This queue is suited for serial and parallel jobs. The list includes, as well, the *fat* queue for jobs with high memory demand; nodes with at least 256 GB of memory. Once a job is submitted to the queue, Platform LSF will start the job according to its requirements and the available resources.

**Table 7.10: Hardware overview of the HPC cluster at GWDG.**

| *Nodes* | *CPU* | *Cores* | *Frequency* | *Memory* | *Interconnect* |
|---|---|---|---|---|---|
| 168 | Ivy-Bridge Intel E5-2670 v2 | 2×10 | 2.5 GHz | 64 GB | InfiniBand<br>Quad data rate |
| 160 | Sandy-Bridge Intel E5-2670 | 2×8 | 2.6 GHz | 64 GB | InfiniBand<br>Quad data rate |
| 76 | Broadwell Intel E5-2650 v4 | 2×12 | 2.2 GHz | 128 GB | InfiniBand<br>Fourteen data rate |

| | | | | | |
|---|---|---|---|---|---|
| 48 | Abu-Dhabi AMD Opteron 6378 | 4×16 | 2.4 GHz | 256 GB | InfiniBand Quad data rate |
| 15 | Broadwell Intel E5-2650 v4 | 2×12 | 2.2 GHz | 512 GB | InfiniBand Fourteen data rate |
| 5 | Haswell Intel E5-4620 v3 | 4×10 | 2.0 GHz | 1.5 TB | InfiniBand Fourteen data rate |
| 1 | Haswell Intel E7-4809 v3 | 4×8 | 2.0 GHz | 2 TB | InfiniBand Quad data rate |

The set of jobs submitted to the *mpi* queue during experimentation are all serial. They were assigned to different nodes according to the node availability at the moment of execution; some of them were executed in nodes belonging to the *fat* queue.

### 7.3.4. DSHEM Parameters

In order to optimize the partition, a set of options are available; they are all related to DSHEM and its nested structure. Table 7.11 shows the list of possible parameters that can be used to tune up the execution.

**Table 7.11: List of DSHEM execution options for METIS.**

| *Option* | *Description and possible values* | | *Values used* |
|---|---|---|---|
| **-dshem_p1** | First percentage for the utility function that matches vertices with DSHEM. Only available for DSHEM. | | Varies for each set of experiments. |
| | ***Numeric value*** | Percentage for the weight of the first part of the utility function | |
| **-dshem_p2** | Second percentage for the utility function that matches vertices with DSHEM. Only available for DSHEM. | | Varies for each set of experiments. |
| | ***Numeric value*** | Percentage for the weight of the second part of the utility function | |
| **-dshem_p3** | Third percentage for the utility function that matches vertices with DSHEM. Only available for DSHEM. | | Varies for each set of experiments. |
| | ***Numeric value*** | Percentage for the weight of the third part of the utility function | |
| **-nctype** | Nested coarsening process. It uses independent SHEM/DSHEM values for the coarsening process and for the initial partition. Only available for DSHEM. | | dshem+dshem<br>dshem+shem<br>shem+dshem<br>shem+shem |
| | **dshem+dshem** | DSHEM for coarsening and DSHEM for partitioning | |
| | **dshem+shem** | DSHEM for coarsening and SHEM for partitioning | |
| | **shem+dshem** | SHEM for coarsening and | |

| | | | |
|---|---|---|---|
| | | DSHEM for partitioning | |
| | **shem+shem** | SHEM for coarsening and<br>SHEM for partitioning | |
| **-maxvwtm** | Maximum vertex weight multiplier. During the coarsening process the vertices are matched. One of the conditions limits the size of the coarsest vertex. It is originally hardcoded in METIS. | | Varies for each set of experiments. |
| | ***Numeric value*** | Percentage of the maximum weight of a vertex | |

These options are received by *gpmetis* and used exclusively for the execution of DSHEM. The values for the experiments depend on the specific set of experiments; details are available in Chapter 8 and Chapter 9 where the individual experiments and their results are presented.

## 7.4. Organization of the Evaluation

The analysis is designed to evaluate different aspects of the execution and behavior of DSHEM to effectively assess its performance. With a wide range of execution parameters for METIS and DSHEM, it is necessary to understand their impact on the final partition. With that in mind, the experimental evaluation is divided in 5 independent major parts:

- The effect of the multiplier *-maxvwtm* on the quality of partitions
- The effect of percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* on the quality of partitions with different types of graphs
- The impact of different degrees of irregularity on the quality of partitions
- The impact of the refinement process on DSHEM and subsequent quality of partitions
- The impact of DSHEM on the execution time (partitioning time) of METIS

To understand the comportment of DSHEM under different conditions it is necessary to evaluate the partitions produced with a wide range of values for the execution parameters. The range of values utilized for the different parameters depend on the individual experiment. They are initially selected to provide a general overview of the performance of DSHEM. Subsequent sets of experiments are based on the results of the previous sets; they provide a more detailed view of DSHEM.

With this evaluation method, it is possible to identify which set of parameters improve the quality of the partition for a certain type of graph. This is of vital importance as performance of any graph partitioning library can be drastically impacted by the incorrect parameters.

# Chapter 8.

# Experimental Analysis of DSHEM

This chapter presents the initial evaluation of DSHEM with four different sets of experiments as described in Chapter 7. First, DSHEM is executed with small graphs and a wide range of values for the parameters *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. Based on the previous results, a new set of experiments is performed with more precise parameters to tune up the algorithm. The third set of experiments is designed to evaluate the scalability of DSHEM with larger graphs. The final set of experiments, with real life graphs, helps confirm the evaluation results. The next sections present a detailed evaluation of the full implementation of DSHEM.

## 8.1. Full DSHEM Partitioning

The first evaluation of DSHEM is based on the original implementation of the strategy as described in Appendix A and Section 6.3.2, Full DSHEM Partitioning; hence, it considers the main idea behind DSHEM. The fundamental change is the condition that determines if the adjacent vertex $j$ is matched to vertex $i$. This implementation is designed to neither increases the memory requirements nor execution time and takes full advantage of the data structures without performance degradation.

## 8.2. First Experiments on Small Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.6 of Chapter 7. It is a set of small synthetic graphs which are designed to evaluate the performance of DSHEM and compare it with SHEM and Random.

### 8.2.1. Execution Parameters

Four main parameters are used to tune up DSHEM, namely *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. The values chosen for the first set of experiments are presented in Table 8.1. This particular set produces 1715 different combinations of values, giving a wide view of the performance of DSHEM. The subsequent experiments are designed based on the results of these initial experiments.

**Table 8.1: DSHEM parameters for the first set of small synthetic graphs.**

| *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|
| 140 to 160, step 5 | 91 to 109, step 3 | 91 to 109, step 3 | 91 to 109, step 3 |

## 8.2.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. First, the effect of the multiplier *-maxvwtm* is evaluated. Next, the three percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are examined to understand their influence. The robustness of DSHEM is also evaluated with different degrees of irregularity introduced to the synthetic graphs. The refinement and its influence on DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

### Multiplier *-maxvwtm*

The multiplier *-maxvwtm* limits the size of vertices during the coarsening process. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a low value may have also undesired effects such as the inability to match vertices that could lead to an infinite loop trying to contract the graph without success.

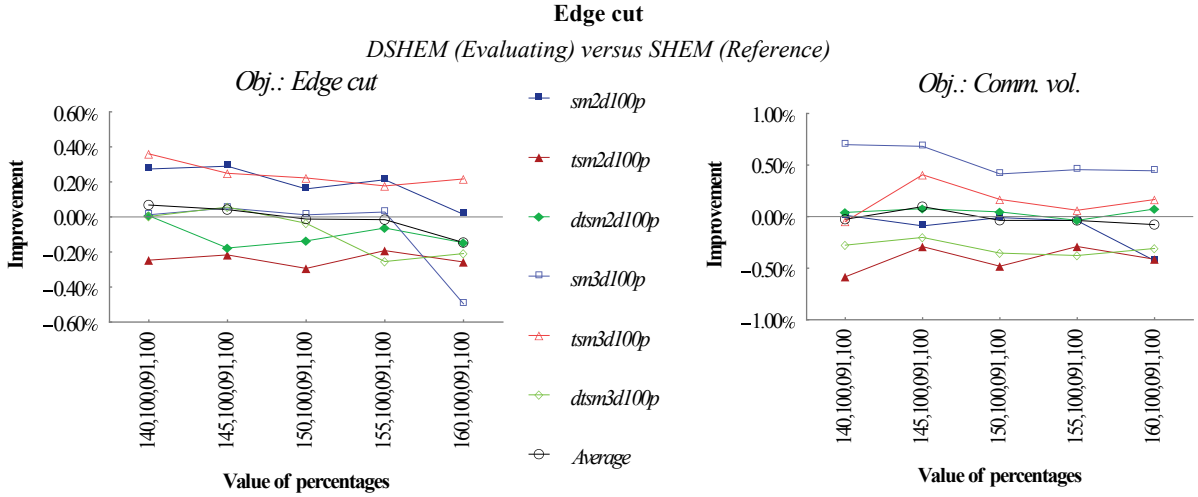


**Figure 8.1. DSHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

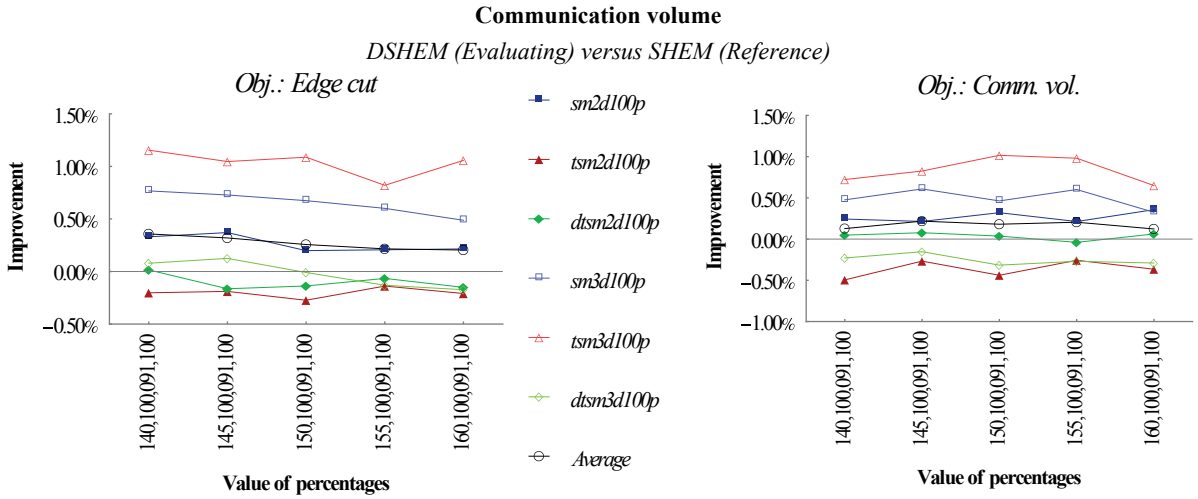


**Figure 8.2. DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

From Figure 8.1, Figure 8.2 and Figure 8.3, it is possible to deduce a pattern on the role of the multiplier *-maxvwtm*; it is more common to obtain better results by reducing its value. It is also evident that the type of graph has a great influence too, being the 3D square graph (*sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) with the highest improvements, and the 2D triangular square graph (*tsm2d100p*) with the worst results.

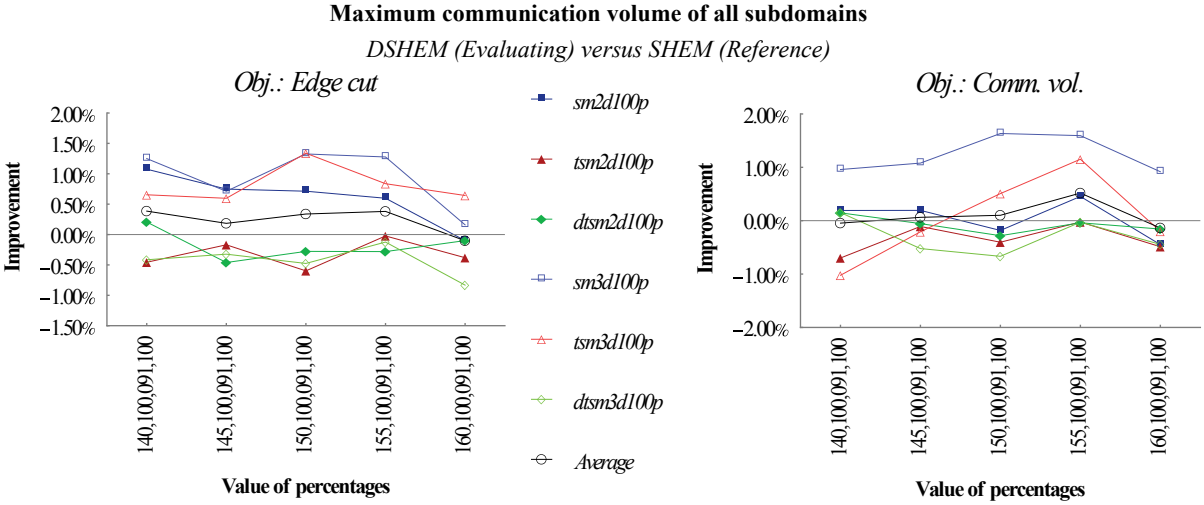


**Figure 8.3. DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are used to modify the behavior of the cost function in DSHEM. It may improve the results by selecting the right values according to the type of graph to partition.

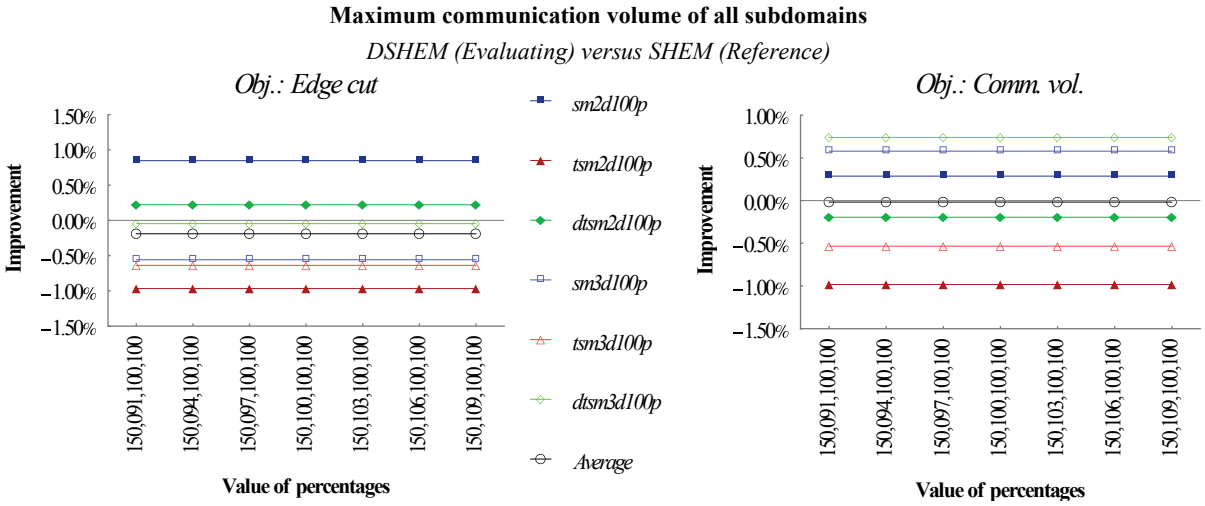


**Figure 8.4. DSHEM vs. SHEM: effect of *-dshem_p1* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

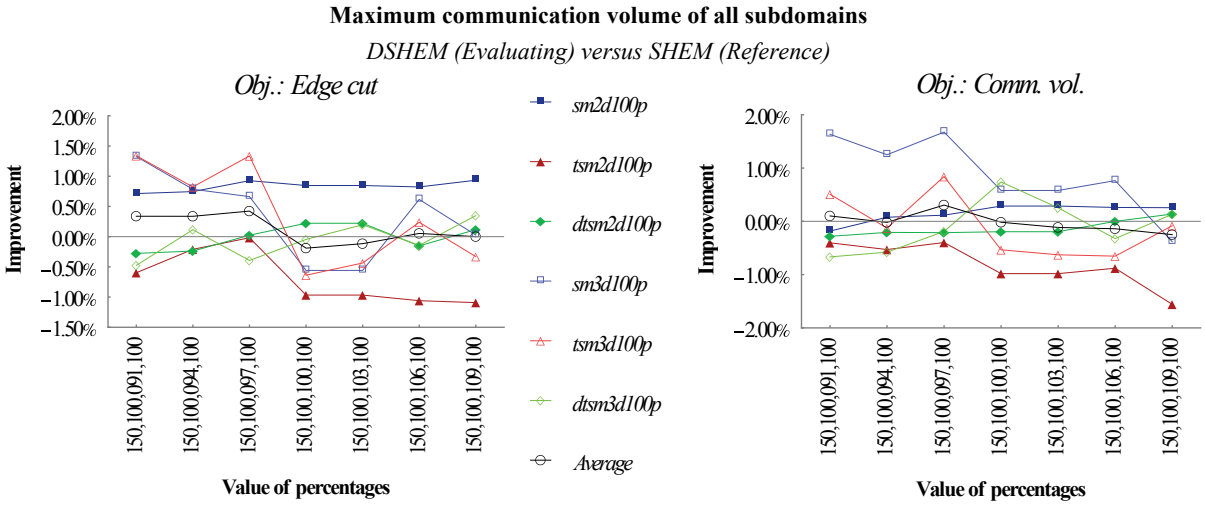


**Figure 8.5. DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

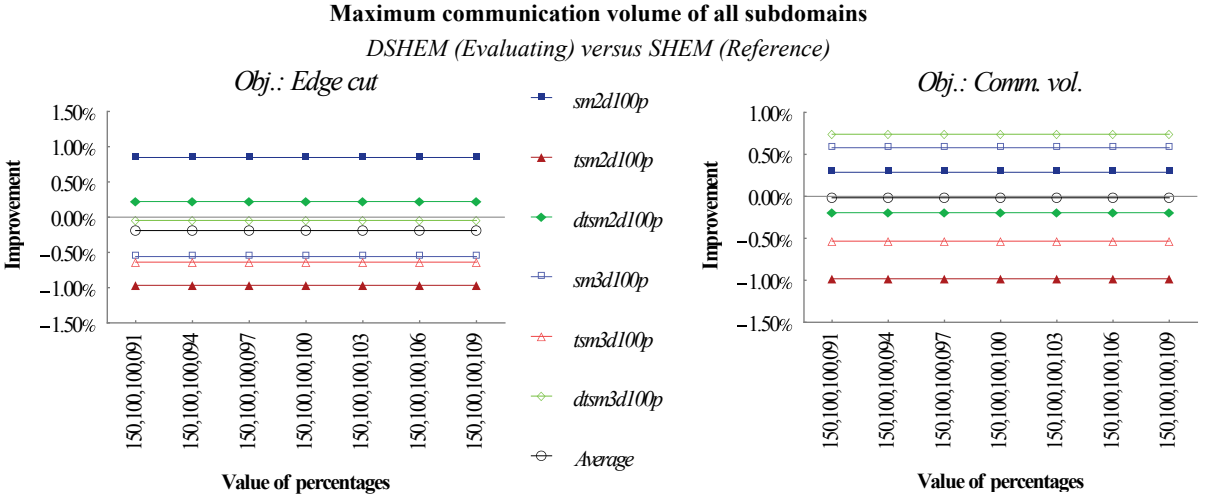


**Figure 8.6. DSHEM vs. SHEM: effect of *-dshem_p3* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Percentage *-dshem_p1* seems to have no influence based on the results from Figure 8.4. This

comportment is present for all metrics and all graphs. Percentage *-dshem_p3* has a similar output as shown in Figure 8.6. However, Figure 8.5 shows that percentage *-dshem_p2* changes the quality of the partitions generated by DSHEM; some types of graphs present a bigger impact by this percentage. Based on the results, it is possible to conclude that lower values for *-dshem_p2* produce better results.

After a deeper analysis of the execution of DSHEM, it was found that the conditional, which includes the three percentages, only evaluates to *TRUE* for *-dshem_p1* and *-dshem_p3* the first time it is executed. The rest of the execution, only *-dshem_p2* may evaluate to *TRUE* according to the conditions of the current matching vertices.

## Graph Irregularity

The performance of DSHEM is also studied with irregular graphs. Sizes $a$ to $e$, in Figure 7.2, Figure 7.3 and Figure 7.4, represent the five biggest 2D and all 3D graphs of the set.

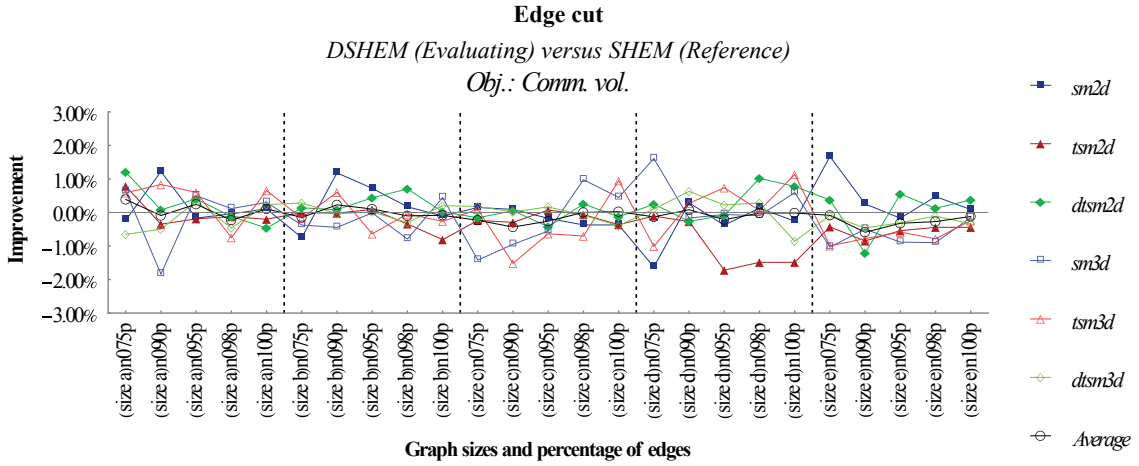


**Figure 8.7. DSHEM vs. SHEM: effect of irregularity on the edge cut with synthetic graphs and communication volume as partitioning objective.**

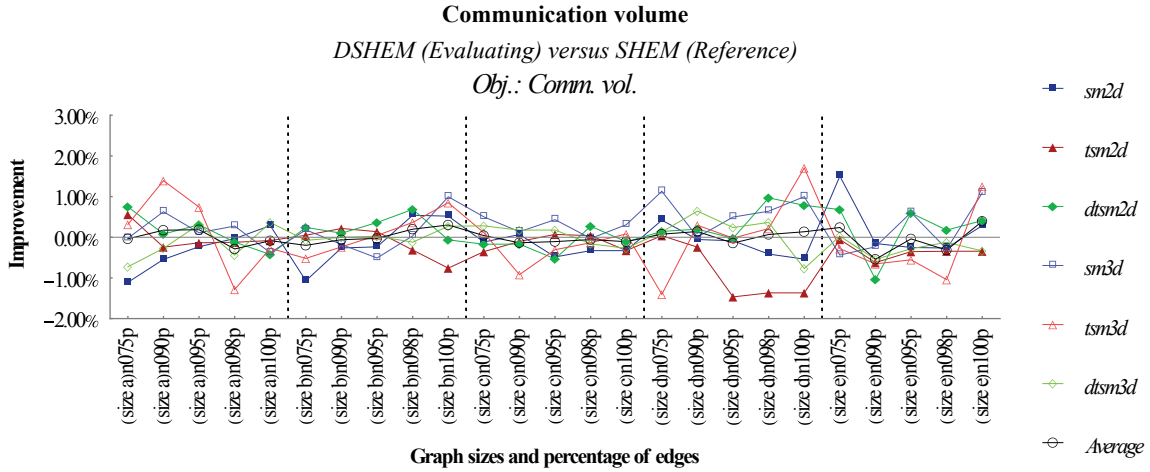


**Figure 8.8. DSHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

From the results, it is possible to see that some types of graphs have greater impact than others. One of the most stable types is the 2D dense triangular square graph (*dtsm2d*) when evaluating the edge cut. While others may show some spikes, there is not a clear pattern. The introduced regularity in the graphs

has an impact, but the percentage of irregularity is not necessarily proportional to the impact on the quality of the partition.

Regarding the communication volume and maximum communication volume of all subdomains, similar behavior can be seen. The irregularity impacts the final partition, but it does not degrade or improves it with a clear pattern.

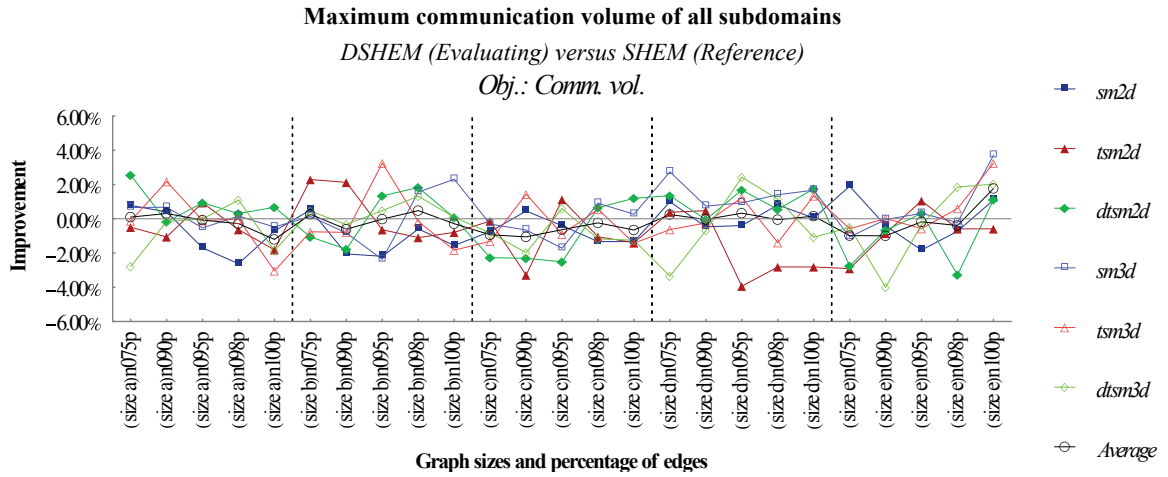


**Figure 8.9. DSHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

## Refinement

METIS is executed without refinement for both, SHEM and DSHEM, to analyze the effect on the partitioning process. Whether the partitioning objective is the edge cut or the communication volume, the results remain the same; SHEM and DSHEM perform the matching without a partitioning objective and without the refinement process no objective is optimized.

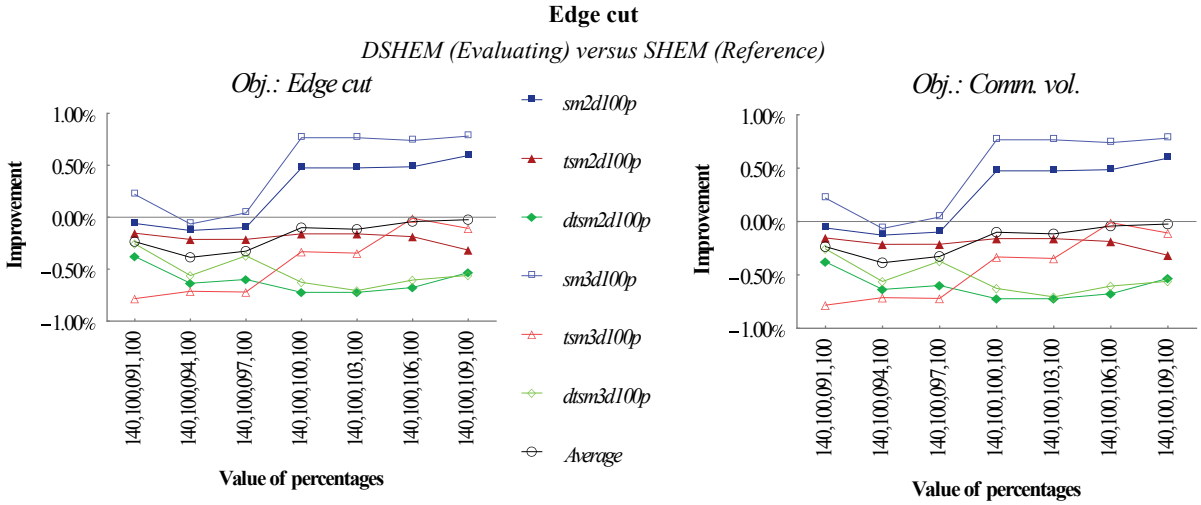


**Figure 8.10. DSHEM vs. SHEM: effect of refinement on the edge cut with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Figure 8.10 shows the effect of the refinement when the edge cut is evaluated. It is clear that the 2D and 3D square graphs (*sm2d100p* and *sm3d100p*) benefit from *-dshem_p2* with values from to 100. The 3D triangular square graph (*tsm3d100p*) has a similar pattern, however in the negative side of the plot. The rest of the graphs have a more stable behavior.

Communication volume is a different story. Figure 8.11 shows that the 2D and 3D square graphs (*sm2d100p* and *sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) have a clear benefit from DSHEM; it is stable for all values of *-dshem_p2*.

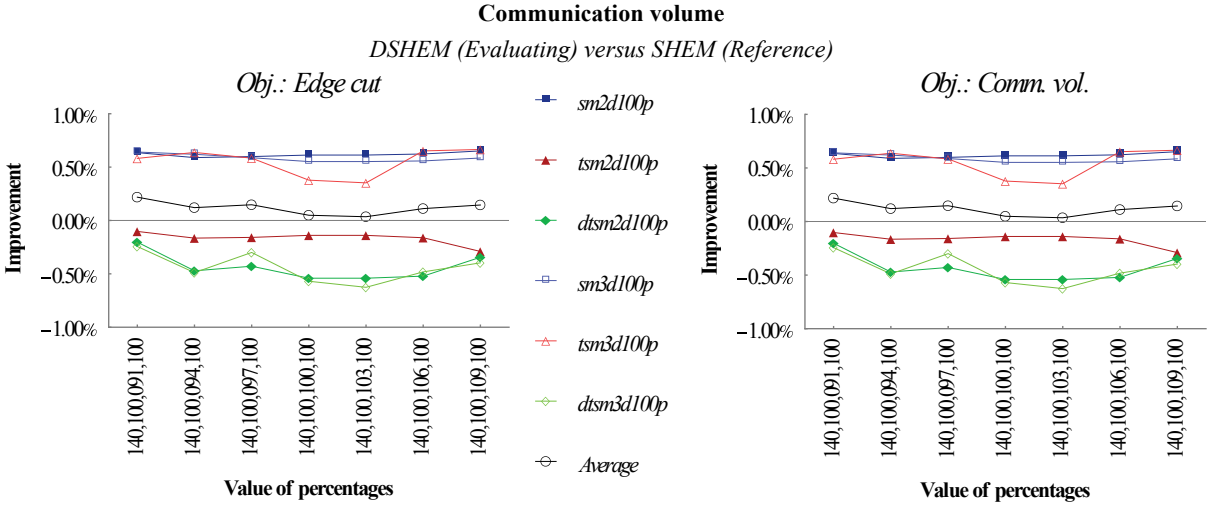


**Figure 8.11. DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

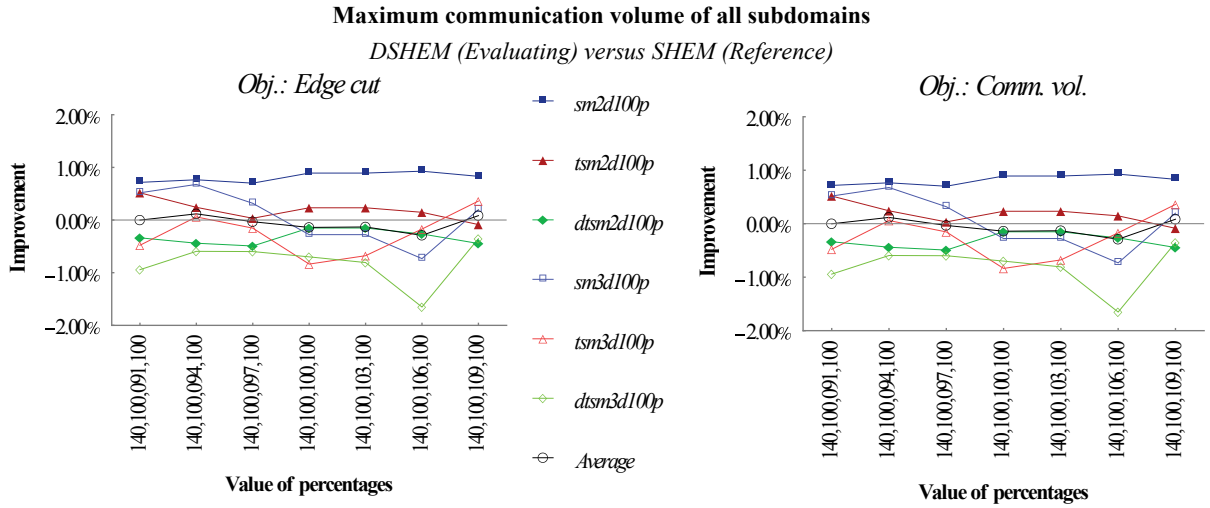


**Figure 8.12. DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Regarding the maximum communication volume of all subdomains, Figure 8.12 shows that the 2D square graph (*sm2d100p*) keeps an almost constant positive behavior. The 3D counterpart performs better with *-dshem_p2* values lower than 100.

### Execution Time

The graphs for these experiments are small enough to make virtually impossible to evaluate the impact of DSHEM on the execution time. The majority of the running times does not even reach one second.

## 8.3. Second Experiments on Small Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.7 of Chapter 7. It is a set of small

synthetic graphs which are designed to evaluate the performance of DSHEM and compare it with SHEM and Random.

### 8.3.1. Execution Parameters

Two main parameters are used to tune up DSHEM for this set of experiments, namely *-maxvwtm* and *-dshem_p2*. The other two parameters have a fix value of 100. The values chosen for the second set of experiments are presented in Table 8.2. This particular set produces 441 different combinations of values, giving a more focused view of the performance of DSHEM. This set of experiments is designed based on the results of the first set.

**Table 8.2: DSHEM parameters for the second set of small synthetic graphs.**

| *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|
| 140 to 160 | 100 | 90 to 110 | 100 |

### 8.3.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. Based on the results of the first set of experiments, the effect of the multiplier *-maxvwtm* is evaluated with greater detail, as well as the percentage *-dshem_p2*. Percentages *-dshem_p1* and *-dshem_p3* are set to 100 as they do not influence the outcome.

Again, the robustness of DSHEM is also evaluated with different degrees of irregularity introduced to the synthetic graphs. The refinement and its influence on DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

#### Multiplier *-maxvwtm*

A closer analysis of the multiplier *-maxvwtm* brings a different scenario of that from the first set of experiments. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a balanced partition does not necessarily mean a smaller edge cut or reduction in communication volume; it only means that the subdomains are more equal in size.

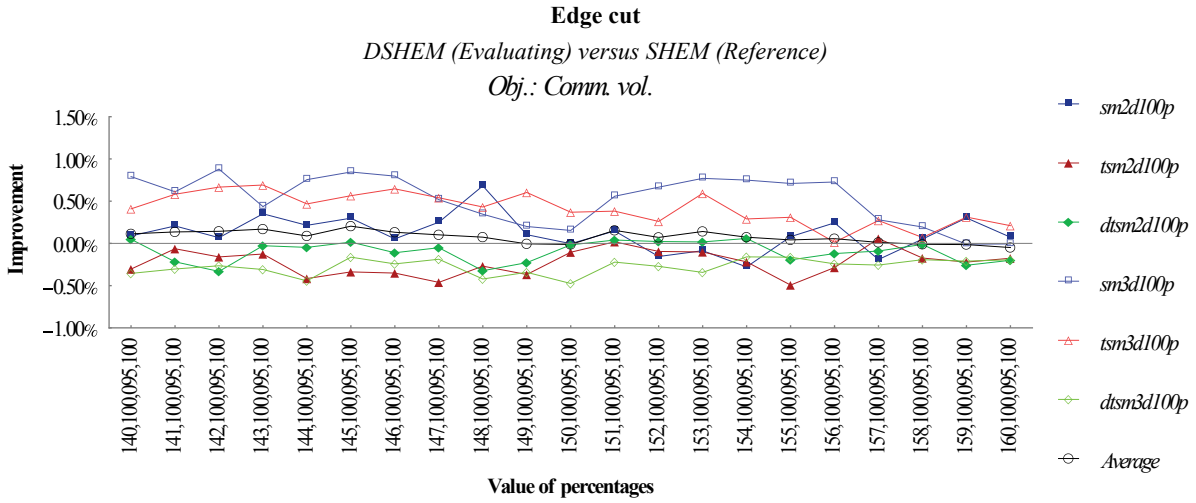


**Figure 8.13. DSHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with synthetic graphs and communication volume as partitioning objective.**

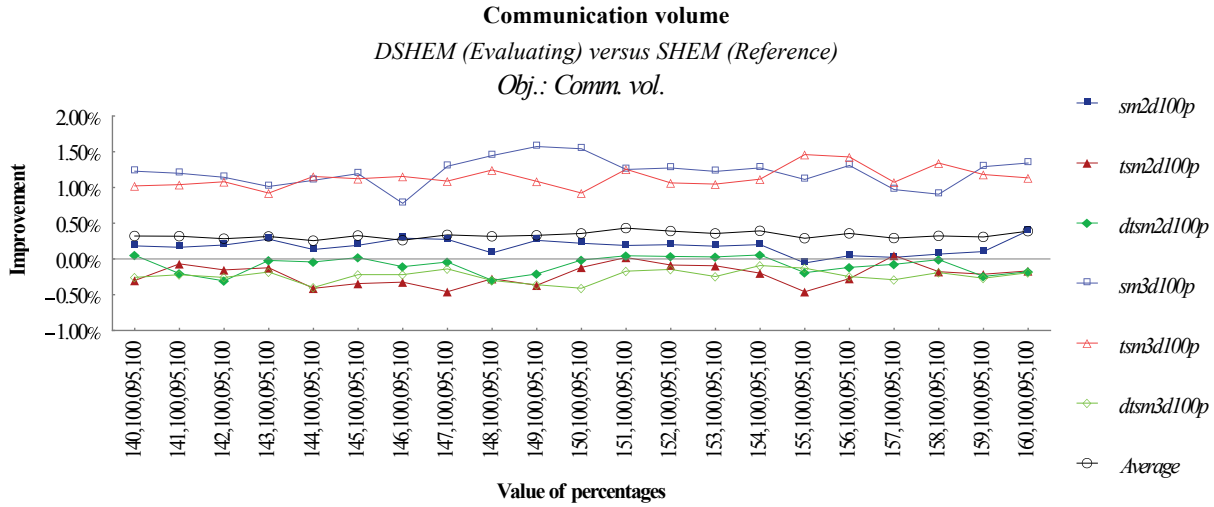


**Figure 8.14. DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

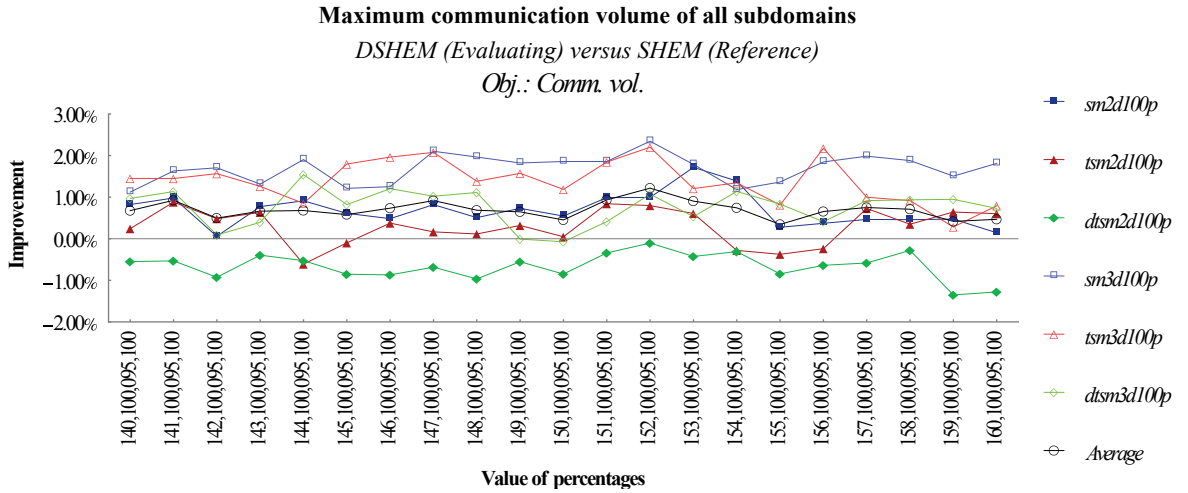


**Figure 8.15. DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

From the results in Figure 8.13, Figure 8.14 and Figure 8.15, it is not possible to find a discernible pattern as suggested from the first set of experiments. It is evident that the multiplier *-maxvwtm* plays a role, but only after trial and error it would be possible to adapt it to specific needs.

It is also confirmed that the type of graph has an influence, being the 3D square graph (*sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) with the highest improvements, and the 2D triangular square graph (*tsm2d100p*) and the 3D dense triangular square graph (*dtsm3d100p*) with the worst results.

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from a deeper analysis as previous results suggest they do not play a role at all in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in DSHEM and improve the partition.

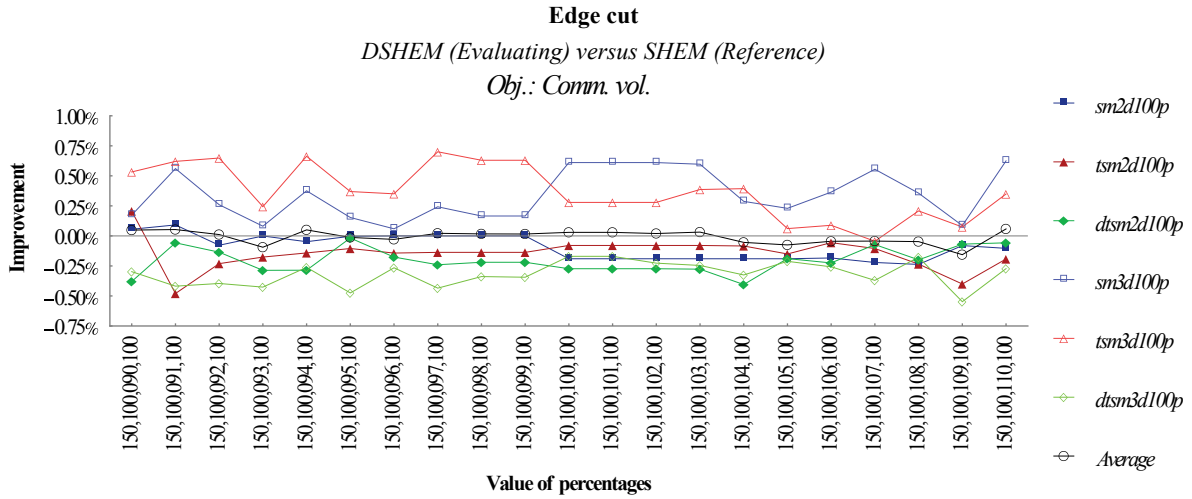


**Figure 8.16. DSHEM vs. SHEM: effect of *-dshem_p2* on the edge cut with synthetic graphs and communication volume as partitioning objective.**

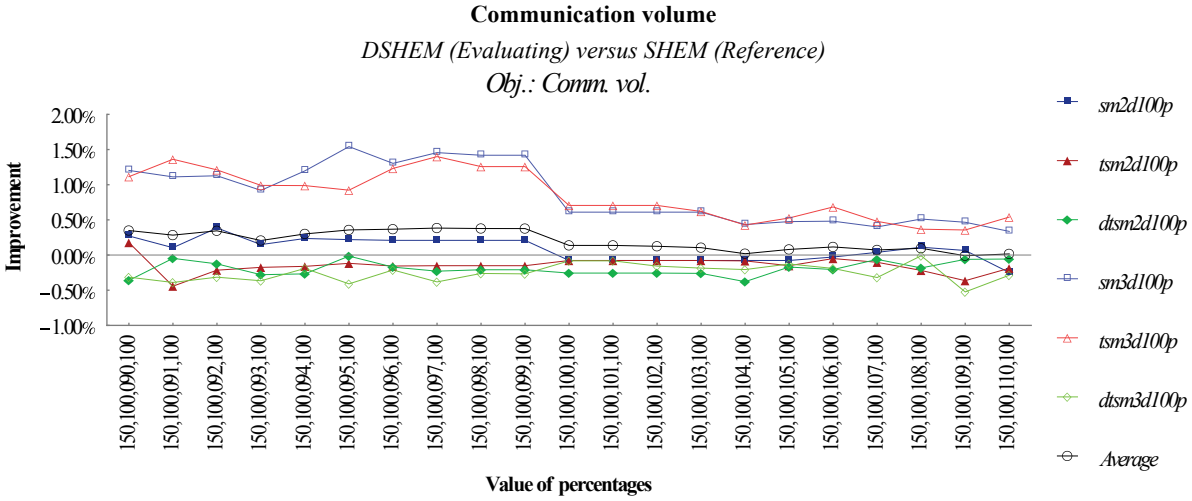


**Figure 8.17. DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

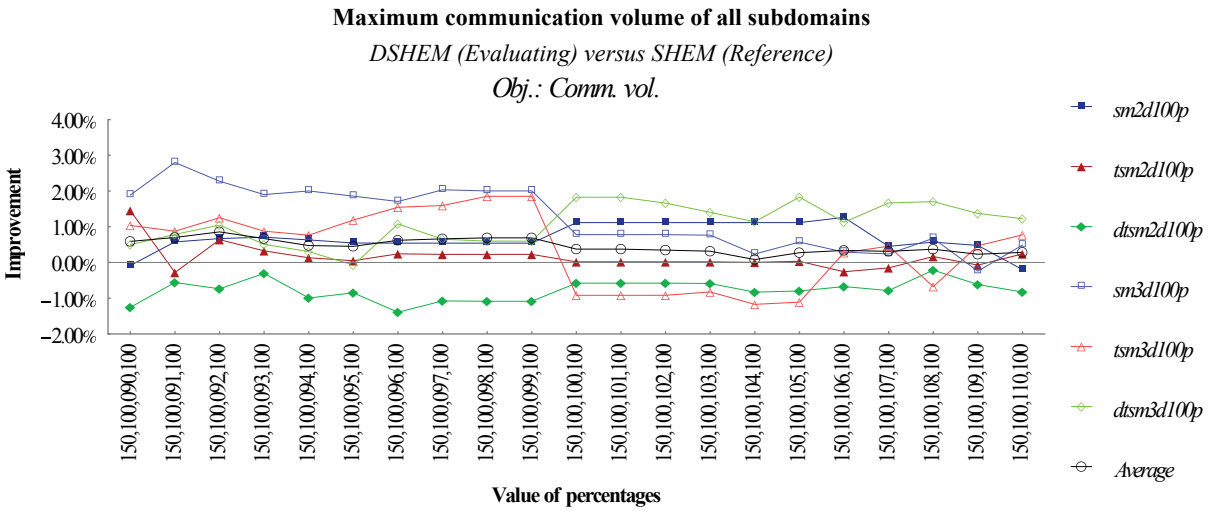


**Figure 8.18. DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Depending on the metric and partitioning objective, percentage *-dshem_p2* has different effects on the output of the partitioning process. Figure 8.16 shows that the influence is more erratic for the edge cut; being the 3D square graph (*sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) the most affected. With communication volume in mind, the same types of graphs benefit from the percentage *-dshem_p2* being lower than 100; see Figure 8.17. Finally, when the maximum communication volume of all subdomains is evaluated, Figure 8.18 shows that values over 100 benefit some graph types.

## Graph Irregularity

The performance of DSHEM is also studied with irregular graphs. Sizes $a$ to $d$, in Figure 8.19, Figure 8.20 and Figure 8.21, represent the four biggest 2D and all 3D graphs of the set.

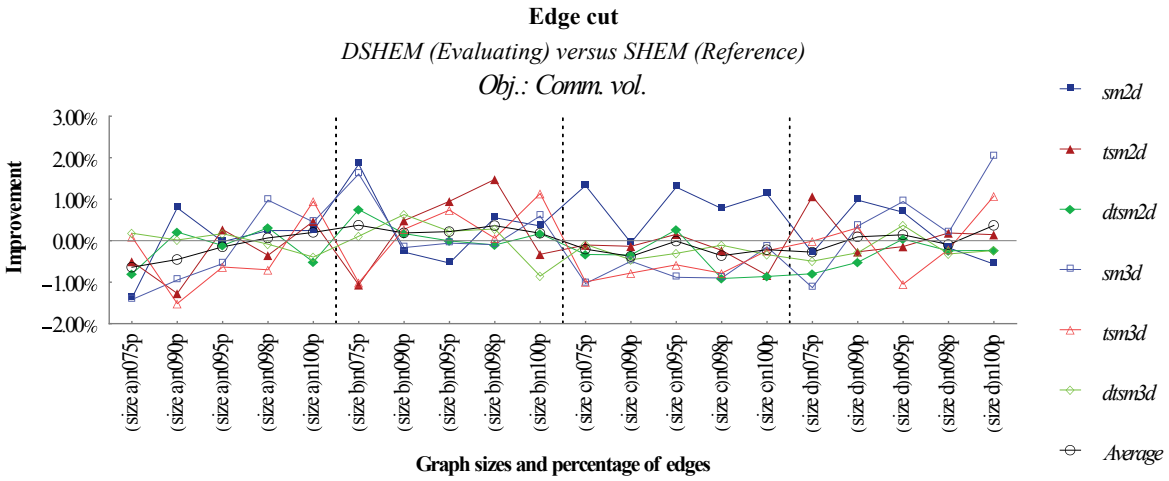


**Figure 8.19. DSHEM vs. SHEM: effect of irregularity on the edge cut with synthetic graphs and communication volume as partitioning objective.**

From the results of the second set of experiments, it is possible to conclude that the results are consistent with the first set or experiments. One of the most stable types is the 2D dense triangular square graph (*dtsm2d*) when evaluating the edge cut. The introduced regularity in the graphs clearly has an impact, but the amount of irregularity does not reflect a proportional impact on the quality of the

partition.

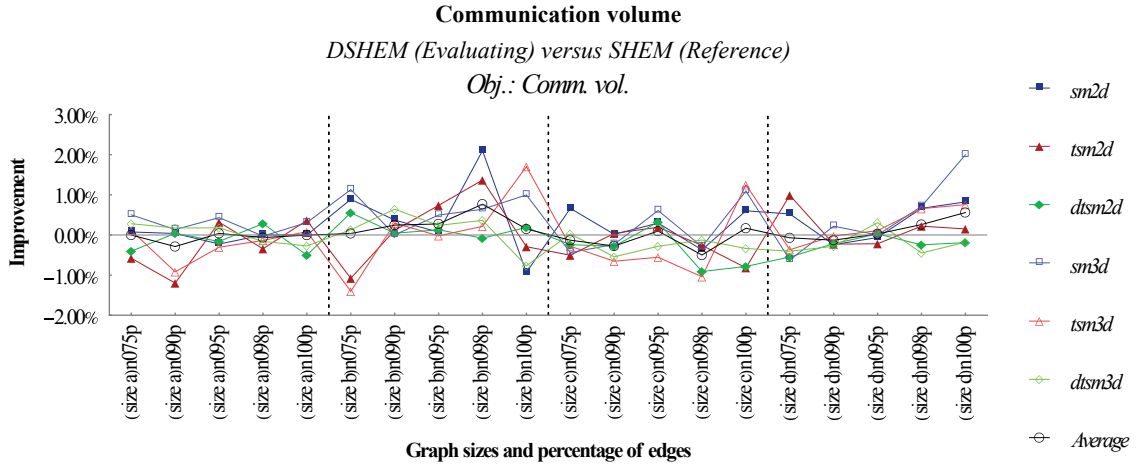


**Figure 8.20. DSHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

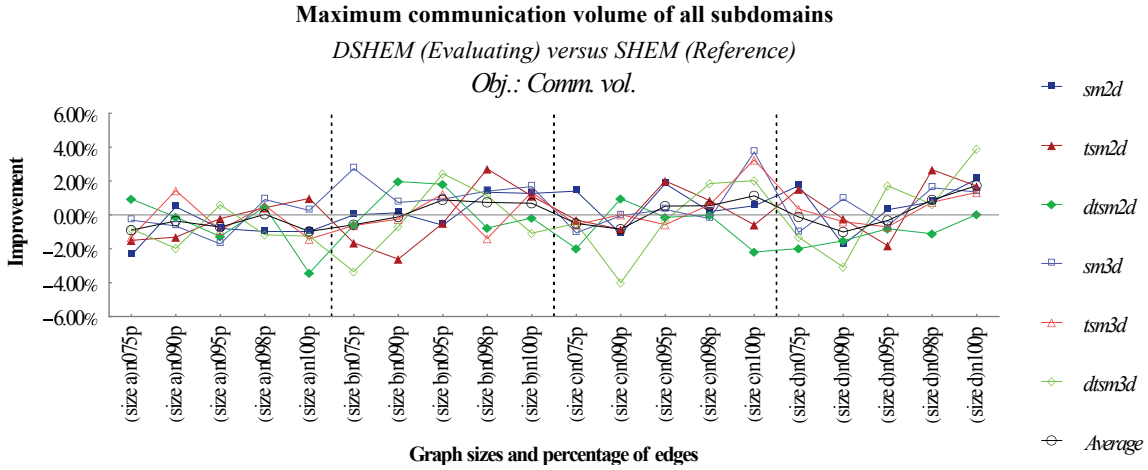


**Figure 8.21. DSHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

With respect to the communication volume and maximum communication volume of all subdomains, similar behavior can be seen. The irregularity impacts the final partition, but it does not degrade or improves it with a clear pattern.

## Refinement

To analyze the effect of the refinement process on the partitions produced by DSHEM, METIS is executed without it; without the refinement process no objective is optimized.

Figure 8.22 shows similar results as those from the first set of experiments: the 2D and 3D square graphs (*sm2d100p* and *sm3d100p*) benefit from *-dshem_p2* with values from 100 and higher. However, the 3D triangular square graph (*tsm3d100p*) presents a less visible pattern as that from the first set. The 3D dense triangular square graph (*dtsm3d100p*) is also affected.

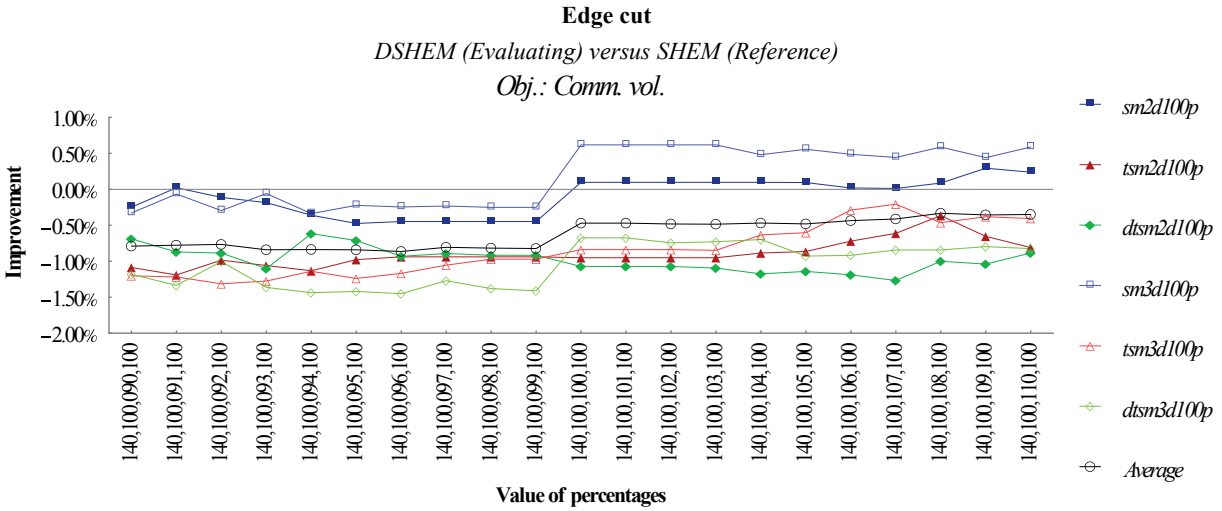


**Figure 8.22. DSHEM vs. SHEM: effect of refinement on the edge cut with synthetic graphs and communication volume as partitioning objective.**

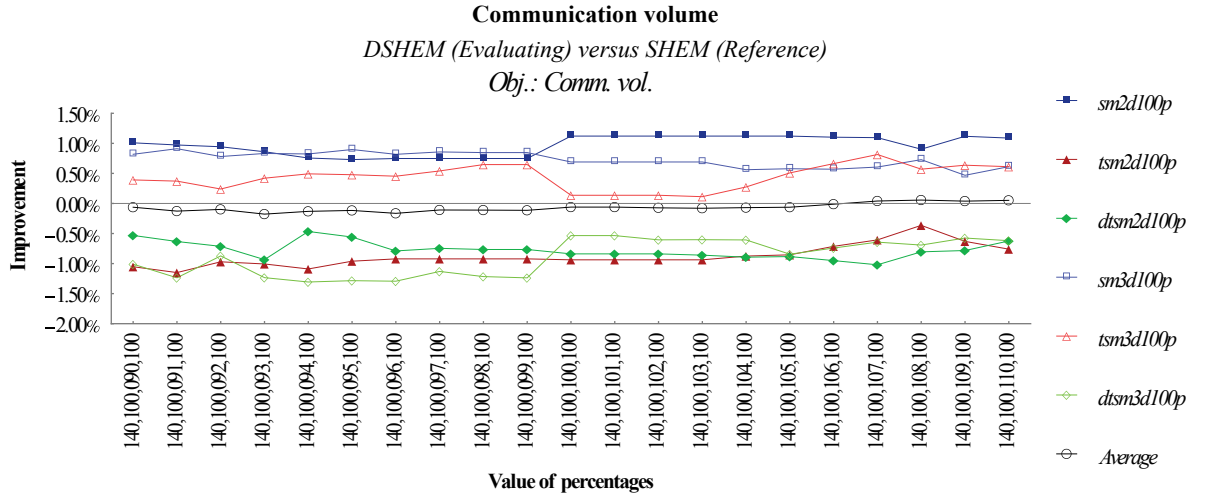


**Figure 8.23. DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

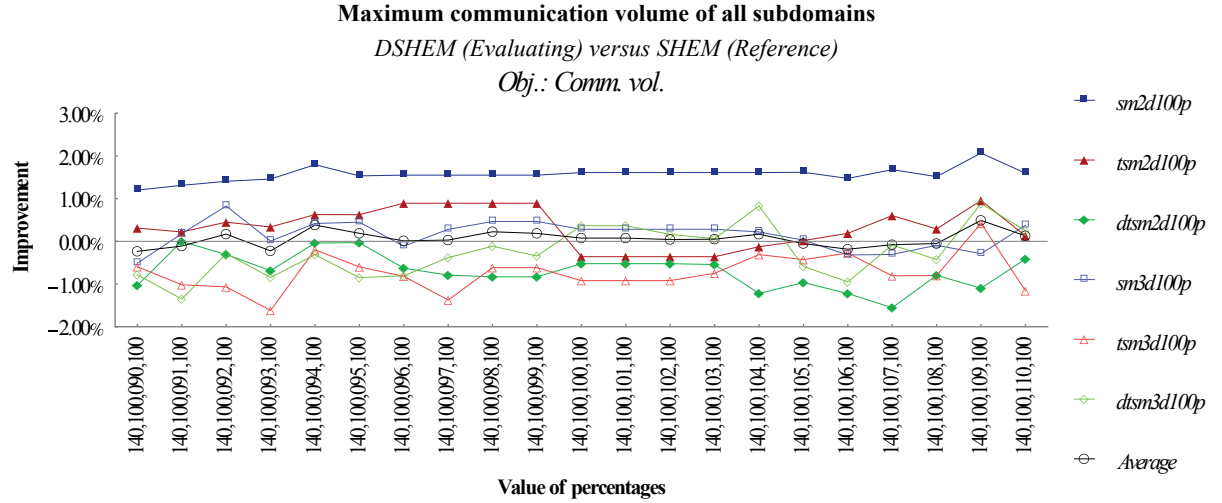


**Figure 8.24. DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Communication volume remains quite stable. Similar as the first set of experiments, Figure 8.23

shows that the 2D and 3D square graphs (*sm2d100p* and *sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) benefit from DSHEM. The percentage *-dshem_p2* has an influence too, with an inflection point in 100.

With respect to the maximum communication volume of all subdomains, Figure 8.24 shows that the 2D square graph (*sm2d100p*) still keeps an almost constant positive benefit. The 2D triangular square graph (*tsm2d100p*) performs better with *-dshem_p2* values lower than 100.

#### Execution Time

The graphs for these experiments are small enough to make virtually impossible to evaluate the impact of DSHEM on the execution time. The majority of the running times does not even reach one second.

## 8.4. Experiments on Medium Size Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.8 of Chapter 7. It is a set of medium size synthetic graphs which are designed to evaluate the performance of DSHEM with larger graphs and compare it with SHEM and Random.

### 8.4.1. Execution Parameters

Two main parameters are used to tune up DSHEM for this set of experiments, namely *-maxvwtm* and *-dshem_p2*. The other two parameters have a fix value of 100. The values chosen for the third set of experiments are presented in Table 8.3. This particular set produces 441 different combinations of values, giving a more focused view of the performance of DSHEM. This set of experiments is designed based on the results of the first and second sets.

**Table 8.3: DSHEM parameters for the set of medium size synthetic graphs.**

| *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|
| 140 to 160 | 100 | 90 to 110 | 100 |

### 8.4.2. Analysis of Results

Based on the results of the first two sets of experiments, the effect of the multiplier *-maxvwtm* is evaluated with detail, as well as the percentage *-dshem_p2*. Percentages *-dshem_p1* and *-dshem_p3* are set to a fixed value of 100 as they do not influence the outcome. The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions.

DSHEM is evaluated with a degree of irregularity introduced to the synthetic graphs. The refinement and its influence on DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the

edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

### Multiplier *-maxvwtm*

The analysis of the multiplier *-maxvwtm* follows that of the second set of experiments; a detailed view of its impact. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a balanced partition does not necessarily mean a smaller edge cut or reduction in communication volume; it only means that the subdomains are more equal in size.

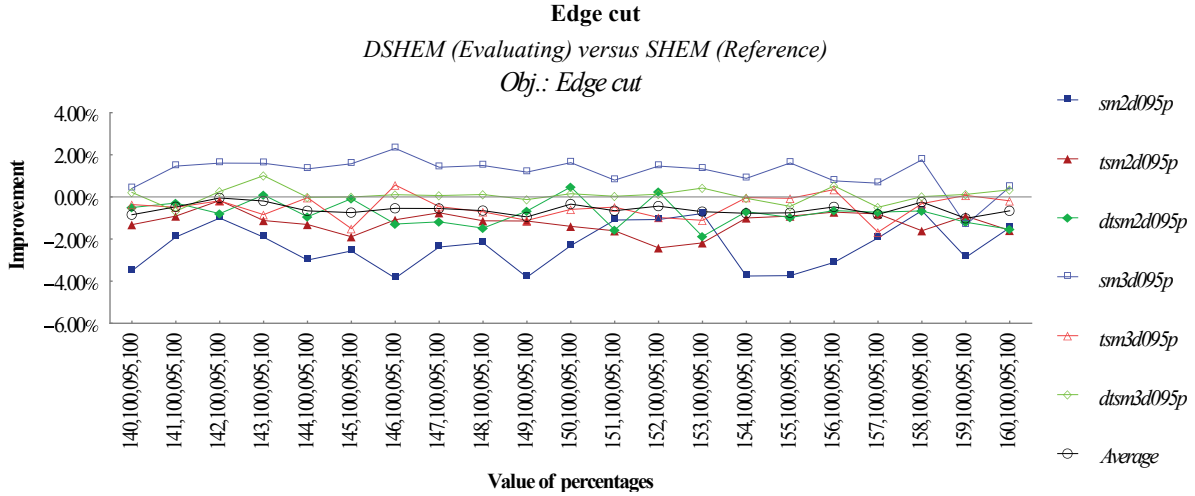


**Figure 8.25. DSHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

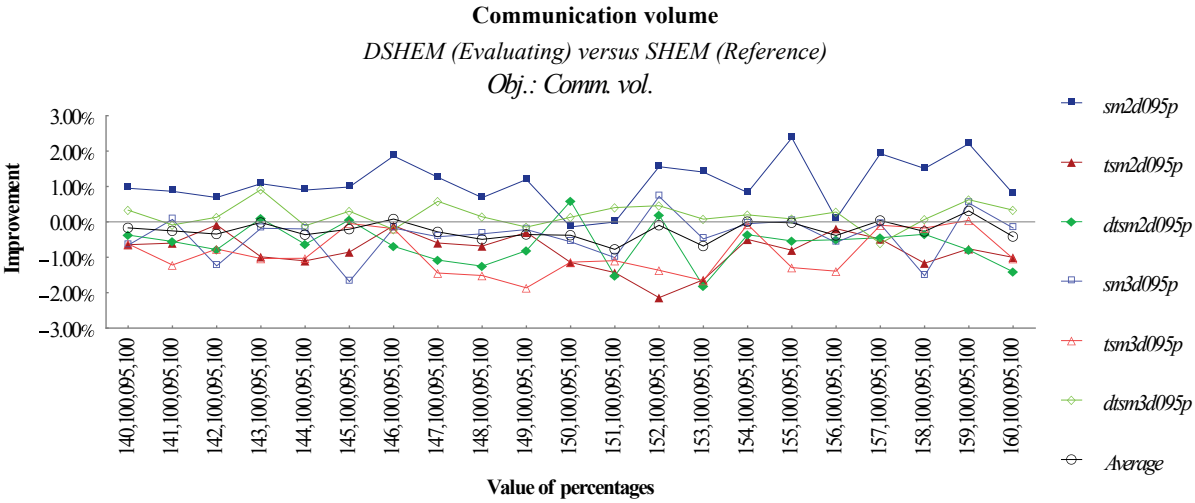


**Figure 8.26. DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

From the results in Figure 8.25, it is evident that the 3D square graph (*sm3d095p*) benefits from DSHEM when the edge cut is evaluated. The same pattern was found with the first and second sets of experiments. However, the 2D square graph (*sm2d095p*) shows significant degradation, contrary to the first two sets; this could be due to the fact that the graphs in this set of experiments have a degree or irregularity introduced. Regarding the total communication volume, the 2D square graph (*sm2d095p*)

presents the best results, see Figure 8.26. It is consistent with the findings of previous experiments. The 3D triangular square graph (*tsm3d100p*), contrary to the previous sets of experiments, presents degradation in the partition quality that could be attributed to the degree or irregularity introduced to the graph. There is not a clear pattern when the maximum communication volume of all subdomains is evaluated, as seen in Figure 8.27.

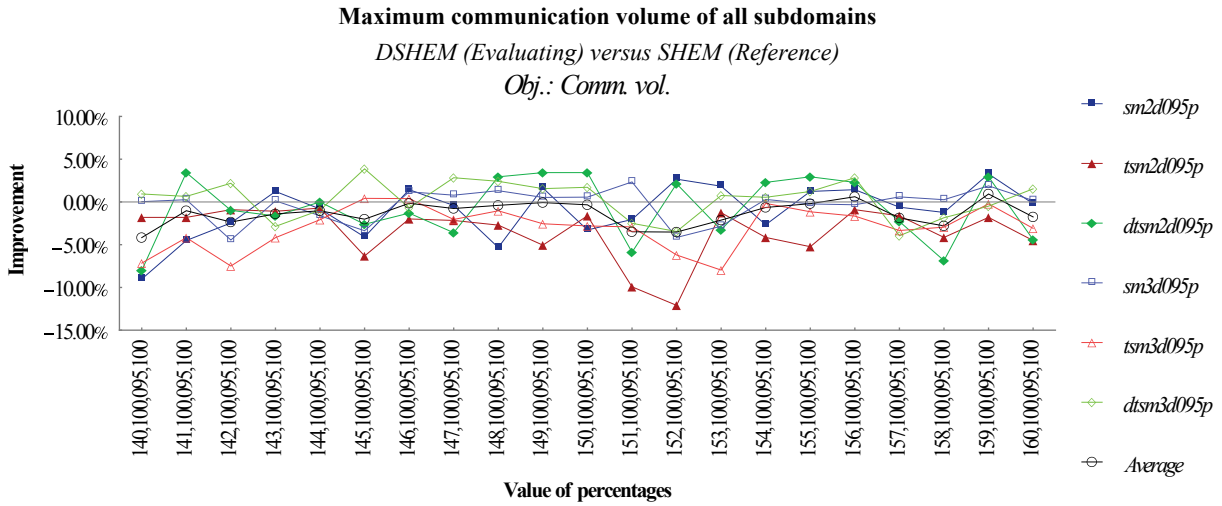


**Figure 8.27. DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

In summary, multiplier *-maxvwtm* does not provide a clear advantage or disadvantage by modifying its value during the partitioning process. It has a clear effect, but it is not predictable.

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from a deeper analysis as previous results suggest they do not play a role at all in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in DSHEM and improve the partition.

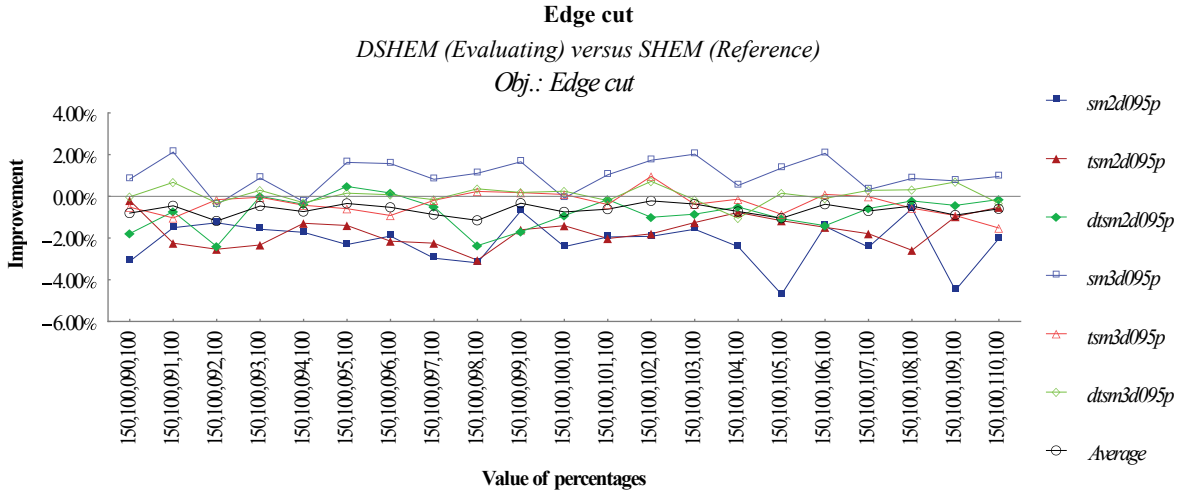


**Figure 8.28. DSHEM vs. SHEM: effect of *-dshem_p2* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

The results from the previous sets of experiments suggest that the percentage *-dshem_p2* could be used to tune up DSHEM and obtain better results according to the type of graph being used. This

particular set of experiments, with only synthetic irregular graphs, does not provide the same pattern when the value of the multiplier varies. Figure 8.28, Figure 8.29 and Figure 8.30 depict the effects of the percentage *-dshem_p2* with a range of 90 to 110. It is possible to see that the 2D square graph (*sm2d095p*) and the 3D triangular square graph (*tsm3d100p*) show the best results in different circumstances, as it has been with previous experiments.

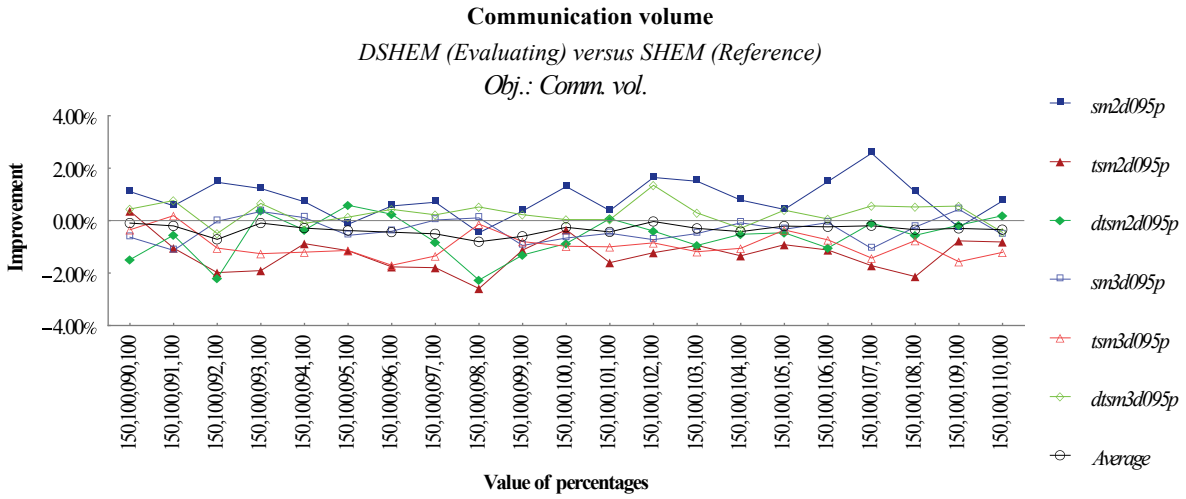


**Figure 8.29. DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

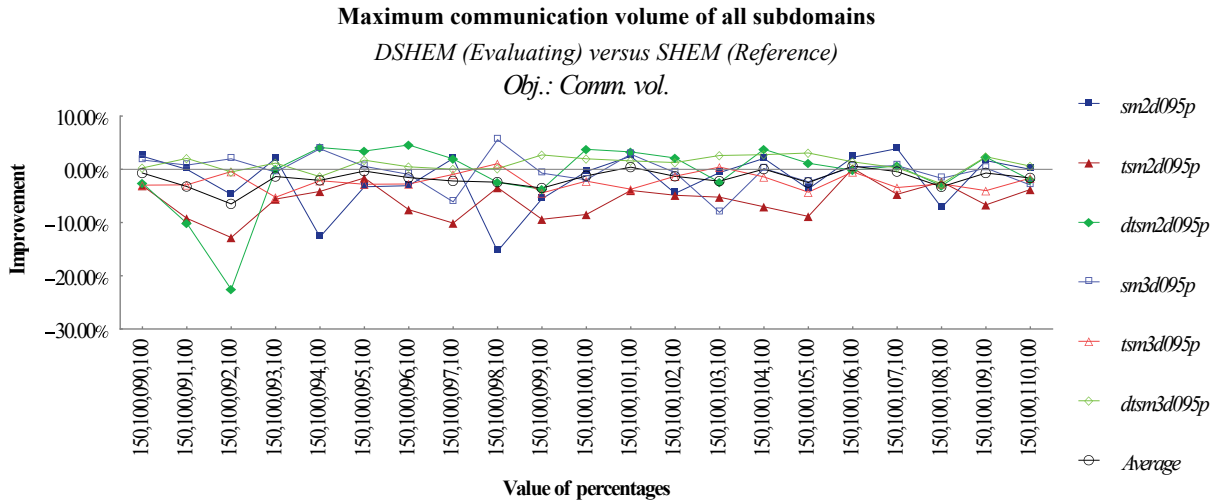


**Figure 8.30. DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

It is not possible to establish a discernible pattern on the effect of the percentage *-dshem_p2* with this particular set of experiments; this situation could be attributed to the degree of irregularity in the graphs.

### Graph Irregularity

The performance of DSHEM is studied with irregular graphs. This set of experiments utilizes only graphs with 95% of the edges. It has been done this way to mimic the real life graphs in a more controlled way.

## Refinement

Once more, METIS is executed without the refinement process to analyze its impact on the quality of the final partitions. Once the refinement is removed from the partitioning process, no objective is optimized and the real impact of DSHEM is shown.

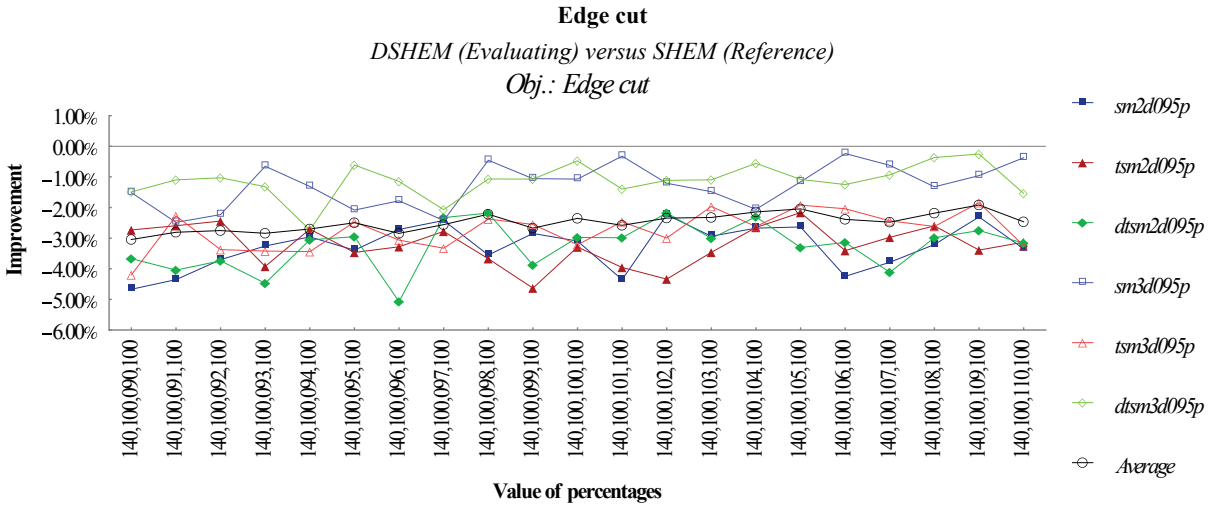


**Figure 8.31. DSHEM vs. SHEM: effect of refinement on the edge cut with synthetic graphs and edge cut as partitioning objective.**

Figure 8.31 suggests that the 3D square graph (*sm3d095p*) may benefit from *-dshem_p2* with values from 100 and higher; it could also be true for the 2D version. The 3D dense triangular square graph (*dtsm3d095p*) is also affected in the same way. However, it is not completely clear, but based on the results of previous sets of experiments this is more evident.

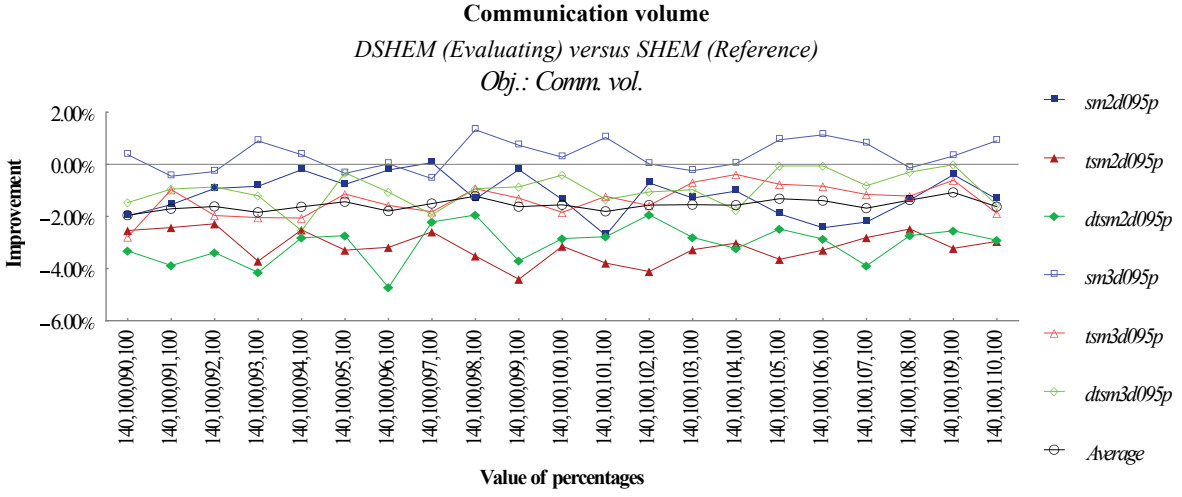


**Figure 8.32. DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

The communication volume, in Figure 8.32, provides not pattern. The experiments with regular graphs suggest that this metric remains stable and this results point in that direction. The 2D and 3D square graphs (*sm2d095p* and *sm3d095p*) continue on the top of the chart with better results compared to the rest.

Figure 8.33 shows that the 3D square graph (*sm3d095p*) benefits from DSHEM with an improvement of around 5% over all values of the percentage *-dshem_p2*. The graph in second place is the 3D dense

triangular square graph (*dtsm3d095p*) with just above the improvement line.

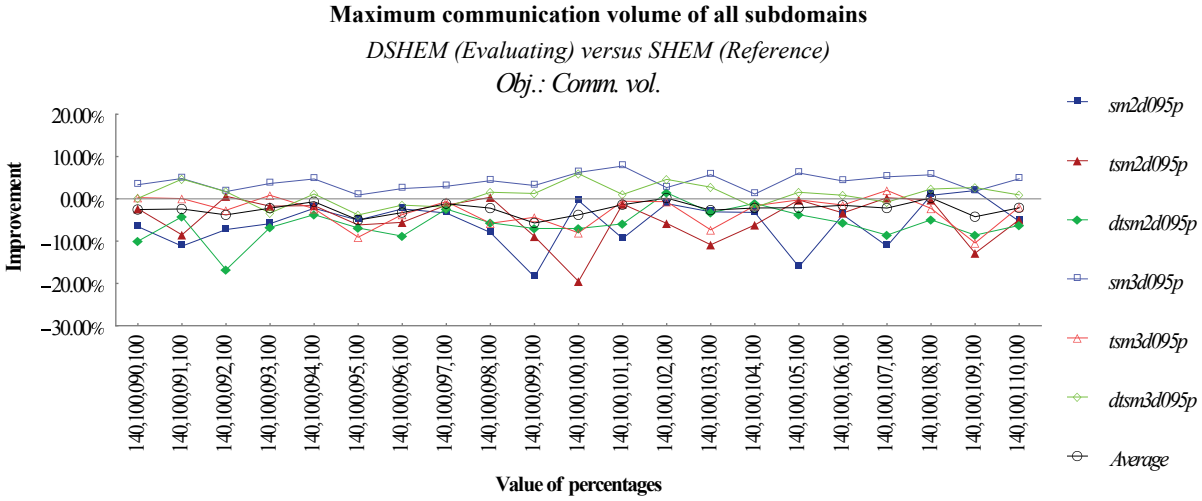


**Figure 8.33. DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Without the refinement process, it is possible to deduce that the graphs with quadrangular-like geometries benefit from DSHEM. Nonetheless, the benefits greatly depend on the particular instance of the problem.

### Execution Time

When METIS is executed without the refinement process, the execution time is mainly defined by the coarsening process and projection of the initial partition towards to the original graph. Only the time spent in the coarsening process may vary depending on the algorithm that is selected for the task. In this way, it is possible to accurately evaluate the expected degradation of the execution time by DSHEM. Without the refinement process, the execution time of METIS increases in a lower rate with the number of subdomains but varies with the matching algorithm; being Random and DSHEM the slowest. The type of graph also influences the partitioning time as demonstrated by the results.

DSHEM is slower compared to SHEM due to the extra time spend on the search of the opposite edges. However, the difference on time is only a few seconds in this set of experiments. If we consider that the overall execution time, including the refinement process, can reach up to 200 minutes, the extra time spent by DSHEM does not have any influence at all. The execution time without refinement is not presented in this section as it does not bring any contribution to the overall execution when the refinement is performed.

The partitioning time can be affected significantly according to the type of graph and partitioning objective as can be seen in Figure 8.34; being the 2D square graph the graph that brings a wider gap between the edge cut and communication volume. In general, Random is the slowest strategy of all. The difference of time between SHEM and DSHEM, with the 2D square graph, is around 1%. With the 2D triangular square graphs, DSHEM is 7% faster with the edge cut as partitioning objective and remains the same with the communication volume. With the 2D dense triangular square graph, DSHEM is 10% faster when the edge cut is the partitioning objective and remains constant with communication volume.

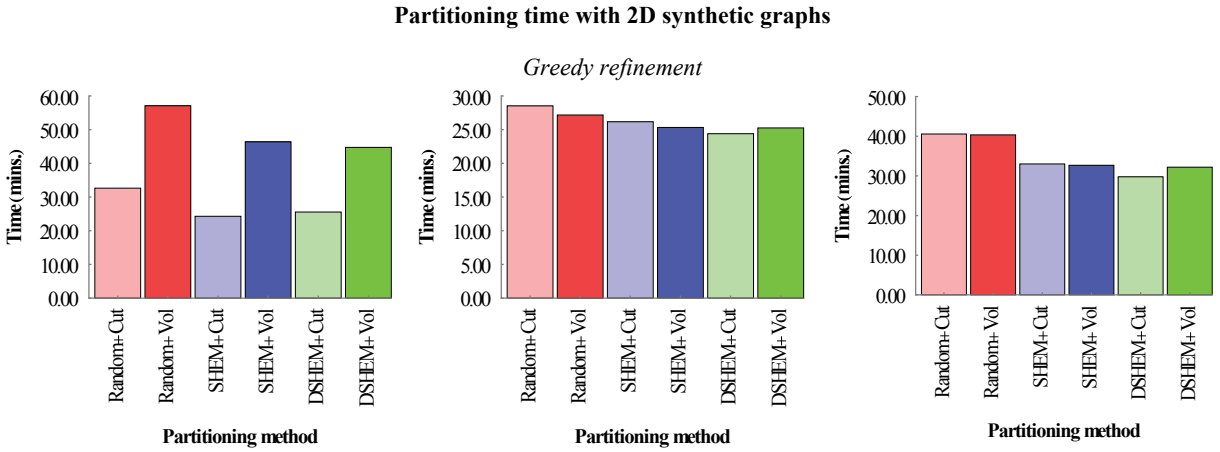


**Figure 8.34. Partitioning time of 2D synthetic graphs with 64 subdomains and greedy refinement. Type of graph: square on the left, triangular square on the middle, dense triangular square on the right.**

The 3D counterparts show a similar behavior of DSHEM, see Figure 8.35. For the 3D square graph, DSHEM is 5% faster with the edge cut and remains constant for the communication volume as partitioning objective. The 3D triangular square graph is different; DSHEM is 5% faster with the edge cut, but 9% slower with the communication volume. Finally, the 3D dense triangular square graph, DSHEM is 4% faster with the edge cut and remains constant with the communication volume.

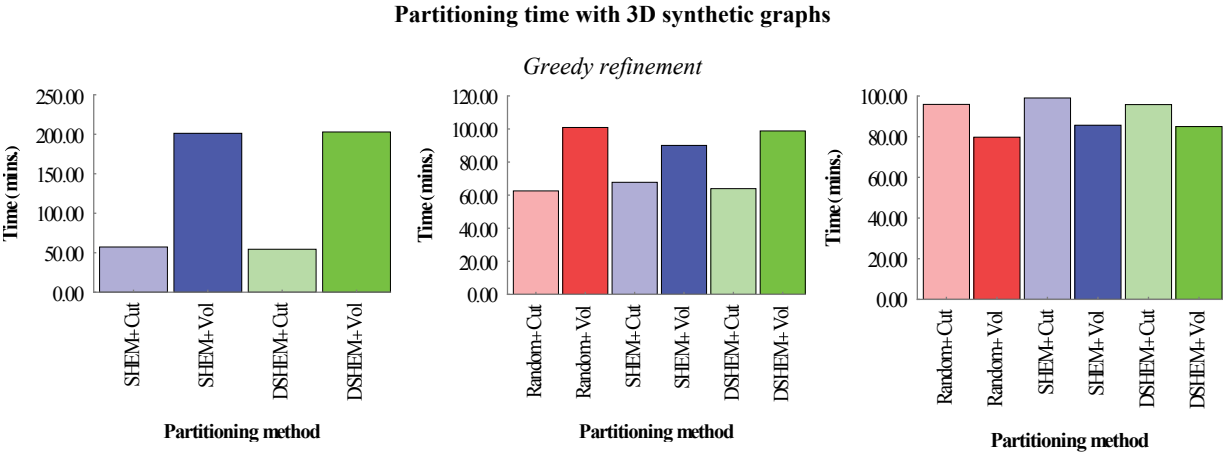


**Figure 8.35. Partitioning time of 3D synthetic graphs with 64 subdomains and greedy refinement. Type of graph: square on the left, triangular square on the middle, dense triangular square on the right.**

The refinement process is dependent of the partitioning objective; optimizing the communication volume could be up to 3.5 times slower as shown in Figure 8.35. The only graph that reverts the tendency is the 3D dense triangular square graph, when the optimization of the communication volume is faster than the edge cut. The two graphs with triangular geometries, in Figure 8.34, keep a balance time whether the edge cut or communication volume is optimized during the refinement process.

## 8.5. Experiments on Real Life Graphs

This particular set of experiments uses the graphs presented in Table 7.9 of Chapter 7. It is a set of real life graphs chosen to evaluate the performance of DSHEM and compare it with SHEM and Random.

### 8.5.1. Execution Parameters

Four main parameters are used to tune up DSHEM, namely *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. The values chosen for the fourth set of experiments are presented in Table 8.4. This particular set produces 1715 different combinations of values, giving a wide view of the performance of DSHEM. They are based on the results from previous experimental results, and used to confirm the findings with the synthetic graphs.

**Table 8.4: DSHEM parameters for the set of real life graphs.**

| *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|
| 140 to 160, step 5 | 91 to 109, step 3 | 91 to 109, step 3 | 91 to 109, step 3 |

### 8.5.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. First, the effect of the multiplier *-maxvwtm* is evaluated. Next, the three percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are examined to understand their influence. The values used for this particular set of experiments are based on the results of the previous three sets; they help validate the initial findings. The refinement and its influence on DSHEM are also studied to confirm the previous results obtained from the synthetic graphs. Finally, the execution time is also examined to estimate the degradation, if any, brought by DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

#### Multiplier *-maxvwtm*

The multiplier *-maxvwtm* limits the size of vertices during the coarsening process. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a low value may have also undesired effects such as the inability to match vertices that could lead to an infinite loop trying to contract the graph without success.

Based on the results obtained from the experiments with synthetic graphs, the values chosen for the multiplier *-maxvwtm* are designed to match those of the first set. They provide a wide view and reduce the number of experiments in the set.

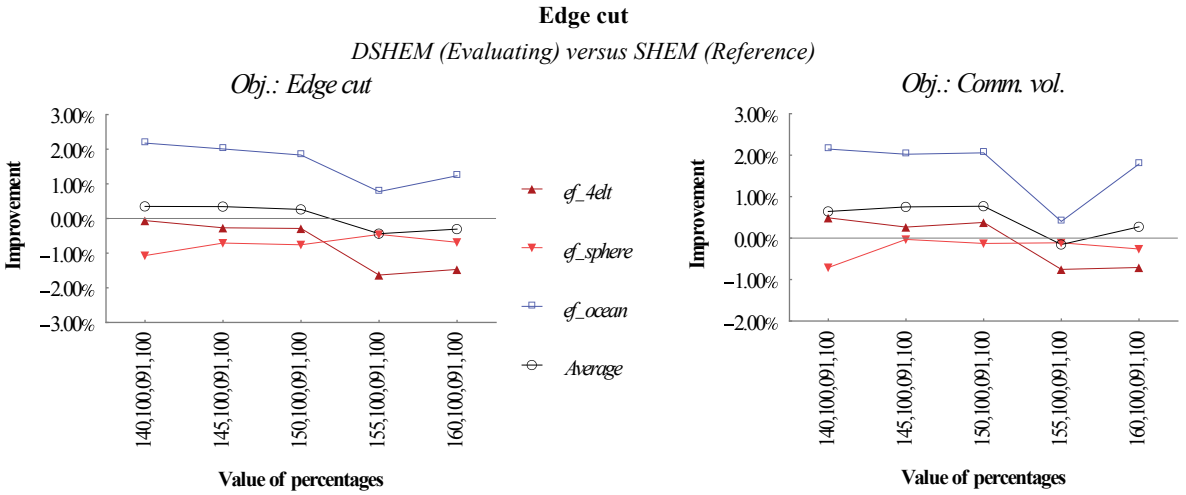


**Figure 8.36. DSHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

From the results shown in Figure 8.36 with real life graphs, it is possible to confirm the findings with the synthetic counterparts. The graph *ef_ocean* has a similar geometry to that of the 3D square graph and presents similar behavior: improvement over most of the values of multiplier *-maxvwtm*. The graphs *ef_4elt* and *ef_sphere* have a triangular geometry, as the 2D triangular square graph, and DSHEM produces poor results with them too.

Figure 8.37 and Figure 8.38 present the evaluation of the total communication volume and the maximum communication volume of all subdomains respectively. It is also evident that the graph *ef_ocean* has a clear improvement with DSHEM. The other two graphs, with triangular geometry, show degradation in the quality of the partition, as initially found with the synthetic graphs.

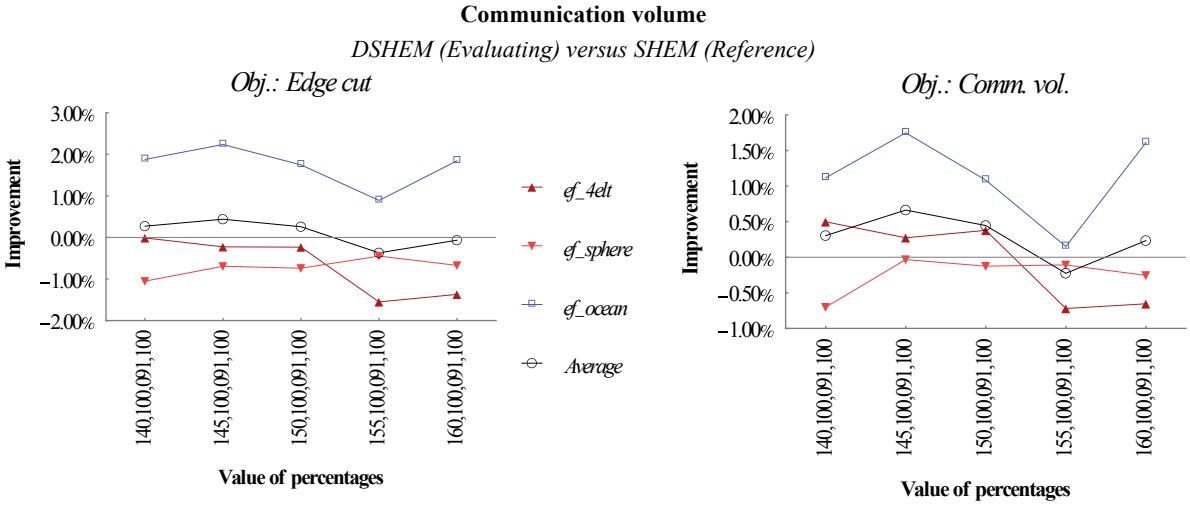


**Figure 8.37. DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

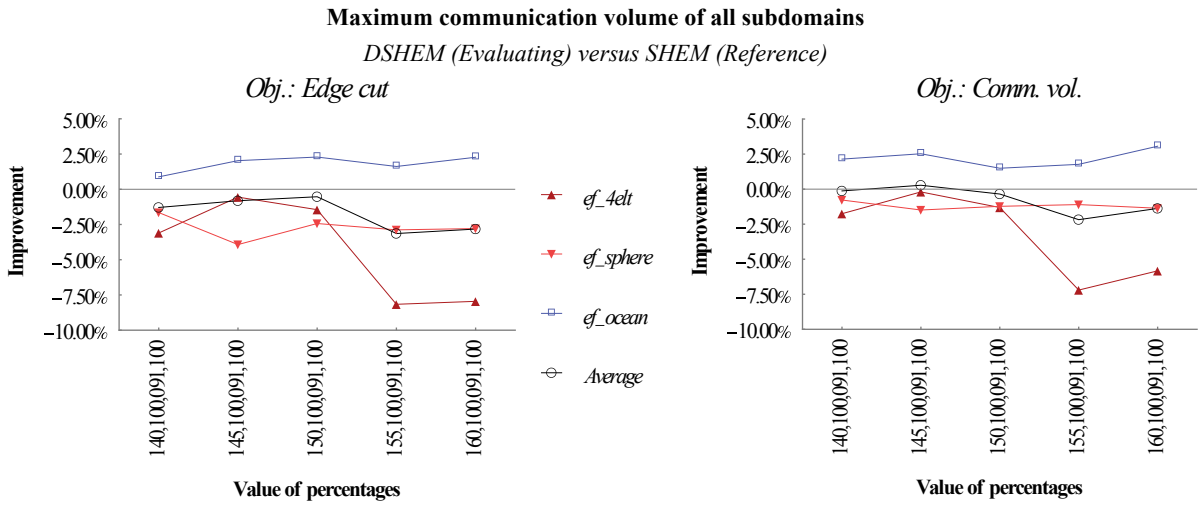


**Figure 8.38. DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

The results may suggest that smaller values for the multiplier *-maxvwtm* improve the results. Though, the results with the second set of experiments demonstrate that variations on the quality of the partition depend on the instance of the problem and not the multiplier.

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from the analysis as they do not play any role in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in DSHEM and improve the partition.

Values inferior to 100 for the percentage *-dshem_p2* produce better results with the graph *ef_ocean*, as shown in Figure 8.39, Figure 8.40 and Figure 8.41. Whether the partitioning objective is the edge cut or communication volume, the graph *ef_ocean* indisputably presents a benefit from DSHEM. The graphs *ef_4elt* and *ef_sphere* remain with little improvement or degradation.

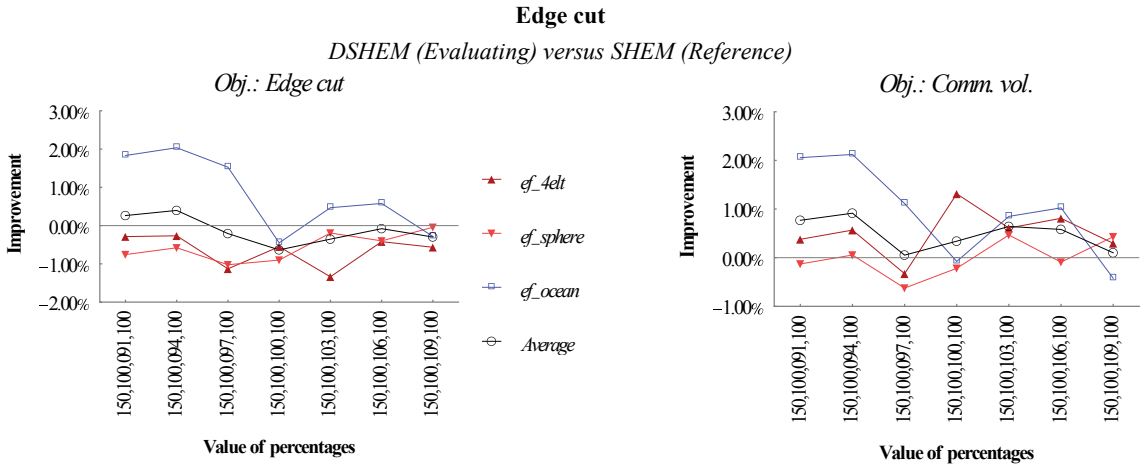


**Figure 8.39. DSHEM vs. SHEM: effect of *-dshem_p2* on the edge cut with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

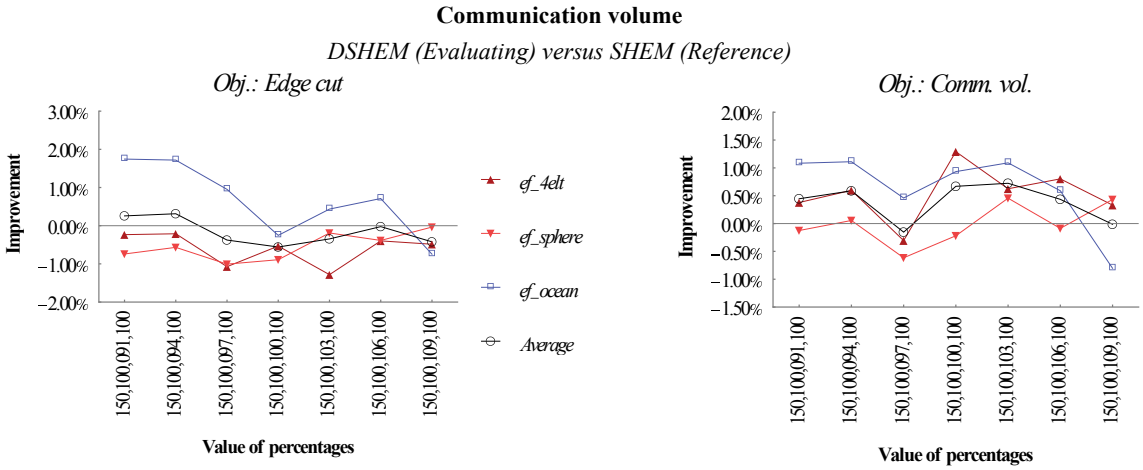


**Figure 8.40. DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

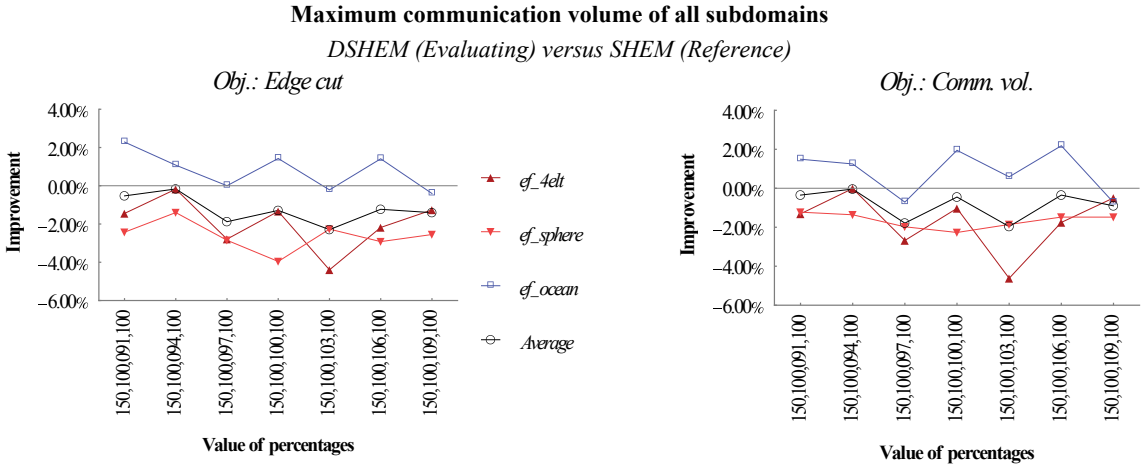


**Figure 8.41. DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

These results confirm the initial findings with the synthetic graphs that suggest the performance of DSHEM is superior with graphs having a quadrangular-like geometry.

## Refinement

METIS is executed without refinement for both, SHEM and DSHEM, to analyze its effect on the partitions. This process is responsible of the majority of the execution time; it improves the initial partition according to the partitioning objective.

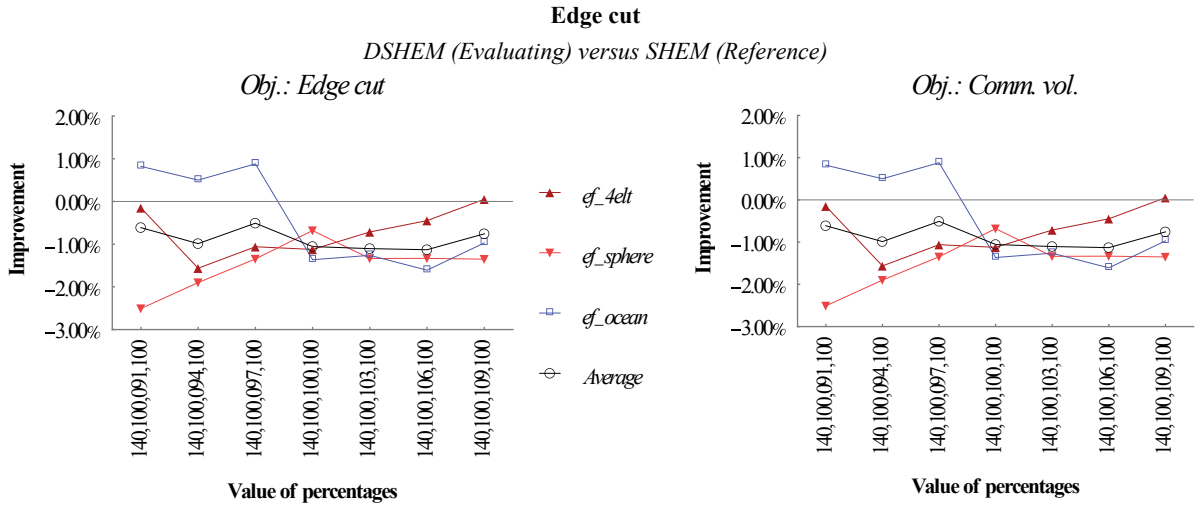


**Figure 8.42. DSHEM vs. SHEM: effect of refinement on the edge cut with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

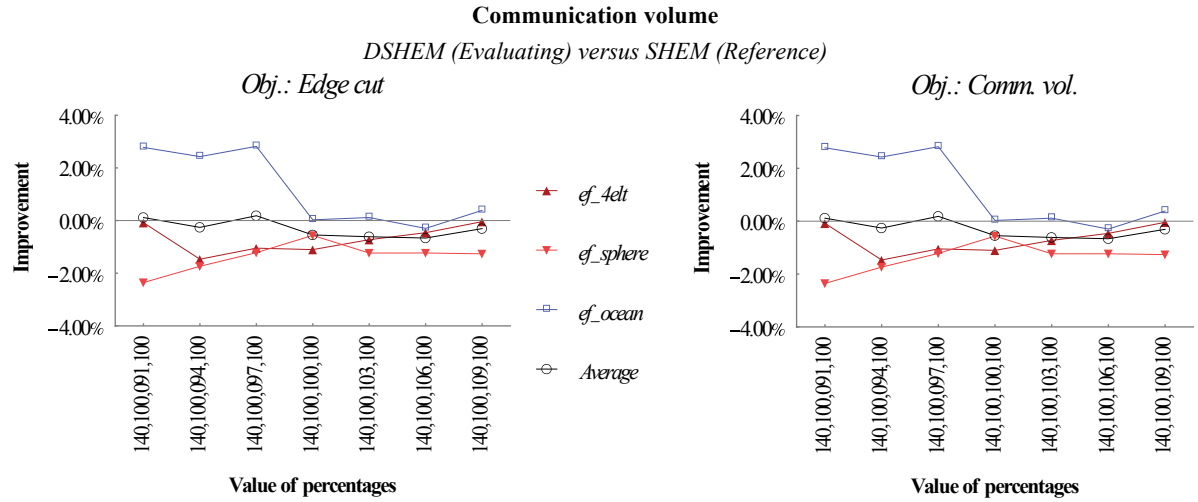


**Figure 8.43. DSHEM vs. SHEM: effect of refinement on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

It is interesting to note that the quality of the partition, when the edge cut is evaluated, improves when values for the percentage *-dshem_p2* are lower than 100, see Figure 8.42; it is the opposite with synthetic graphs and square geometry. It is not clear why DSHEM presents this behavior with the graph *ef_ocean*. The graphs with triangular geometry have a similar output to the synthetic 2D triangular square graph.

Regarding the total communication volume and the maximum communication volume of all subdomains, Figure 9.31 and Figure 9.32 depict a congruent scenario with the synthetic graphs. The total communication volume improves with values for the percentage *-dshem_p2* being 100 or higher. The maximum communication volume of all subdomains is more irregular, but in general, lower values produce better results.

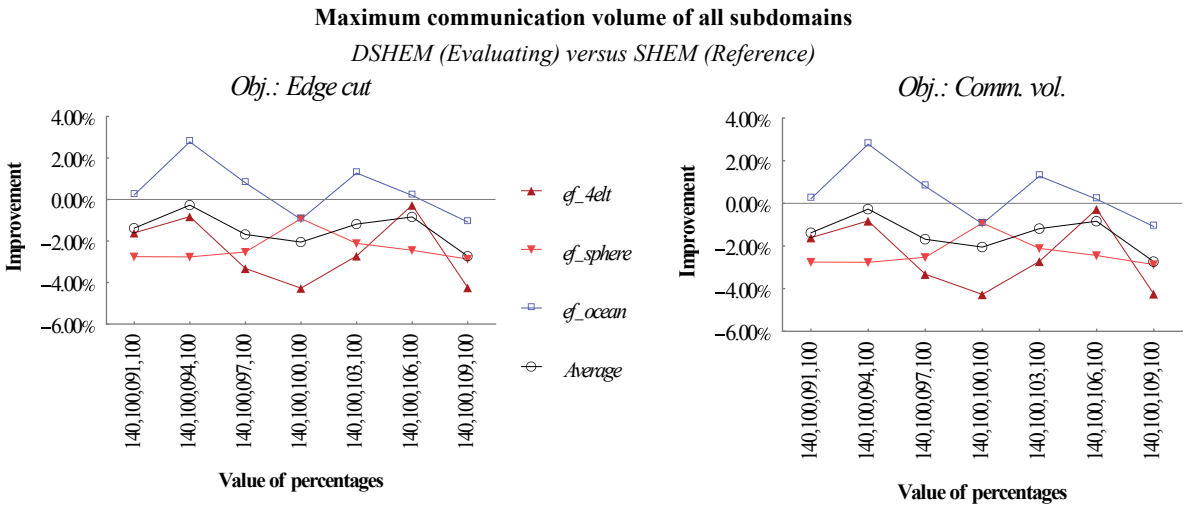


**Figure 8.44. DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

## Execution Time

The two graphs with triangular geometries confirm that this type keeps the partitioning time balanced between edge cut and communication volume, as depicted in Figure 8.45 and Figure 8.46. There is virtually no difference in execution time whether the edge cut or communication volume is optimized by the refinement process.

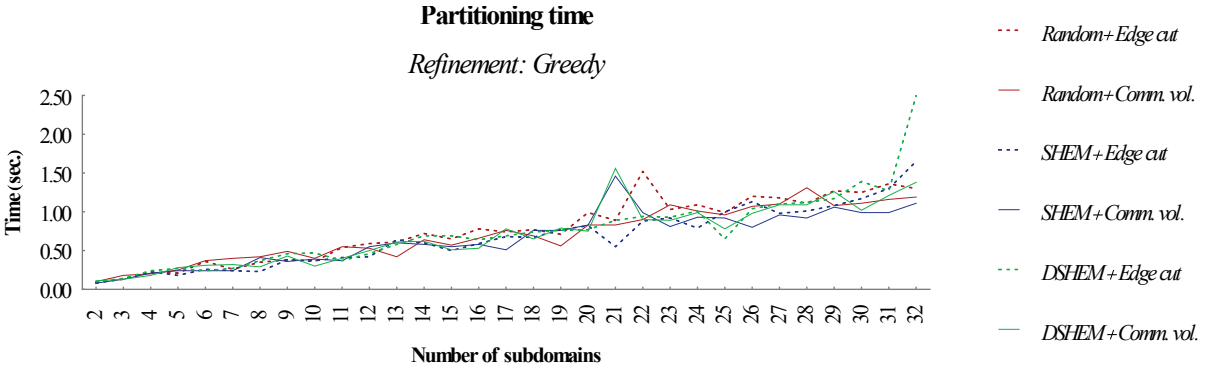


**Figure 8.45. Partitioning time with graph *ef_4elt* and greedy refinement.**

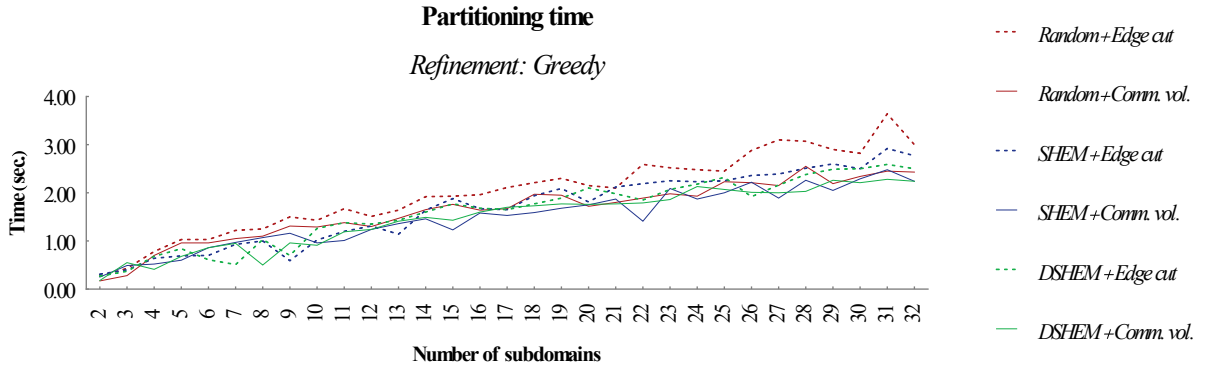


**Figure 8.46. Partitioning time with graph *ef_sphere* and greedy refinement.**

The experimental results confirm that the time spent on the refinement process greatly depends on the number of subdomains; from a few seconds for 2 subdomains to some minutes for 32 subdomains with

the graph *ef_ocean*. In addition, it is possible to deduce that the partitioning objective has also an effect on the execution time; communication volume could be up to four times slower than edge cut for 32 subdomains, as shown in Figure 8.47. The matching strategy affects the refinement process too; it takes around 35% longer when Random and communication volume are used instead of SHEM or DSHEM. When METIS is executed with 32 subdomains with the graph *ef_ocean*, the refinement process takes around than 99% of the total time of execution.

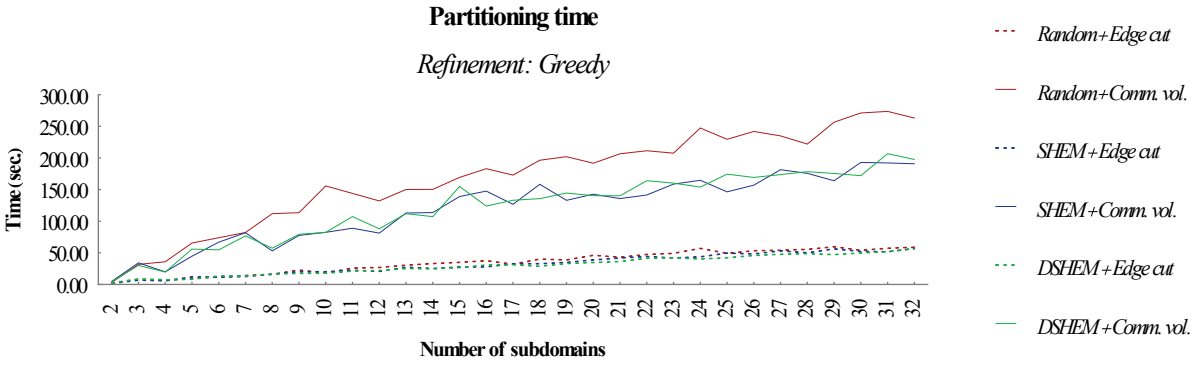


**Figure 8.47. Partitioning time with graph *ef_ocean* and greedy refinement.**

The results show that DSHEM does not increase the partitioning time regardless the number of subdomains or the type of graph. It depends on the particular situation whether SHEM or DSHEM is faster, by a marginal value, compared to the counterpart.

## 8.6. Discussion

The initial study of DSHEM provides important information of its efficiency and performance. The discussion is divided in the same way as the experiments. Next, a general overview of the different executions of DSHEM is presented.

### 8.6.1. Impact of Multiplier *-maxvwtm*

The initial results suggest that the multiplier *-maxvwtm* could help improve the partitions generated by DSHEM when its value is reduced from the default 150. Subsequent experiments show that there is not a clear pattern on how the multiplier affects the performance of DSHEM. It greatly depends on the type of graph, objective to optimize and metric to consider.

Despite the lack of correlation between the value of the multiplier and the quality of the final partition, it seems that DSHEM produces better results more frequently when the multiplier is smaller than 150. It would be advisable to test the graph and subsequently adjust the value to improve the partitions for a particular graph.

### 8.6.2. Impact of Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

The original design of DSHEM contemplates three different percentages that work together to improve the partitions according to the type of graph being partitioned. The initial results show that the two percentages *-dshem_p1* and *-dshem_p3* do not modify the behavior of the algorithm. It is also confirmed

with subsequent experiments, even though the results are not presented in this thesis. A close analysis was also performed with a clear conclusion, the conditionals that use these two percentages evaluate to true only the first time, rendering them superfluous.

When the refinement is not part of the partitioning process, the percentage *-dshem_p2* has a measurable and predictable impact on DSHEM. The value 100 is an inflexion point for most of the synthetic graphs, some showing improvement when the percentage is 100 or higher and others when lower. When the refinement is performed, that pattern is modified and less evident; nonetheless, it is still possible to infer a change around the value 100.

### 8.6.3. Impact of Graph Irregularity on DSHEM

Upon the study of the experimental results, there is not a clear correlation between the irregularity of the graphs and the quality of partitions DSHEM produces. The irregularity introduced to the synthetic graphs affects in a similar way to Random, SHEM and DSHEM without a clear pattern; it is more the instance of the problem that changes the results than the amount of irregularity.

It is reasonable to assume that if the degree of irregularity reaches a threshold, it will affect DSHEM in a direct way. As the results show, DSHEM tends to perform better with certain types of geometries. If the degree of irregularity reaches the point when the geometry of the graph is lost, then DSHEM will no longer guarantee a good partition. Based on the results presented in this chapter, DSHEM can perform well within a reasonable degree of irregularity introduced to the graphs.

### 8.6.4. Impact of Refinement on DSHEM

The refinement plays an important role in the partitioning process. It improves the initial partition of the coarsest graph and keeps the quality during the projection of that partition back to the original graph. The two partitioning objectives that can be optimized by the refinement are the edge cut and the communication volume; in most cases, optimizing the edge cut is much cheaper.

The results from the synthetic graphs show that DSHEM can improve the quality of the partitions when the graph has a quadrangular-like geometry, and in certain circumstances with triangular-like geometries. Some patterns emerge from the synthetic graphs that can be used to ensure a higher quality in the partition according to the type of graph.

Once the refinement process is included in the partitioning process, those patterns are distorted, and in some cases replaced. This leads as to believe that the refinement process does not efficiently interact with the coarsening process. Random and SHEM were originally designed to reduce the number of edges in the cut and the refinement process designed accordingly. In a later release of METIS, the optimization of the communication volume was introduced. The refinement process tries to optimize one objective when the graph was contracted with a different one; the original coarsening process remains the same, focused on the edge cut.

DSHEM changes the focus of the coarsening process in order to improve the partitions when the communication volume is considered, but the refinement process is still not designed to interact with the new objective and in some cases it degrades the quality of the partitions.

### 8.6.5. Impact of DSHEM on the Execution Time

The two steps in the partitioning process that contribute the most to the execution time is the coarsening process and, in greater degree, the refinement process. The coarsening process contracts the graph to a size that is easier to work with. The refinement process improves the initial partition throughout the projection back to the original graph; it optimizes the partitioning objective.

From the experimental analysis, it is possible to estimate the overall impact of the coarsening process on the time required to partition a graph. It is evident that the time required by the three strategies, namely Random, SHEM and DSHEM, varies with the type of geometry present in the graph. DSHEM is the most complex algorithm and evidently the slowest too. Nonetheless, it would be incorrect to state that DSHEM is in disadvantage do to the longer processing time required to contract the graph; the structure of the contracted graph affects the refinement process too.

The refinement process can take up to 99% of the execution time and it is heavily affected by the coarsening algorithm. Random produces poor coarse graphs and the refinement process requires longer time to improve the partition. SHEM and DSHEM require longer time to contract the graph, but the refinement process is faster as the contracted graphs have a much higher quality and less time is necessary to improve the partitions. In many cases, the overall partitioning time is reduced up to 7% when DSHEM contracts the graph compared to SHEM.

### 8.6.6. Global Evaluation of DSHEM

In general, it can be concluded that DSHEM can improve the quality of the partitions when the communication volume is considered. Under certain circumstances the improvement is guaranteed; especially when the graph has a quadrangular-like geometry, whether it is a 2D or 3D variation.

It is still not completely clear how the refinement process affects the performance of DSHEM, but the experimental analysis suggests that its design is not ideal for the type of coarse graph produced by DSHEM. This is clear when the refinement is not part of the process and DSHEM constantly produces better results compared to SHEM. Once the refinement is included, the benefits of DSHEM are degraded or even wiped out. Still, tuning up the algorithm with the different execution parameters may help keep that improvement when DSHEM is used to contract the graph. It is also evident that optimizing the communication volume is more expensive than the edge cut; attributed to the fact that the initial design of the coarsening process in METIS focuses on the edge cut.

Although DSHEM is slower during the coarsening process, due to the search of the opposite edges, the overall execution time of METIS is not affected, and in many cases reduced. This is due to the structure of the coarsest graph generated by DSHEM and the refinement process spending less time improving the partition throughout the projection back to the original graph.

The irregularity introduced to the graphs has no clear effect on the efficiency of DSHEM. The results remain stable when the degree or irregularity is reasonable; when the geometry of the graph is kept.

Finally, the analysis performed with the real life graphs confirms the findings with the synthetic counterparts. DSHEM brings clear benefits when the geometry of the graph is quadrangular.

# Chapter 9.

# Experimental Analysis of Nested DSHEM

This chapter presents the second evaluation of DSHEM, using the nested version, with four different sets of experiments as described in Chapter 7. Similar to the previous evaluation, nested DSHEM is executed with small graphs and a wide range of values for the parameters *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. Based on those results, a new set of experiments is performed with more precise parameters to tune up the algorithm. The third set is designed to evaluate the scalability of nested DSHEM with large graphs. The final set of experiments helps confirm the evaluation results. The next sections present a detailed evaluation of the nested version of DSHEM.

## 9.1. Nested DSHEM Partitioning

The evaluation of nested DSHEM is based on the implementation of the strategy described in Chapter 6, Section 6.3.3, Nested DSHEM Partitioning; a joint effort by SHEM and DSHEM to generate the partitions. This implementation increases the memory requirements as a second graph is created and kept in memory during the partitioning process.

## 9.2. First Experiments on Small Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.6 of Chapter 7. It is a set of small synthetic graphs which are designed to evaluate the performance of nested DSHEM and compare it with SHEM and Random.

### 9.2.1. Execution Parameters

Five main parameters are used to tune up nested DSHEM, namely *-nctype*, *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. The values chosen for the first set of experiments are presented in Table 9.1. This particular set produces 3430 different combinations of values, giving a wide view of the performance of nested DSHEM. The subsequent experiments are designed based on the results of these initial experiments.

**Table 9.1: Nested DSHEM parameters for the first set of small synthetic graphs.**

| *-nctype* | *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|---|
| dshem+shem<br>shem+dshem | 140 to 160, step 5 | 91 to 109, step 3 | 91 to 109, step 3 | 91 to 109, step 3 |

### 9.2.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. First, the effect of the multiplier *-maxvwtm* is evaluated. Next, the three percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are examined to understand their influence. The robustness of nested DSHEM is also evaluated with different degrees of irregularity introduced to the synthetic graphs. The refinement and its influence on nested DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by nested DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

#### Multiplier *-maxvwtm*

The multiplier *-maxvwtm* limits the size of vertices during the coarsening process. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a low value may have also undesired effects such as the inability to match vertices that could lead to an infinite loop trying to contract the graph without success.

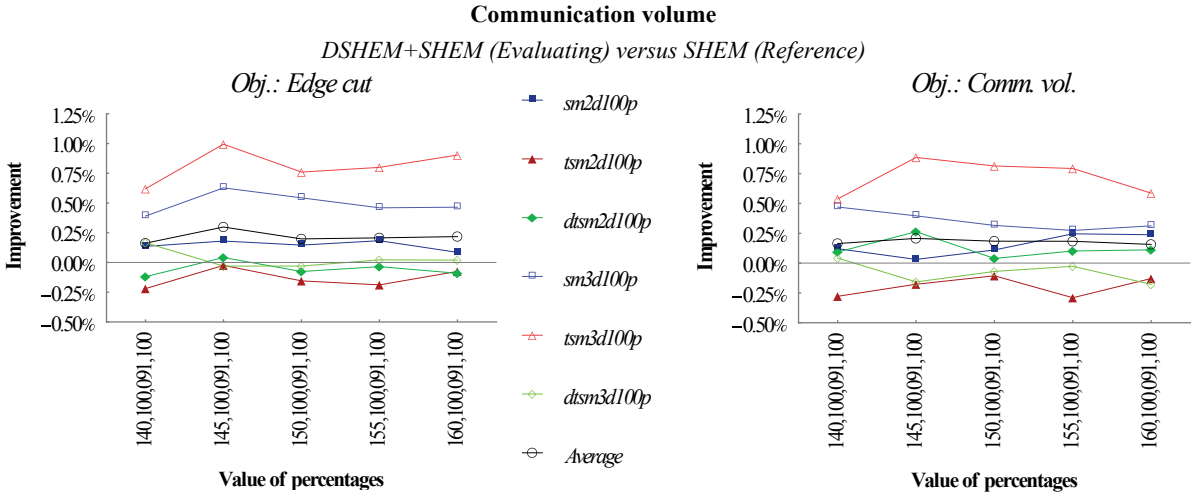


**Figure 9.1. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Regarding the total communication volume, Figure 9.1 suggests that multiplier *-maxvwtm* does not have a discernible pattern on how it influences the final partition with the nested DSHEM executed as

DSHEM+SHEM. Similar results can be found with SHEM+DSHEM in Figure 9.2; however the improvement achieved with some types of graphs is reduced.

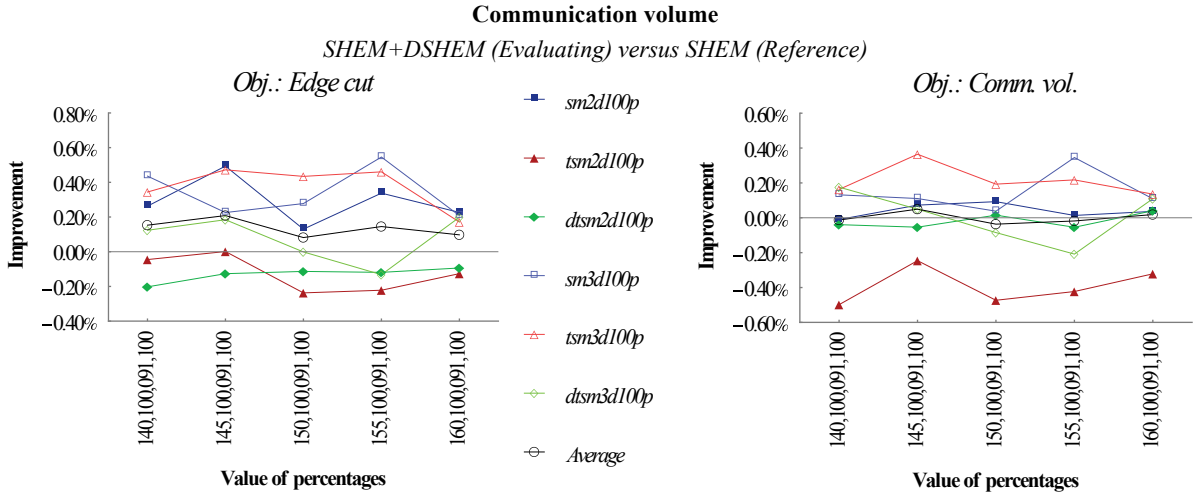


**Figure 9.2. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

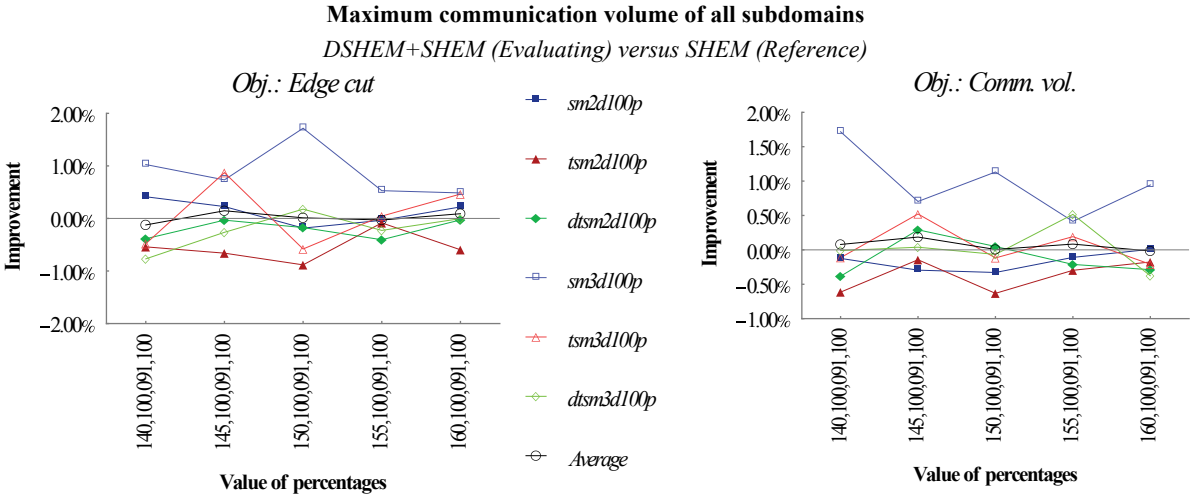


**Figure 9.3. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

When the maximum communication volume of all subdomains is considered, Figure 9.3 shows that in some cases *-maxvwtm* has an important impact when nested DSHEM is executed as DSHEM+SHEM with some types of graphs: the 3D square graph (*sm3d100p*) and the 3D triangular square graph (*tsm3d100p*). When nested DSHEM is executed as SHEM+DSHEM, a similar scenario is shown in Figure 9.4.

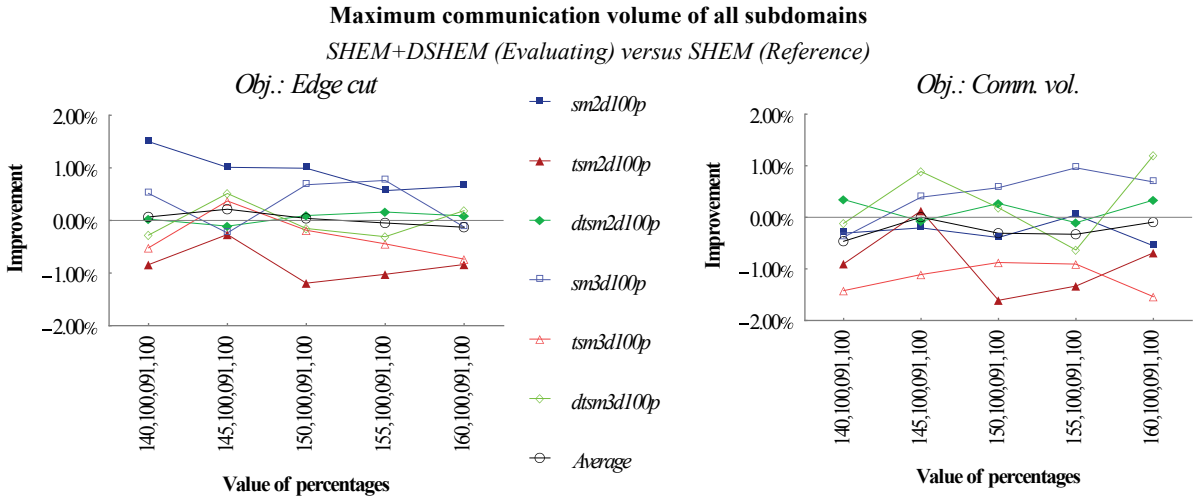


**Figure 9.4. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

## Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are used to modify the behavior of the cost function in nested DSHEM. It may improve the results by selecting the right values according to the type of graph to partition.

Percentages *-dshem_p1* and *-dshem_p3* have no impact in the partitioning process of nested DSHEM, as found in the experimental analysis of the full DSHEM implementation in the previous chapter. For this reason, the results are not included in this chapter.

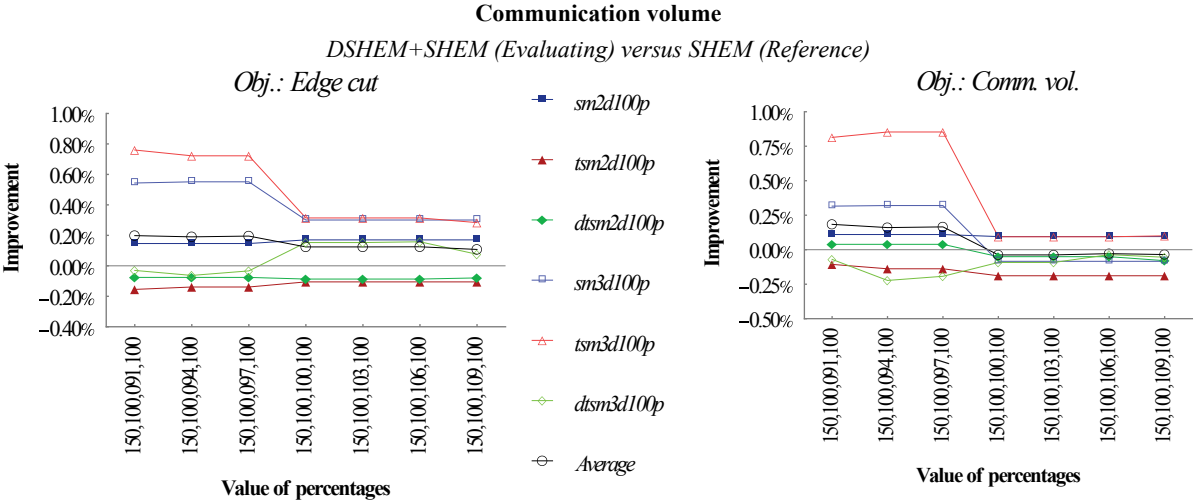


**Figure 9.5. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

With nested DSHEM executed as DSHEM+SHEM, shown in Figure 9.5, percentage *-dshem_p2* has a clear impact in total communication volume of the final partition, with an inflection point in the value 100. The types of graphs with the biggest impact are the 3D triangular square graph (*tsm3d100p*), the 3D square graph (*sm3d100p*) and the 3D dense triangular square graph (*dtsm3d100p*) when the edge cut is the partitioning objective. The 2D dense triangular square graph (*dtsm2d100p*) is also affected when the partitioning objective is the communication volume. With nested DSHEM executed as SHEM+DSHEM,

shown in Figure 9.6, the pattern is more unpredictable. The 3D triangular square graph (*tsm3d100p*) is heavily impacted by the percentage *-dshem_p2* when the partitioning objective is the edge cut; the rest of the types of graphs have little or no impact. The total communication volume remains stable with the communication volume as partitioning objective. This situation can be attributed to the fact that the coarsening process was performed by SHEM, and the subsequent partitioning and refinement of the coarsest graph with DSHEM values could not improve the results.

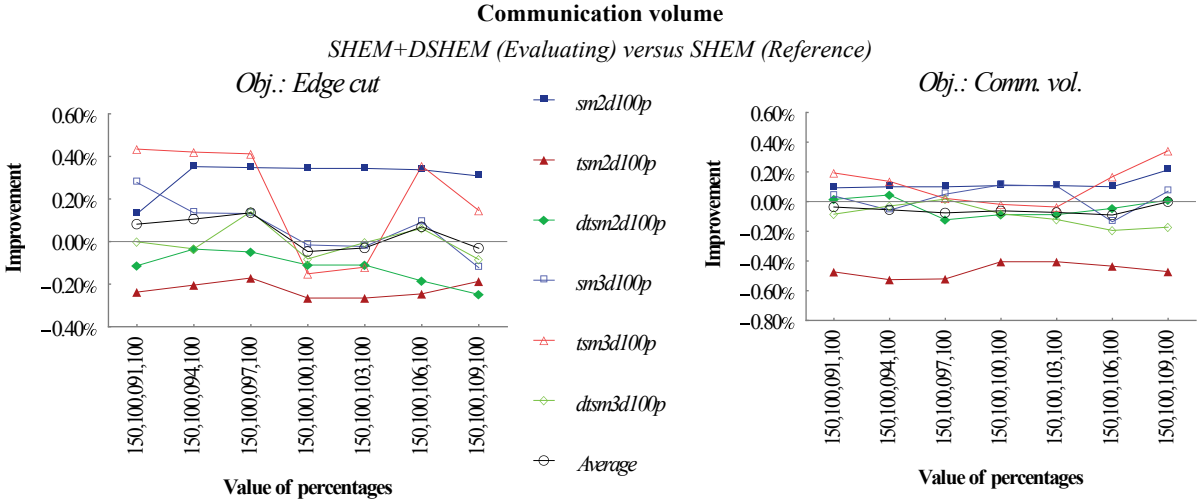


**Figure 9.6. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

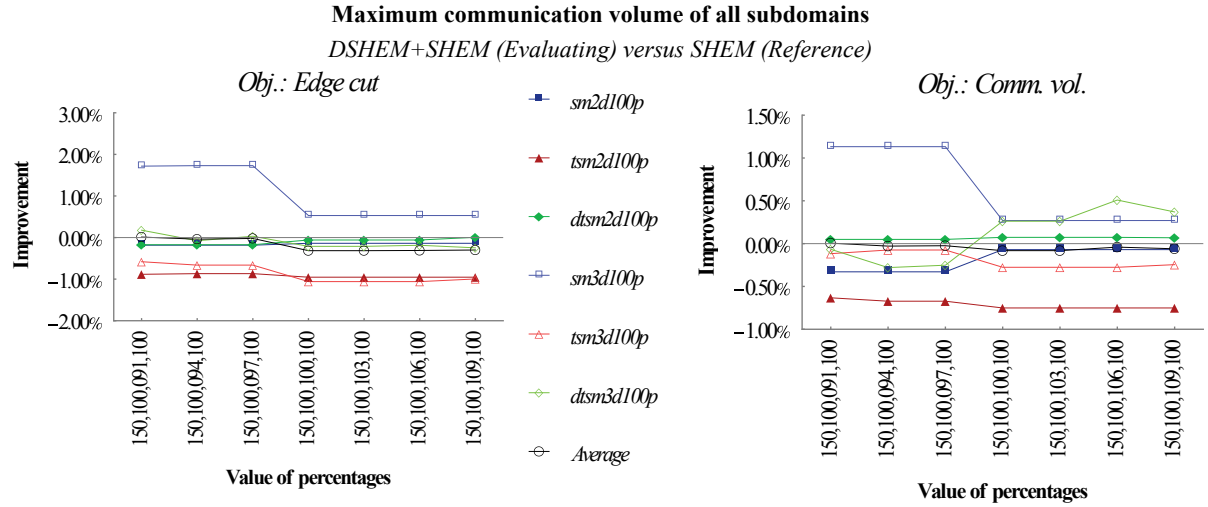


**Figure 9.7. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Figure 9.7 and Figure 9.8 show the impact of percentage *-dshem_p2* on the maximum communication volume of all subdomains when nested DSHEM is executed as DSHEM+SHEM and SHEM+DSHEM respectively. It is possible to see that the results are similar to those of the total communication volume. It appears that the best results can be achieved with DSHEM+SHEM; and more predictable too.

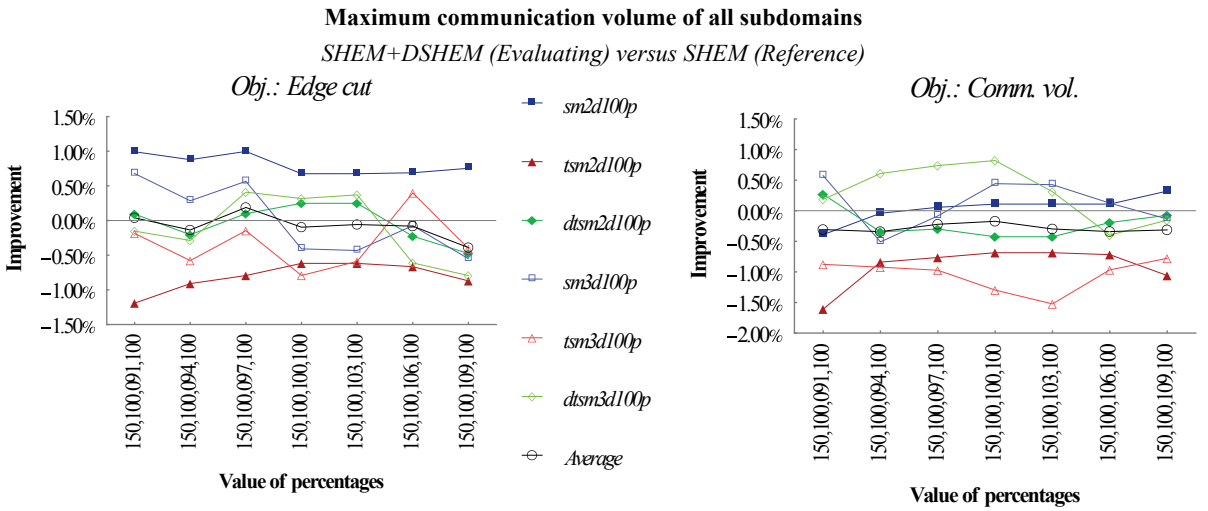


**Figure 9.8. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

## Graph Irregularity

The performance of nested DSHEM is also studied with irregular graphs. Sizes $a$ to $e$, in Figure 9.9, Figure 9.10, Figure 9.11 and Figure 9.12, represent the five biggest 2D and all 3D graphs of the set.

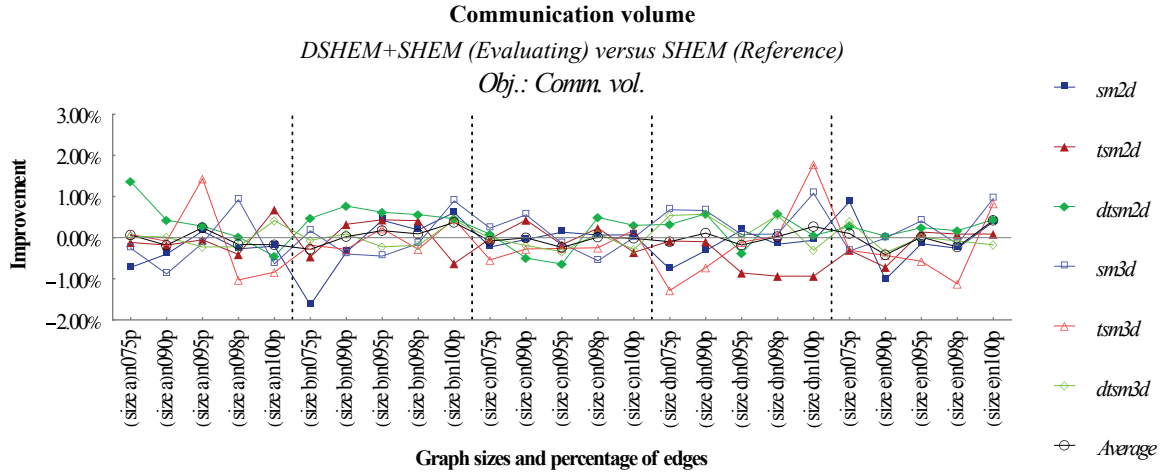


**Figure 9.9. DSHEM+SHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

From the results in Figure 9.9 and Figure 9.10, the impact on the total communication volume seems to have a random pattern. However, a closer analysis shows that some types of graphs suffer more than others. For example 2D and 3D triangular square graphs (*tsm2d* and *tsm3d*) and the 2D square graph (*sm2d*) are the most impacted; nevertheless, the degree of the irregularity in the graphs is not proportional to the impact on the quality of the partition.

Regarding the maximum communication volume of all subdomains, similar behavior can be seen in Figure 9.11 and Figure 9.12. The irregularity impacts the final partition, but it does not degrade or improves it with a clear pattern; it is dependent of the instance of the problem.

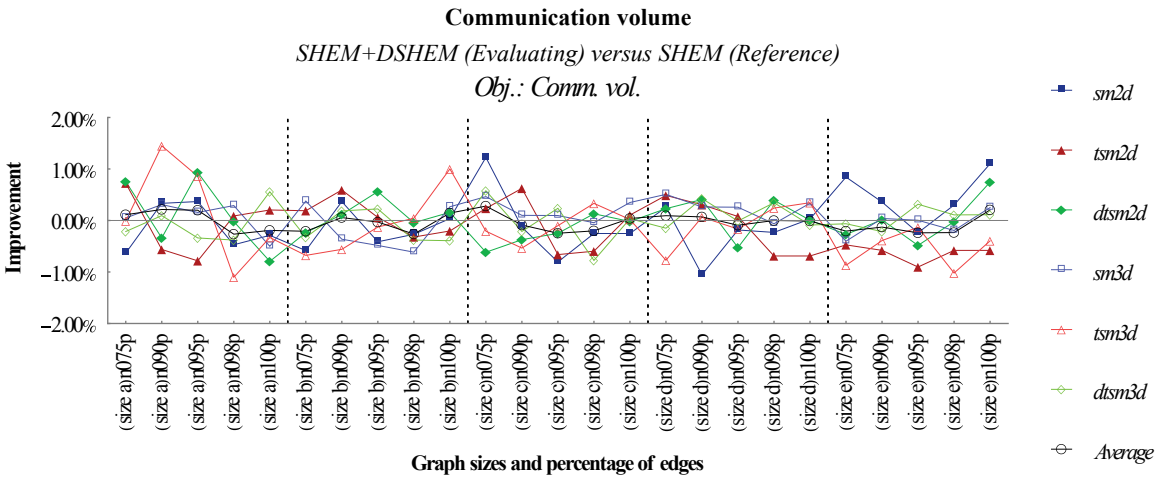


**Figure 9.10. SHEM+DSHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

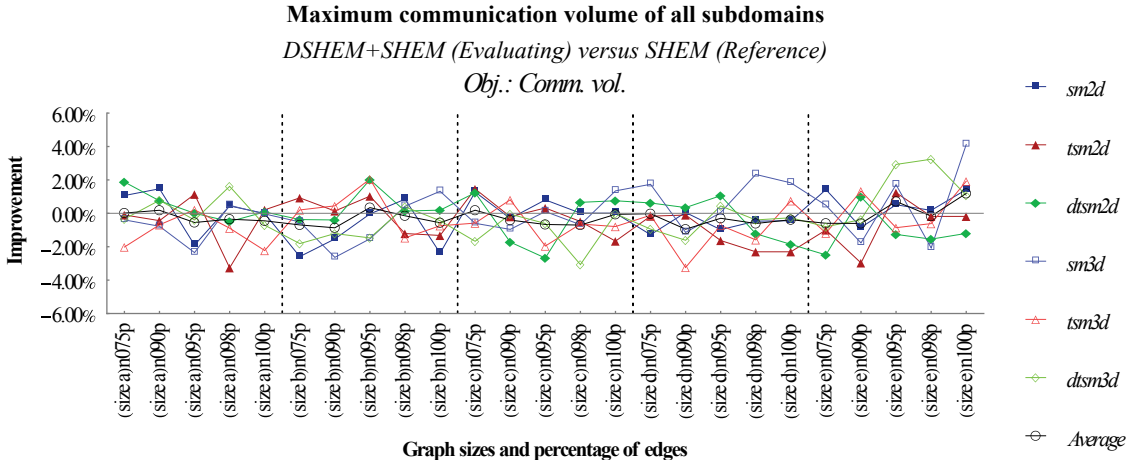


**Figure 9.11. DSHEM+SHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

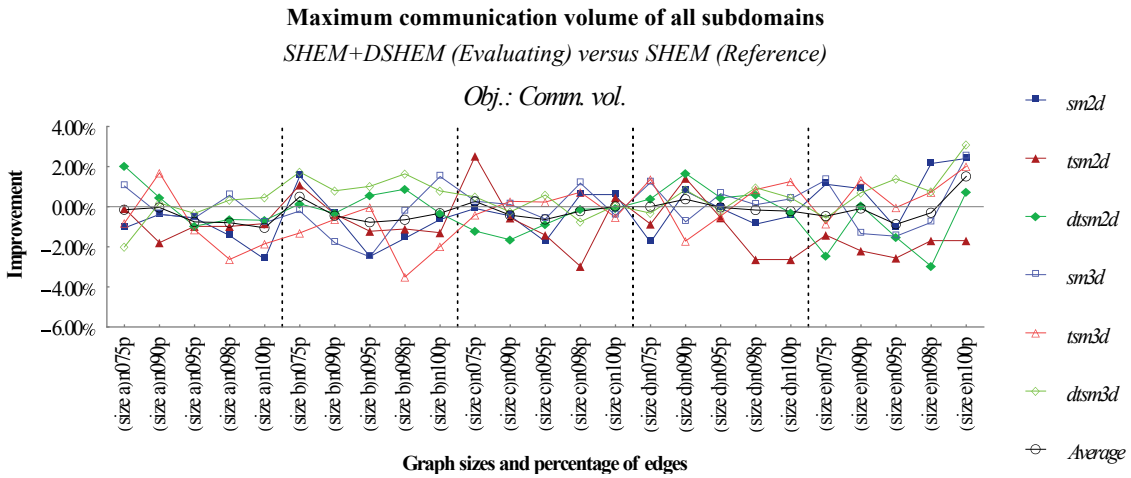


**Figure 9.12. SHEM+DSHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

### Refinement

METIS is executed without refinement for both, SHEM and nested DSHEM, to analyze the effect on the partitioning process. Whether the partitioning objective is the edge cut or the communication volume, the results remain the same; SHEM and nested DSHEM perform the matching without a partitioning objective and without the refinement process no objective is optimized.

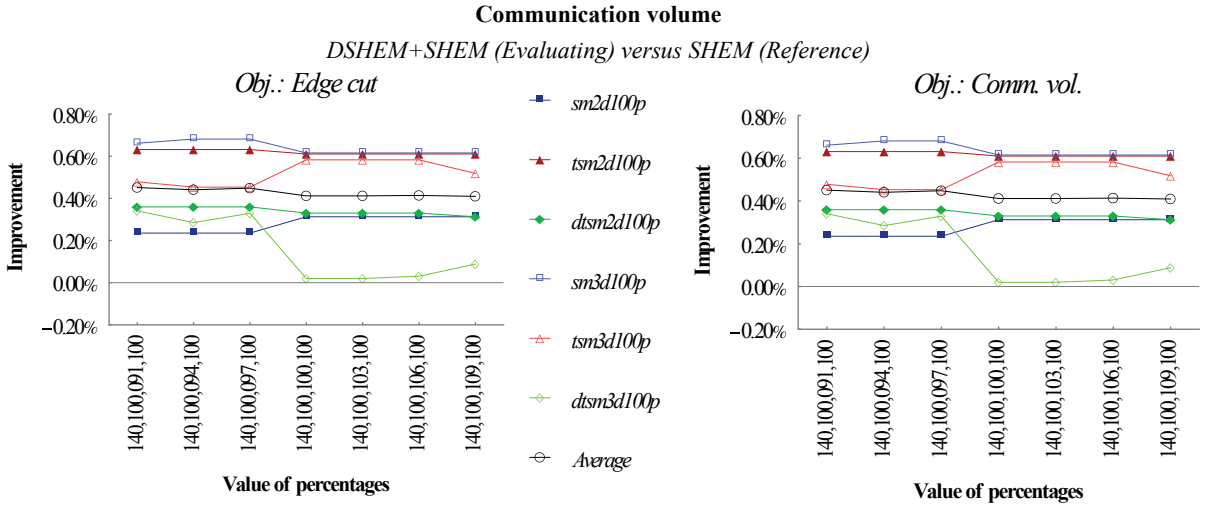


**Figure 9.13. DSHEM+SHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

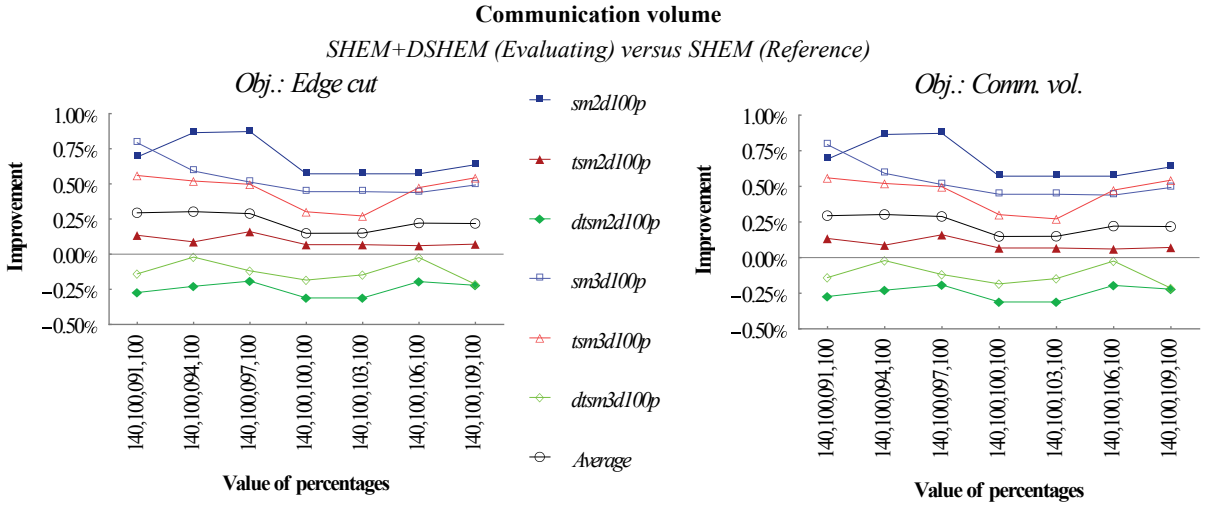


**Figure 9.14. SHEM+DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

It is possible to see the real advantage of nested DSHEM when METIS is executed without refinement. There is always an improvement on the total communication volume when nested DSHEM is executed as DSHEM+SHEM; shown in Figure 9.13. If executed as SHEM+DSHEM, the partitions of the 2D square graph (*sm2d100p*) have the greatest improvement, see Figure 9.14; the rest of the types of graphs suffer a negative impact compared to DSHEM+SHEM.

With respect to the maximum communication volume of all subdomains, Figure 9.15 and Figure 9.16 illustrate an analogous scenario as that of total communication volume; the 2D square graph (*sm2d100p*) benefits more from SHEM+DSHEM that DSHEM+SHEM.

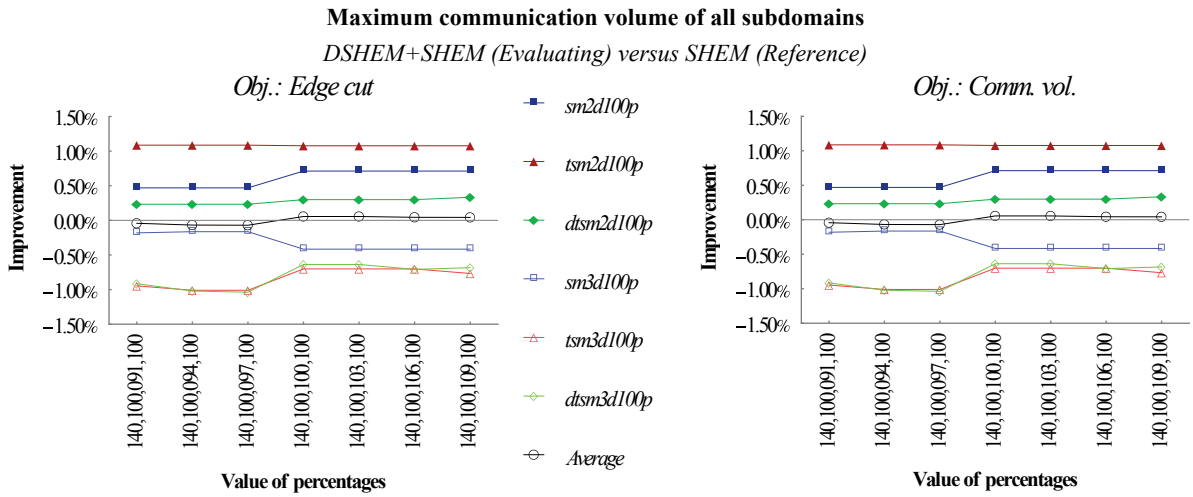


**Figure 9.15. DSHEM+SHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

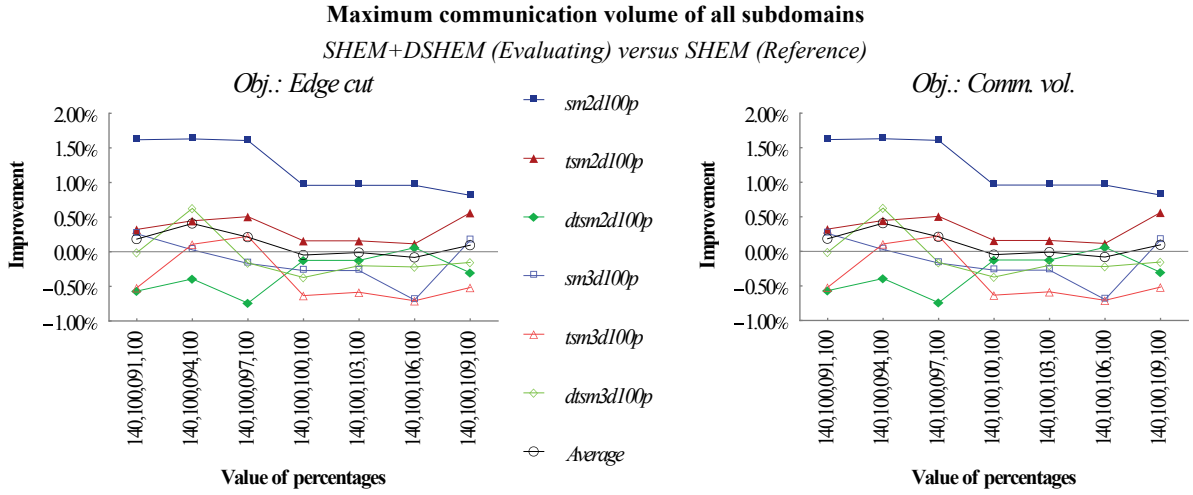


**Figure 9.16. SHEM+DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

It is interesting to notice that the 2D square graph (*sm2d100p*) is heavily affected by the refinement process. The higher quality of the partitions with SHEM+DSHEM is degraded when the refinement is performed. This is evident when comparing these results to those of the percentage *-dshem_p2*.

### Execution Time

The graphs for these experiments are small enough to make virtually impossible to evaluate the impact of nested DSHEM on the execution time. The majority of the running times does not even reach one second.

## 9.3. Second Experiments on Small Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.7 of Chapter 7. It is a set of small synthetic graphs which are designed to evaluate the performance of nested DSHEM and compare it with SHEM and Random.

### 9.3.1. Execution Parameters

Three main parameters are used to tune up nested DSHEM, namely *-nctype*, *-maxvwtm* and *-dshem_p2*. Based on the previous experiments, the two parameters *-dshem_p1* and *-dshem_p3* have a fix value of 100. The values chosen for the second set of experiments are presented in Table 9.2. This particular set produces 882 different combinations of values, giving a detailed view of the performance of nested DSHEM.

**Table 9.2: Nested DSHEM parameters for the second set of small synthetic graphs.**

| *-nctype* | *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|---|
| dshem+shem<br>shem+dshem | 140 to 160 | 100 | 90 to 110 | 100 |

### 9.3.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. Based on the results of the first set of experiments, the effect of the multiplier *-maxvwtm* is evaluated with greater detail, as well as the percentage *-dshem_p2*. Percentages *-dshem_p1* and *-dshem_p3* are set to 100 as they do not influence the outcome.

Again, the robustness of nested DSHEM is also evaluated with different degrees of irregularity introduced to the synthetic graphs. The refinement and its influence on nested DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by nested DSHEM. The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

#### Multiplier *-maxvwtm*

A closer analysis of the multiplier *-maxvwtm* brings a different scenario of that from the first set of experiments. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a balanced partition does not necessarily mean a smaller edge cut or reduction in communication volume; it only means that the subdomains are more equal in size.

Figure 9.17 and Figure 9.18 show that nested DSHEM executed as DSHEM+SHEM produces better results when the total communication volume is evaluated; being the 3D square graph (*sm3d100p*) and the 3D triangular square graph (*tsm3d100p*) with the highest improvements and the 2D square graph (*sm2d100p*) with some improvement. Also, the benefit of using DSHEM+SHEM is more stable than that of SHEM+DSHEM. These results suggest that the multiplier *-maxvwtm* plays some role in the partitioning process, but only after trial and error it would be possible to adapt it to specific needs; more precisely with SHEM+DSHEM.

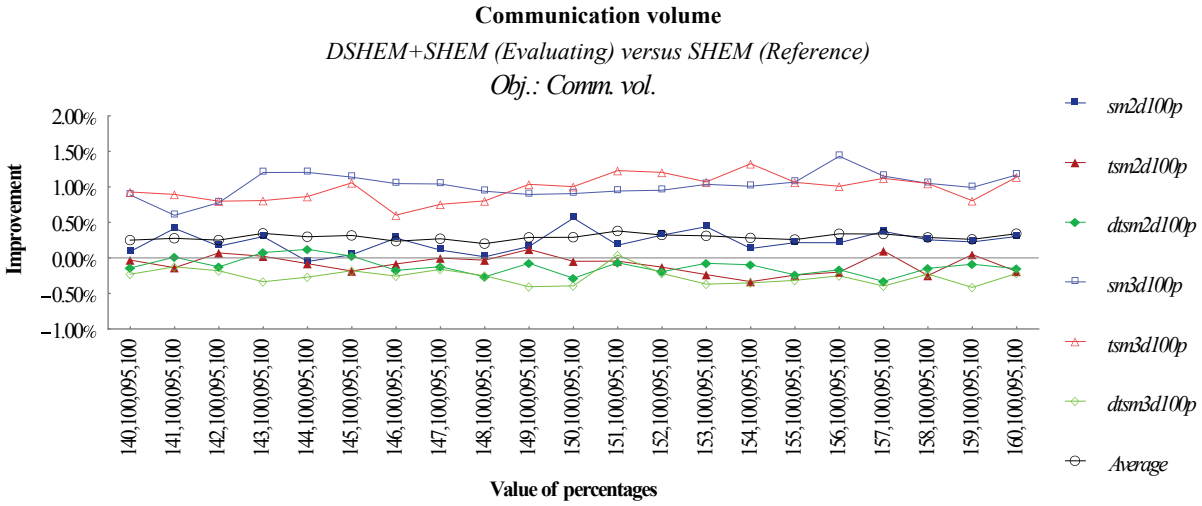


**Figure 9.17. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

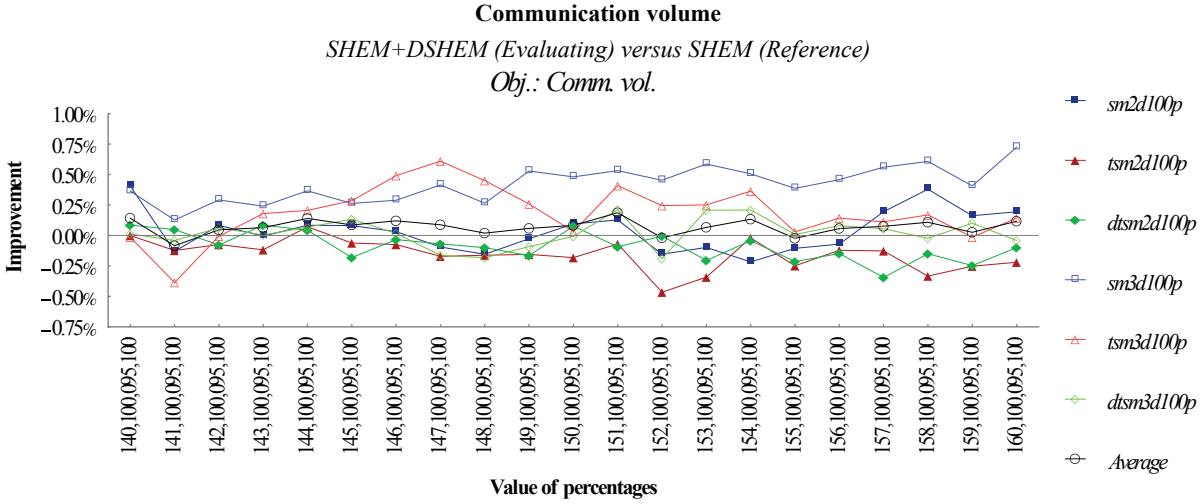


**Figure 9.18. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

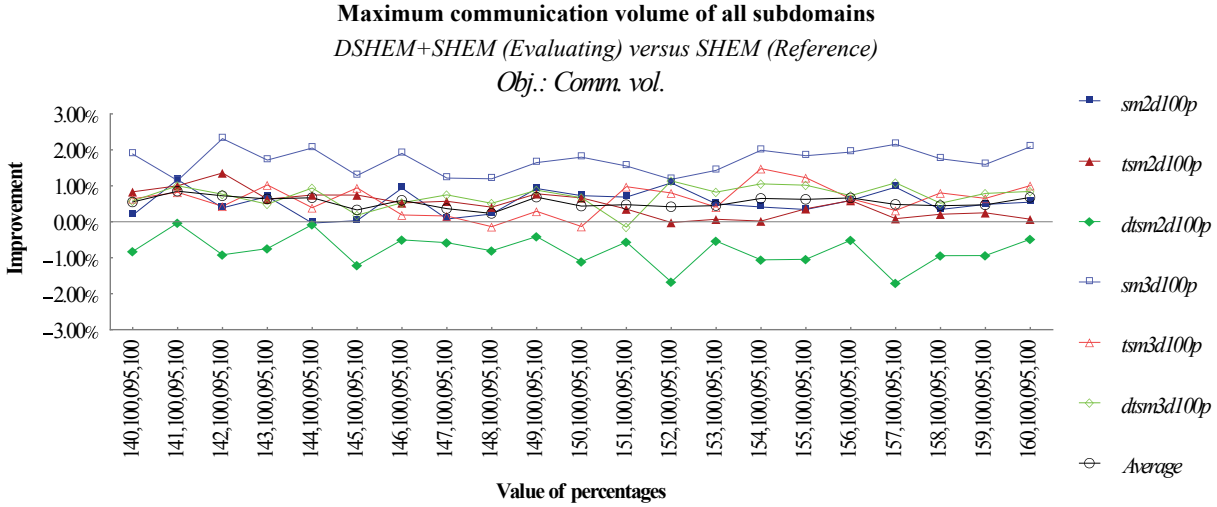


**Figure 9.19. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

With respect to the maximum communication volume of all subdomains, Figure 9.19 and Figure 9.20 confirm that results produced by DSHEM+SHEM are more stable and predictable. Again the 3D triangular square graph (*tsm3d100p*) presents the highest improvements. The irregularity of the results produced by SHEM+DSHEM could be attributed to the fact that once the graph has been contracted by SHEM, the initial partition with DSHEM values, and the subsequent refinement, could not improve the results due to the restrictions posed by the coarsest graph.

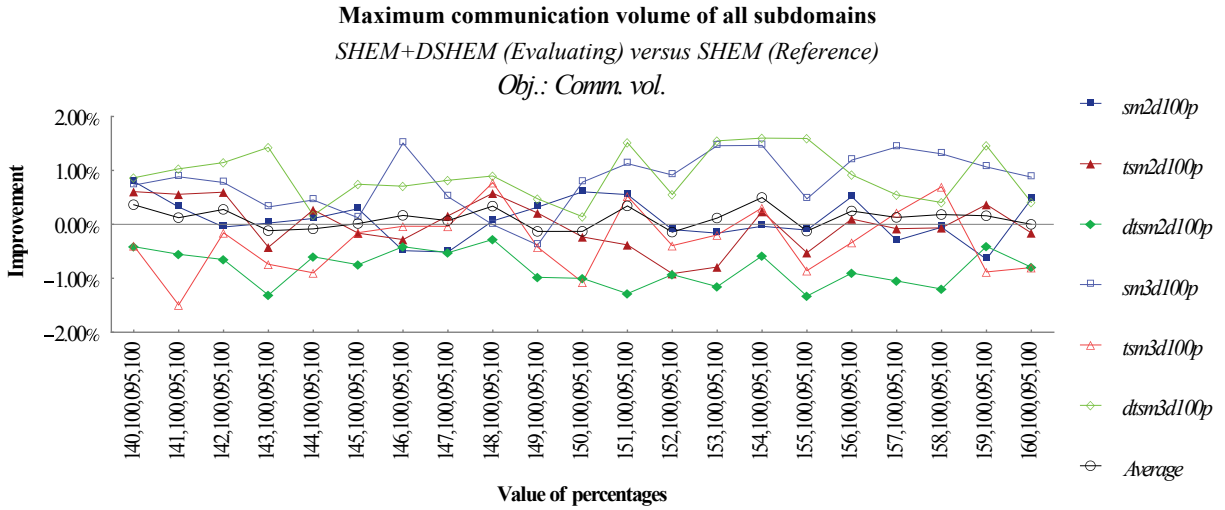


**Figure 9.20. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

The multiplier *-maxvwtm* may improve the balance of the initial partition when its value is low, however, it is not possible to determine when and how it improves the quality of the final partition as the refinement process plays an important role in this regard.

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from a deeper analysis as previous results suggest they do not play a role at all in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in nested DSHEM and improve the partition.

The effect of percentage *-dshem_p2* on the total communication volume is evident from the results presented in Figure 9.21, when nested DSHEM is executed as DSHEM+SHEM. Three types of graphs have a clear improvement when the percentage value is smaller than 100, the other three produce better results when the value is 100 or higher. When nested DSHEM is executed as SHEM+DSHEM, as depicted in Figure 9.22, the pattern seen before is replaced by a more constant behavior over all the values used for the percentage. The quality of the partitions is also degraded over all types of graphs.

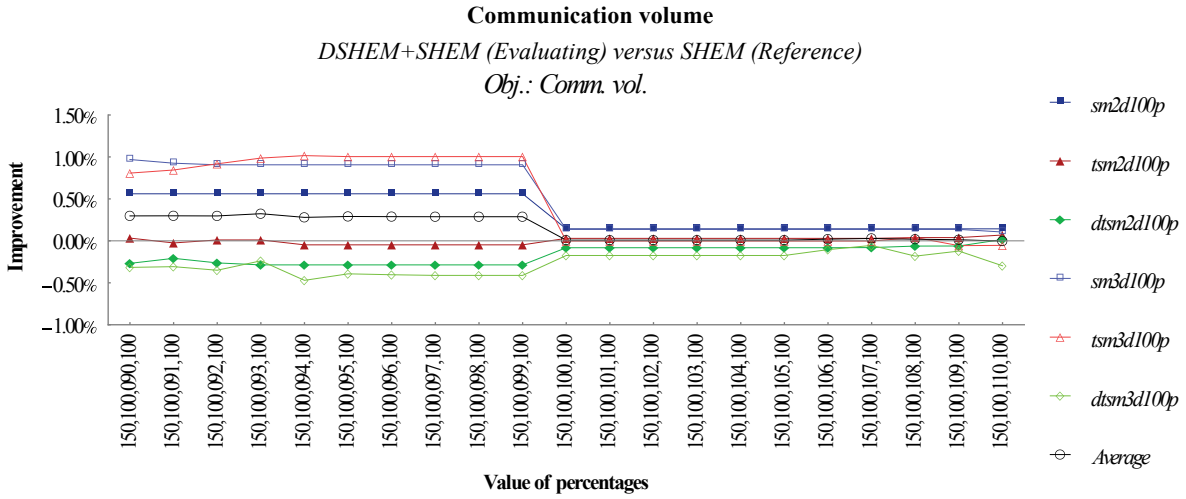


**Figure 9.21. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

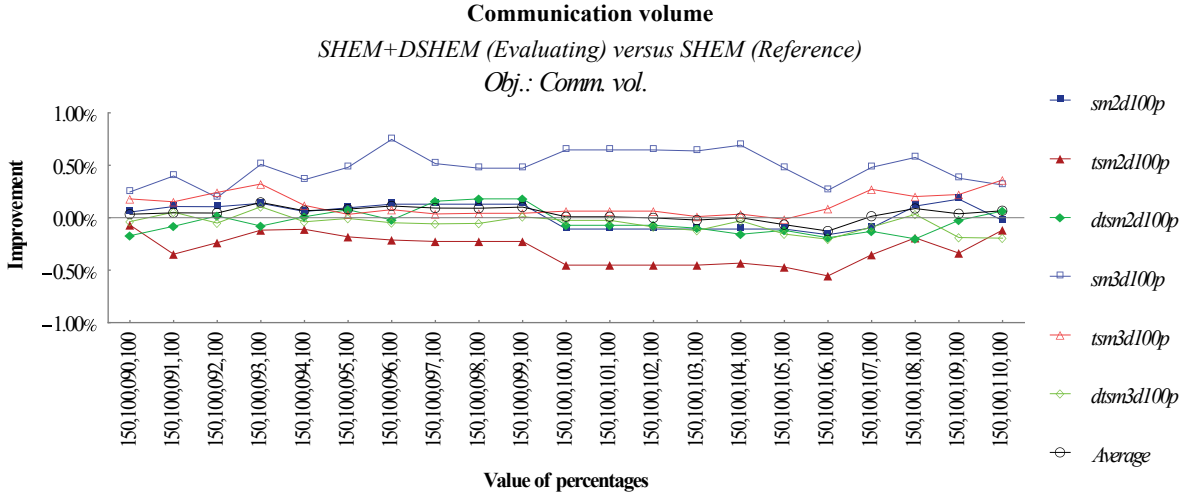


**Figure 9.22. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

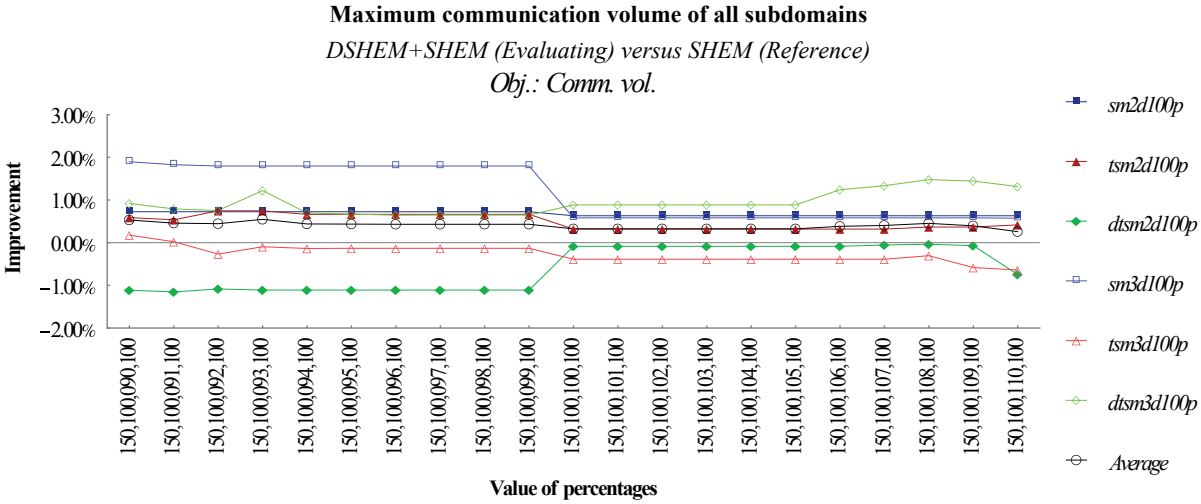


**Figure 9.23. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Again, a similar behavior can be appreciated when the maximum communication volume of all subdomains is evaluated; Figure 9.23 and Figure 9.24 confirm that nested DSHEM executed as DSHEM+SHEM produces a discernible pattern in terms of quality of the final partition. Though, in this case some types of graphs, such as the 2D square graph (*sm2d100p*), remain constant over all values of the percentage *-dshem_p2*.

When nested DSHEM is executed as SHEM+DSHEM the results are more predictable, see Figure 9.24. The graphs that benefit from this type of execution are the 2D and 3D square graph (*sm2d100p* and *sm3d100p* respectively) and the 3D dense triangular square graph (*dtsm3d100p*).

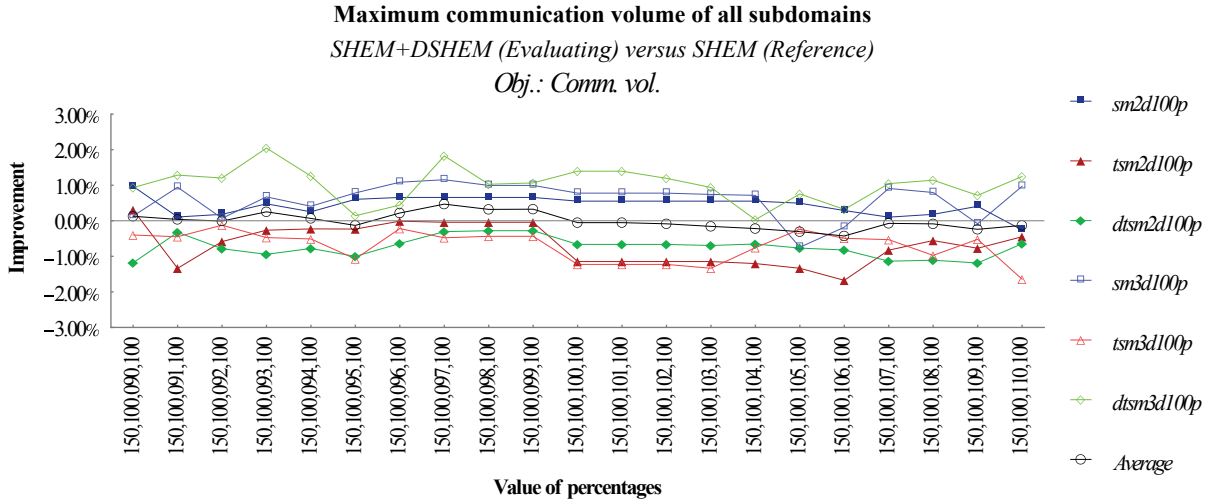


**Figure 9.24. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

The influence of percentage *-dshem_p2* on the quality of the partitions is more evident with DSHEM+SHEM. In this case DSHEM has a better opportunity to have an impact and not being eclipsed by SHEM.

## Graph Irregularity

The performance of nested DSHEM is also studied with irregular graphs. Sizes $a$ to $d$, in Figure 9.25, Figure 9.26, Figure 9.27 and Figure 9.28, represent the four biggest 2D and all 3D graphs of the set.

Figure 9.25 and Figure 9.26 confirm the results obtained with the first set of experiments. With the total communication volume in mind, both types of execution of nested DSHEM are not proportionally affected by the degree of irregularity introduced to the synthetic graphs. It is obvious that the irregularity impacts the quality of the final partition, but it is dependent on the specific case of graph type, size and irregularity. A higher degree of irregularity in the graph does not mean a higher improvement or degradation of the quality of the final partition.

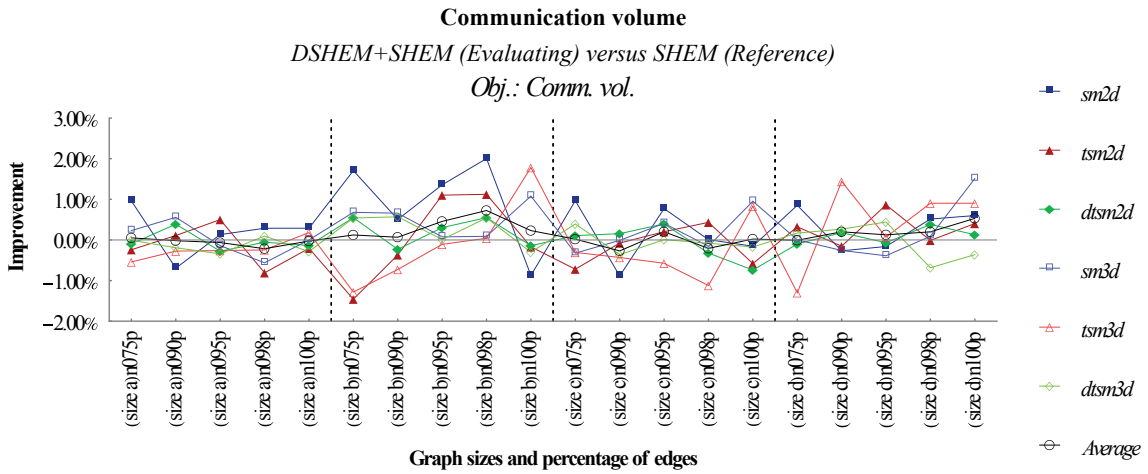


**Figure 9.25. DSHEM+SHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

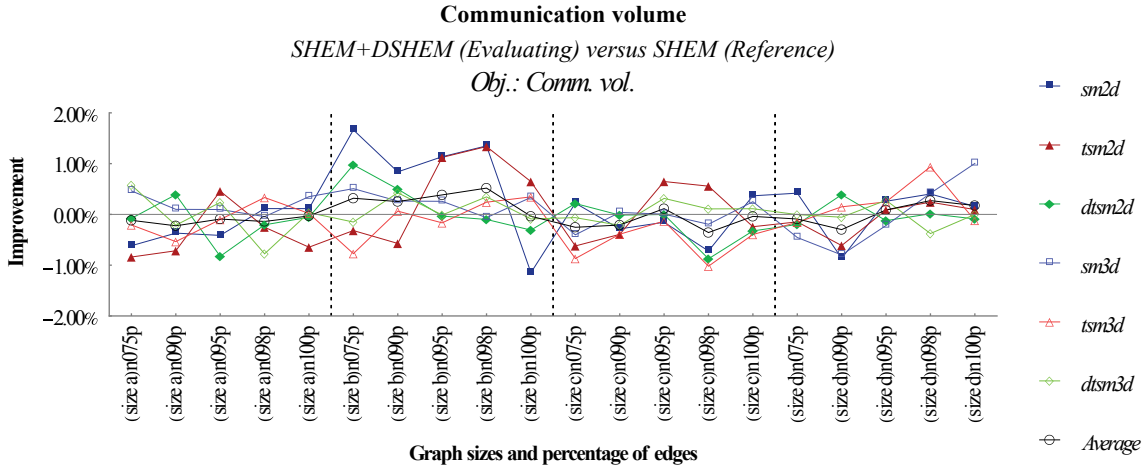


**Figure 9.26. SHEM+DSHEM vs. SHEM: effect of irregularity on the communication volume with synthetic graphs and communication volume as partitioning objective.**

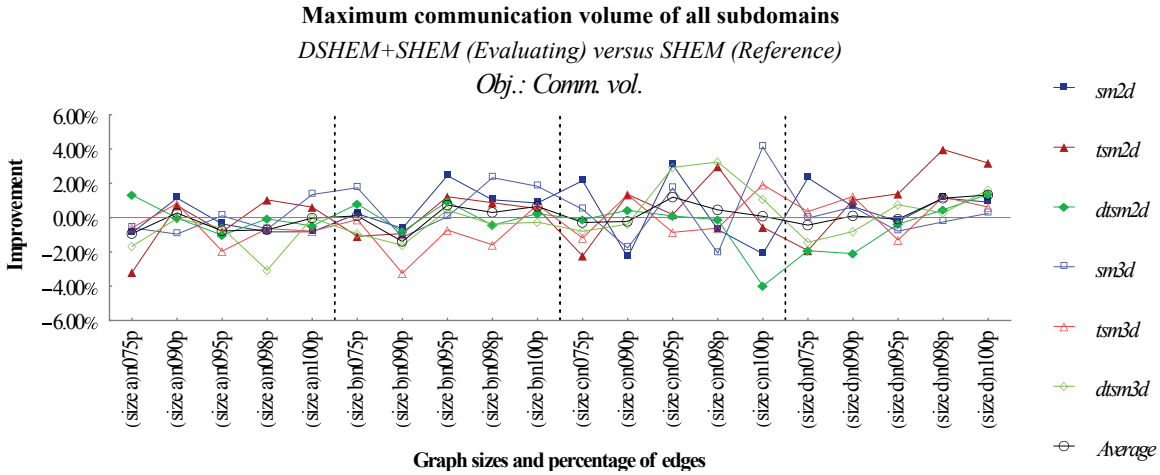


**Figure 9.27. DSHEM+SHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

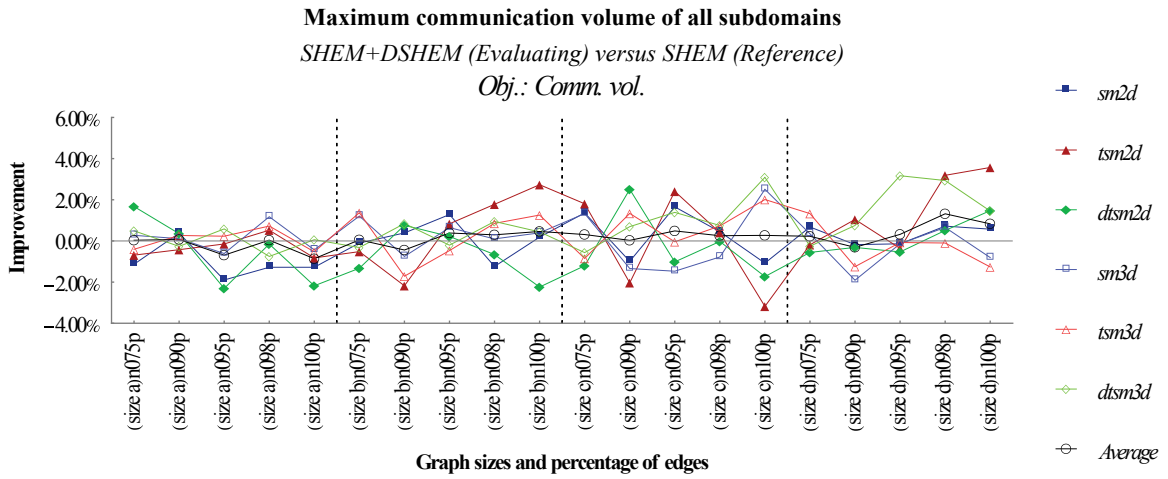


**Figure 9.28. SHEM+DSHEM vs. SHEM: effect of irregularity on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

A similar scenario is shown by Figure 9.27 and Figure 9.28 when the maximum communication volume of all subdomains is evaluated; the role of the irregularity does not affect the quality of the results in a proportional amount.

## Refinement

To analyze the effect of the refinement process on the partitions produced by nested DSHEM, METIS is executed without it; without the refinement process no objective is optimized.

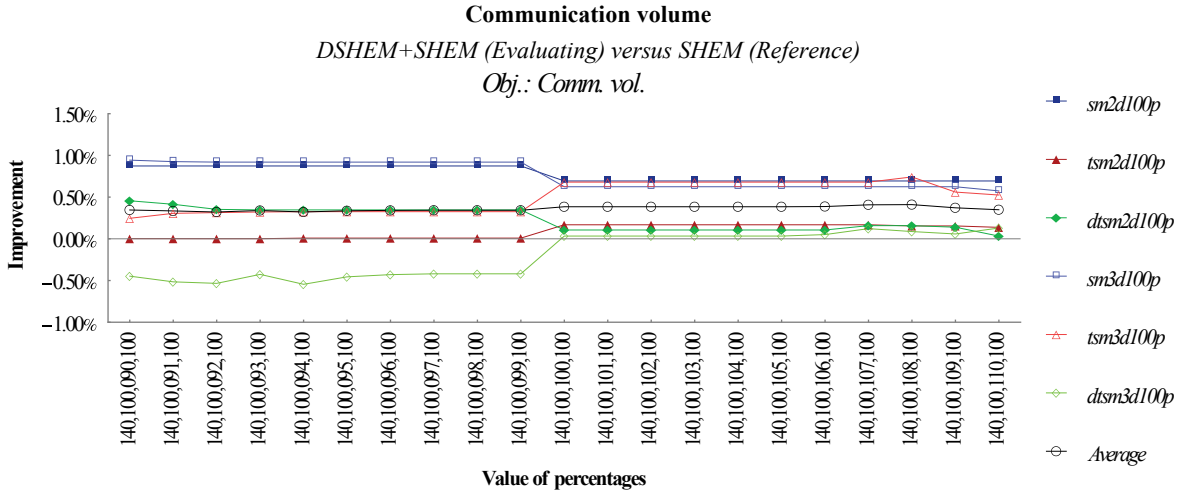


**Figure 9.29. DSHEM+SHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

Similar to the previous results, the percentage *-dshem_p2* heavily impact the results. It is evident when nested DSHEM is executed as DSHEM+SHEM, as depicted in Figure 9.29 and Figure 9.31. SHEM takes over with the type of execution being SHEM+DSHEM as no refinement process is performed, see Figure 9.30 and Figure 9.32.

When nested DSHEM is executed, the algorithm used for the coarsening process seems to have the greatest impact in the final partition. This is clear when the refinement process is taken out of the equation, as the real effects of the coarsening process become visible and not hidden when a partitioning objective is optimized.

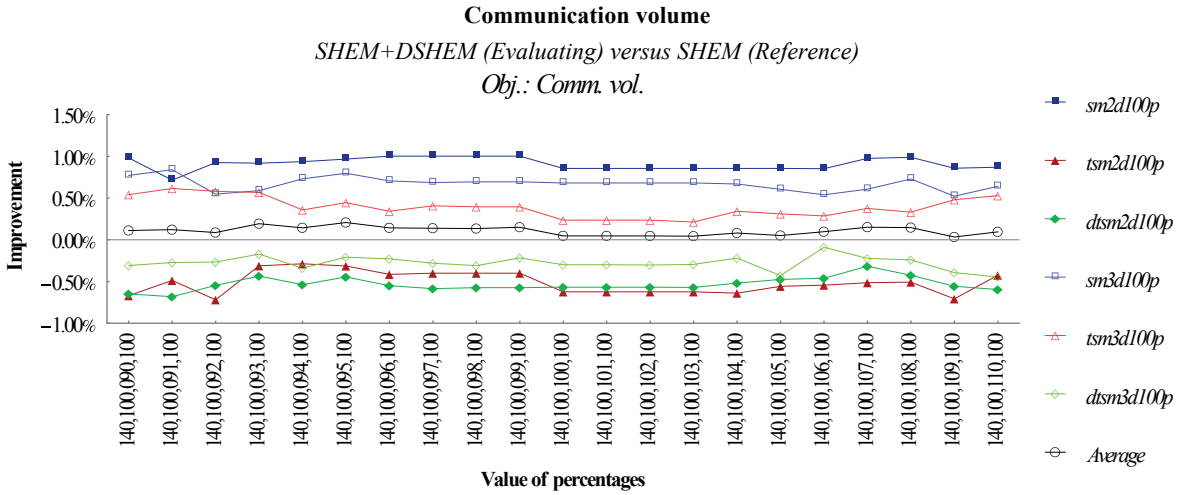


**Figure 9.30. SHEM+DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

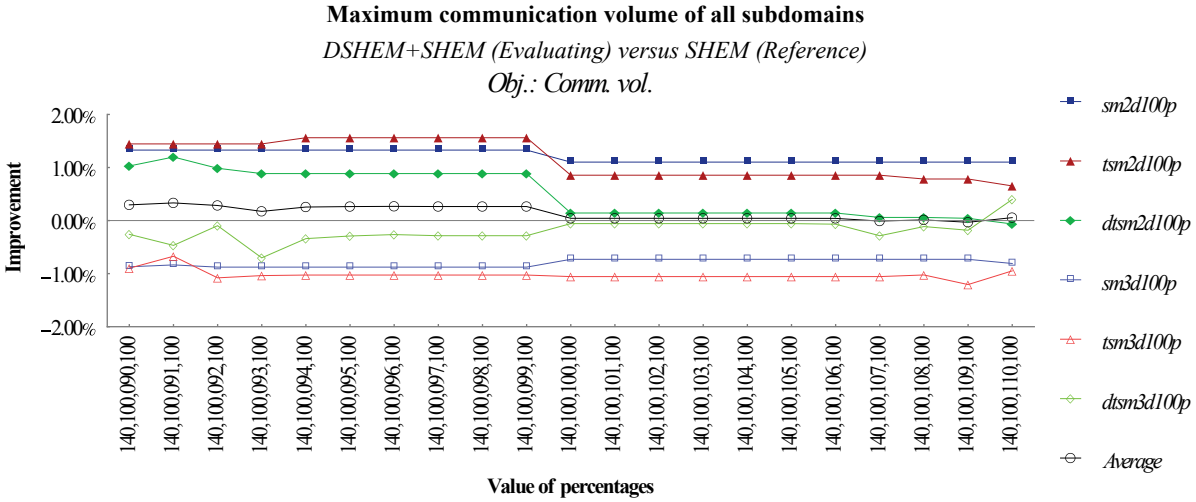


**Figure 9.31. DSHEM+SHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

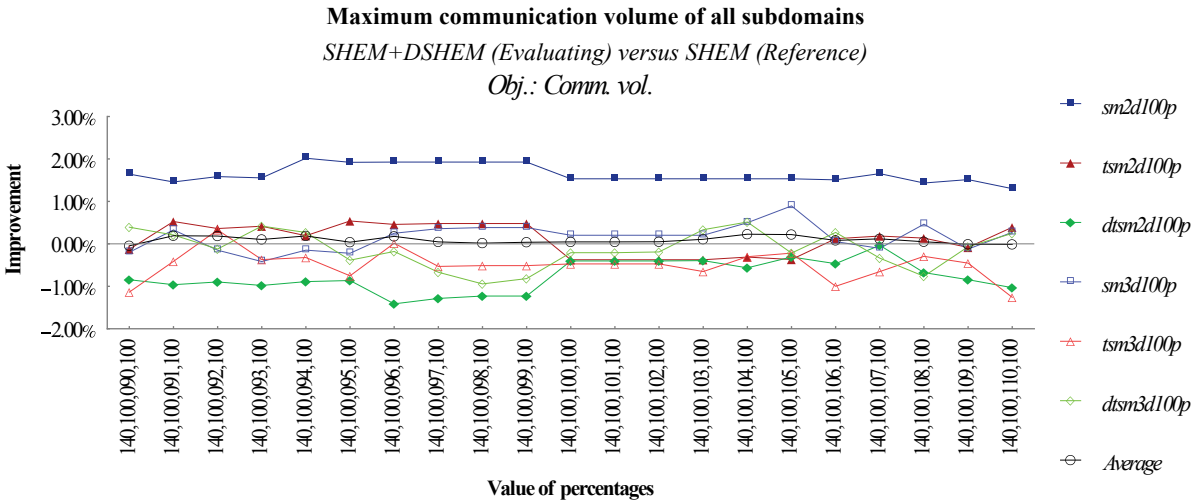


**Figure 9.32. SHEM+DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

#### Execution Time

The graphs for these experiments are small enough to make virtually impossible to evaluate the impact of nested DSHEM on the execution time. The majority of the running times does not even reach one second.

## 9.4. Experiments on Medium Size Synthetic Graphs

This particular set of experiments uses the graphs presented in Table 7.8 of Chapter 7. It is a set of medium size synthetic graphs which are designed to evaluate the performance of nested DSHEM with larger graphs and compare it with SHEM and Random.

### 9.4.1. Execution Parameters

Three main parameters are used to tune up nested DSHEM, namely *-nctype*, *-maxvwtm* and *-dshem_p2*. Based on the previous sets of experiments, the two parameters *-dshem_p1* and *-dshem_p3* have a fix value of 100. The values chosen for the third set of experiments are presented in Table 9.3. This particular set produces 882 different combinations of values, giving a detailed view of the performance of nested DSHEM.

**Table 9.3: Nested DSHEM parameters for the set of medium size synthetic graphs.**

| *-nctype* | *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|---|
| dshem+shem<br>shem+dshem | 140 to 160 | 100 | 90 to 110 | 100 |

### 9.4.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. Based on the results of the first two sets of experiments, the effect of the multiplier *-maxvwtm* is evaluated with detail, as well as the percentage *-dshem_p2*. Percentages *-dshem_p1* and *-dshem_p3* are set to 100 as they do not influence the outcome. The refinement and its influence on nested DSHEM are also studied. Finally, the execution time is also examined to estimate the degradation, if any, brought by nested DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

#### Multiplier *-maxvwtm*

A close analysis of the multiplier *-maxvwtm* is presented with this set of experiments, similar to the

second set. From the results in Figure 9.33 and Figure 9.34, it is evident that nested DSHEM, executed as SHEM+DSHEM, brings better results for the edge cut. Nonetheless, the multiplier *-maxvwtm* does not bring a conclusive impact on the results when its value varies from 140 to 160.

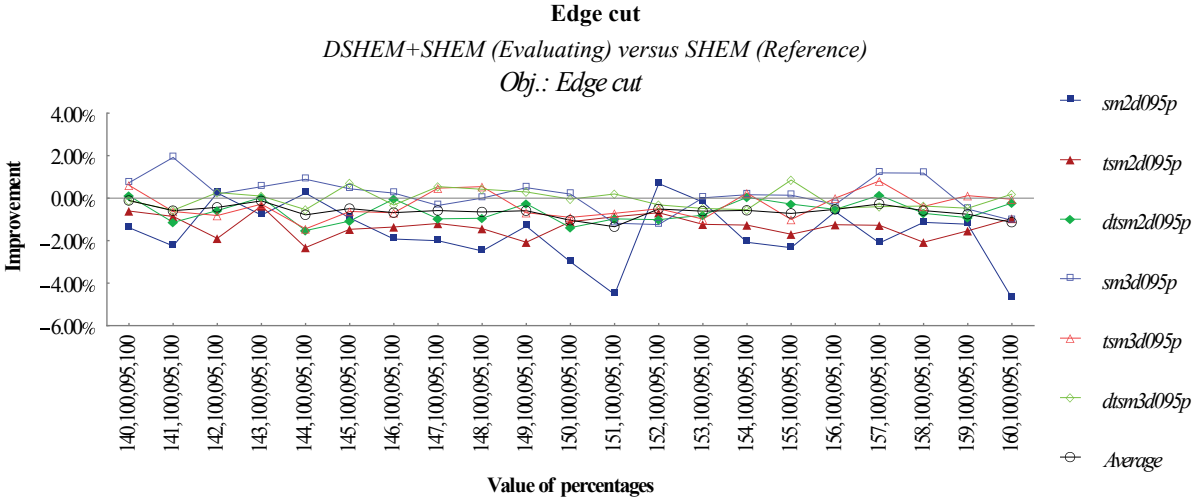


**Figure 9.33. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

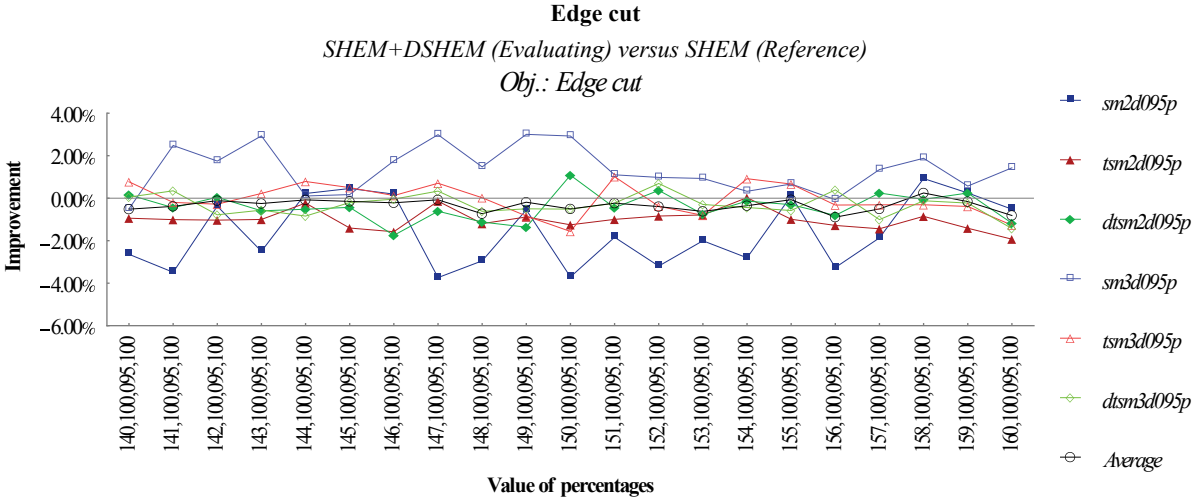


**Figure 9.34. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

With respect to the communication volume, the results seem to be more stable when nested DSHEM is executed as SHEM+DSHEM, as seen in Figure 9.35 and Figure 9.36. The multiplier *-maxvwtm* does not affect the results in a discernible pattern. In both cases, nested DSHEM provides improvements to the 2D square graph (*sm2d095p*); the graphs with triangular geometries are adversely affected.

The maximum communication volume of all subdomains, Figure 9.37 and Figure 9.38, is different in this case. The results seem random and without a clear advantage or disadvantage with DSHEM+SHEM or SHEM+DSHEM and different values for the multiplier. The only difference is that SHEM+DSHEM is more stable when the maximum communication volume of all subdomains is evaluated.

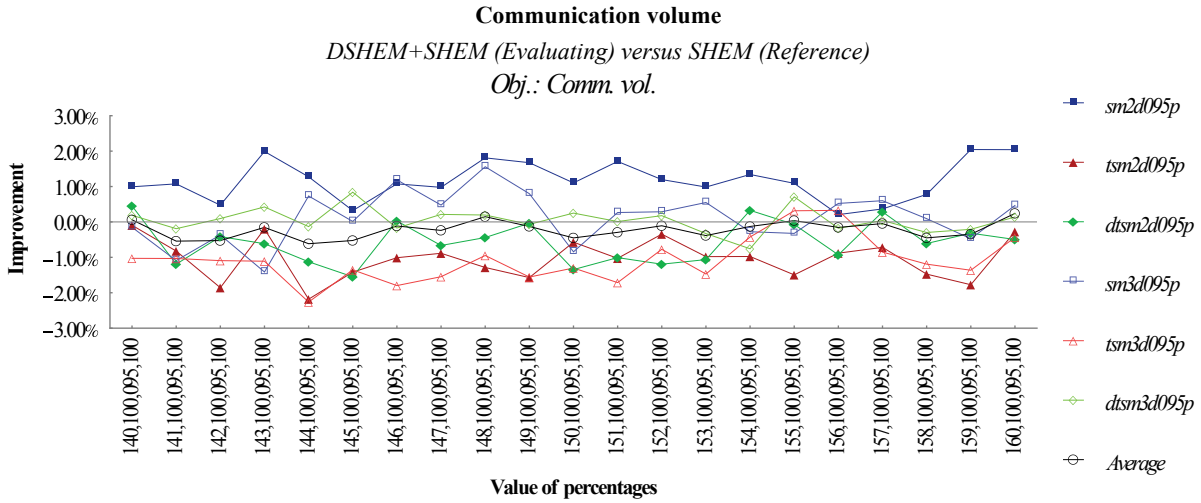


**Figure 9.35. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

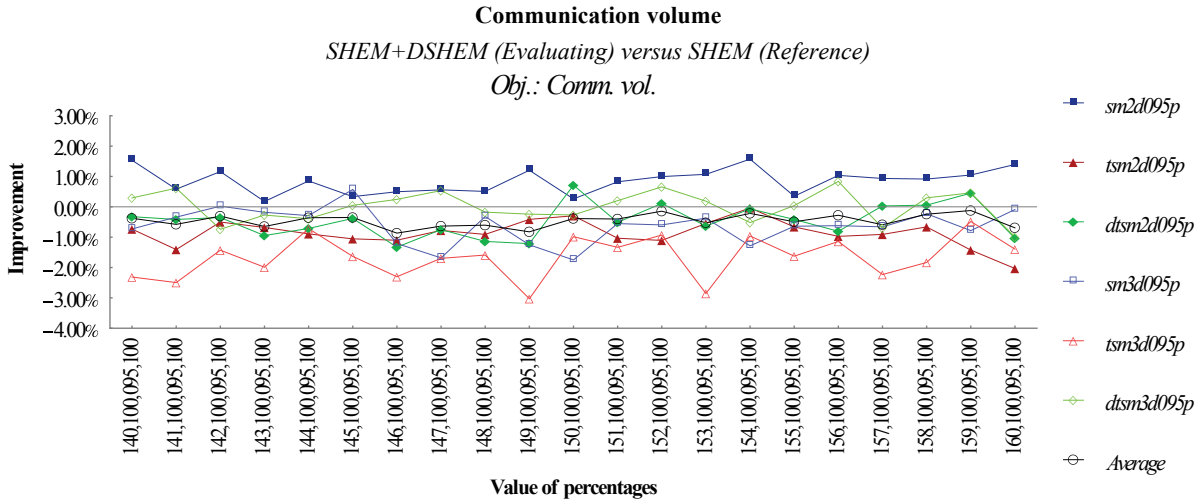


**Figure 9.36. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

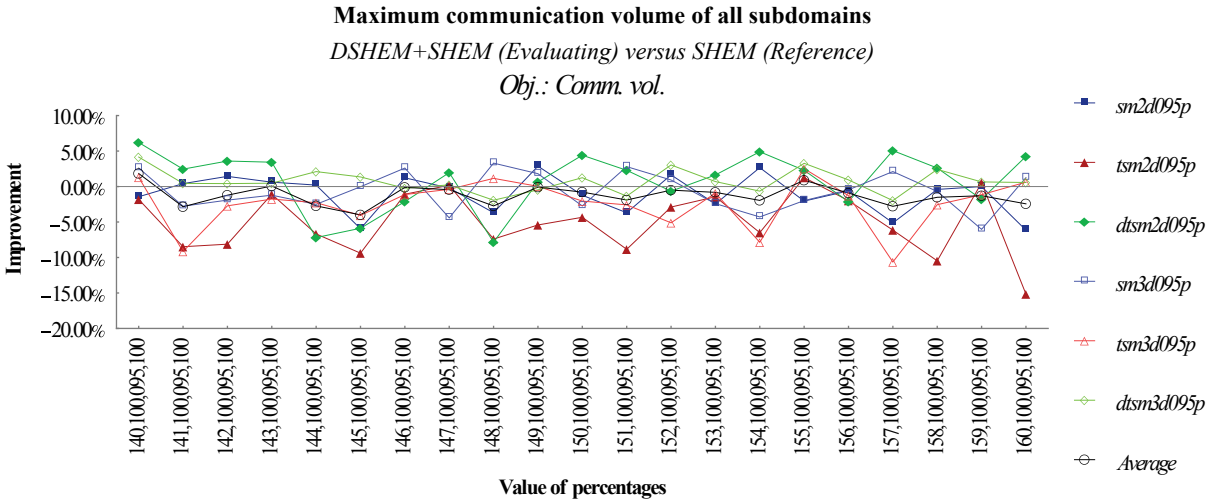


**Figure 9.37. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

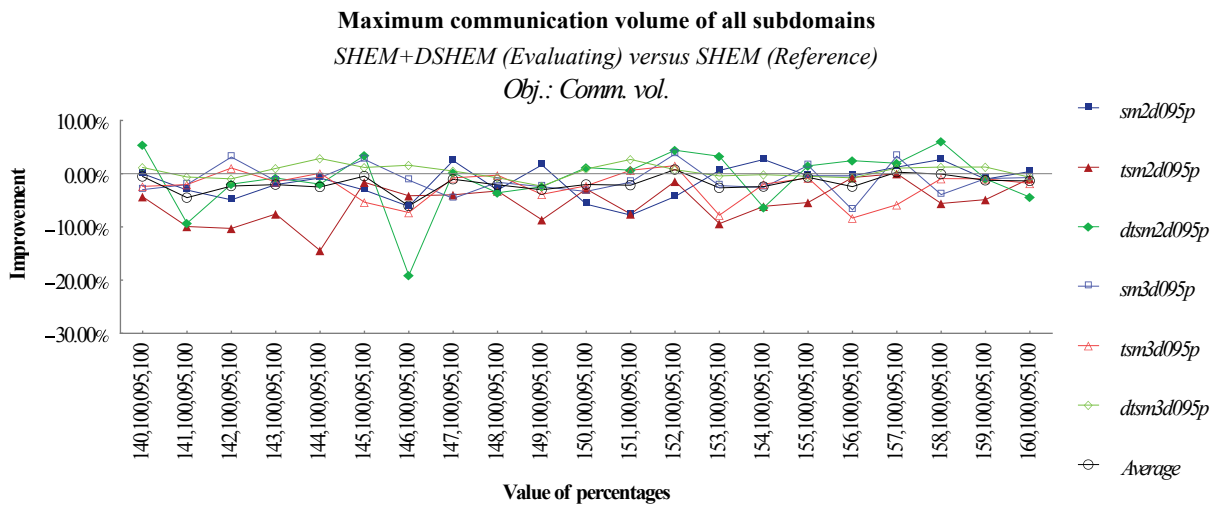


**Figure 9.38. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

## Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from a deeper analysis as previous results suggest they do not play a role at all in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in nested DSHEM and improve the partition.

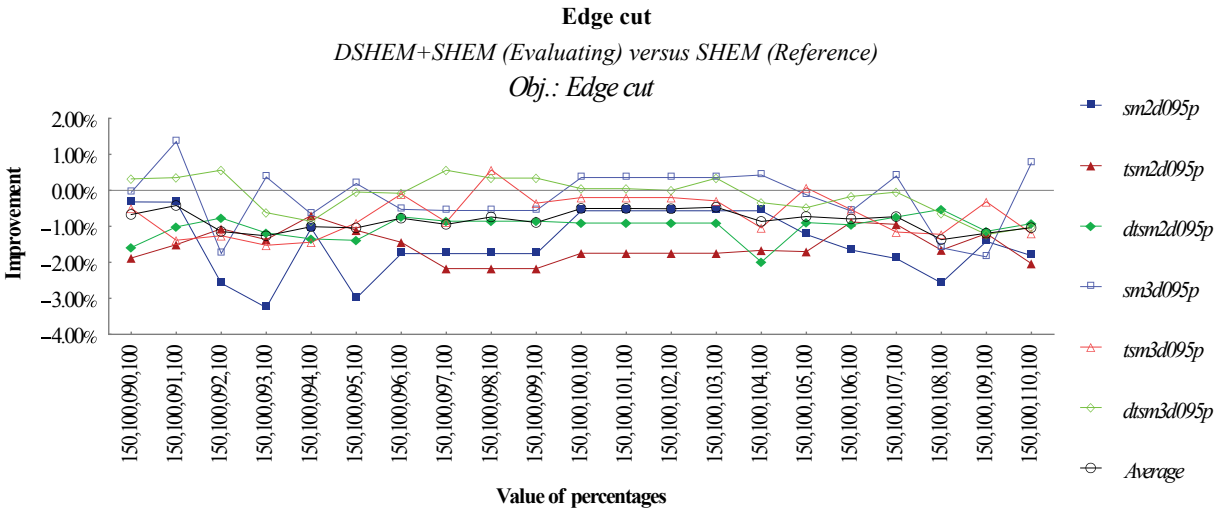


**Figure 9.39. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

As depicted in Figure 9.39, when nested DSHEM is executed as DSHEM+SHEM and the edge cut is evaluated, with the irregularity introduced to the synthetic graphs, the percentage *-dshem_p2* has a modified pattern. With regular graphs, the behavior of the percentage is predictable, with an inflection point in 100. The change, when *dshem_p2* is set to 100, is still evident and the influence of the percentage remains stable when it is close to 100. It becomes more scattered as it is further away from the inflection point. In the case of SHEM+DSHEM, Figure 9.40 shows a more irregular behavior, following the results of previous sets of experiments.

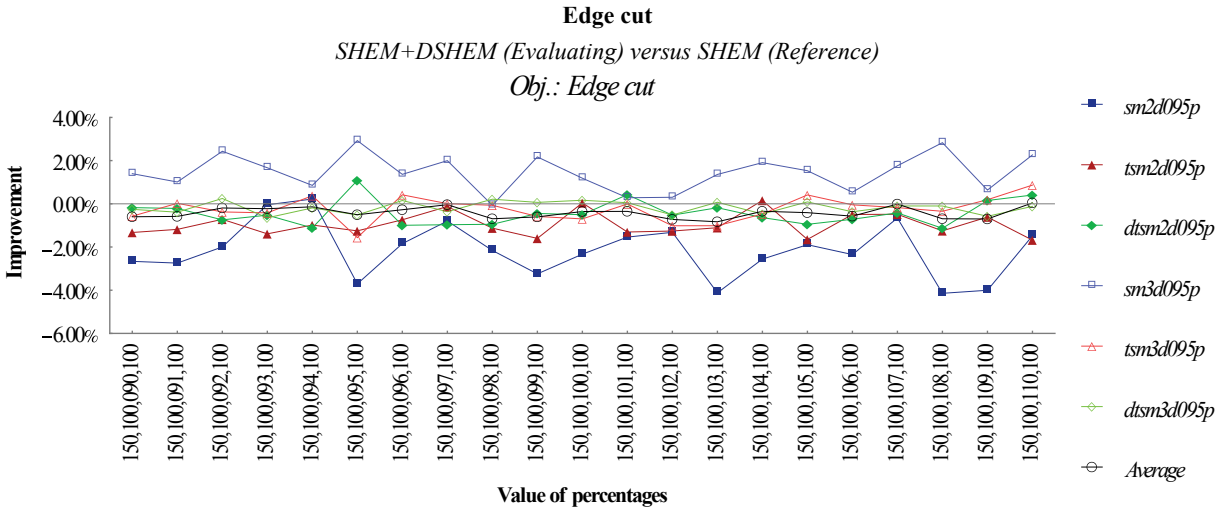


**Figure 9.40. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the edge cut with synthetic graphs and edge cut as partitioning objective.**

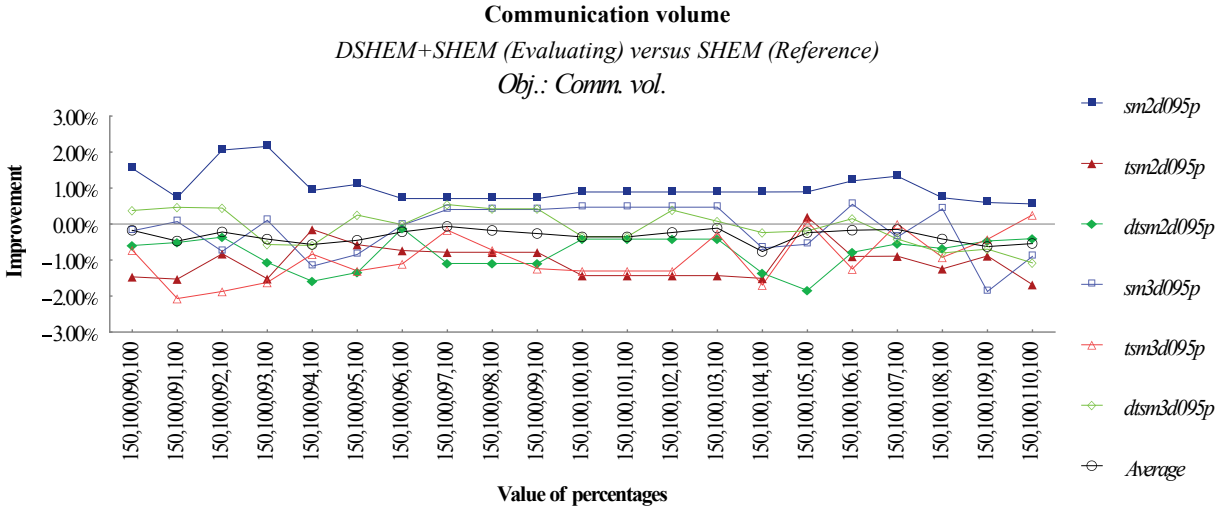


**Figure 9.41. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

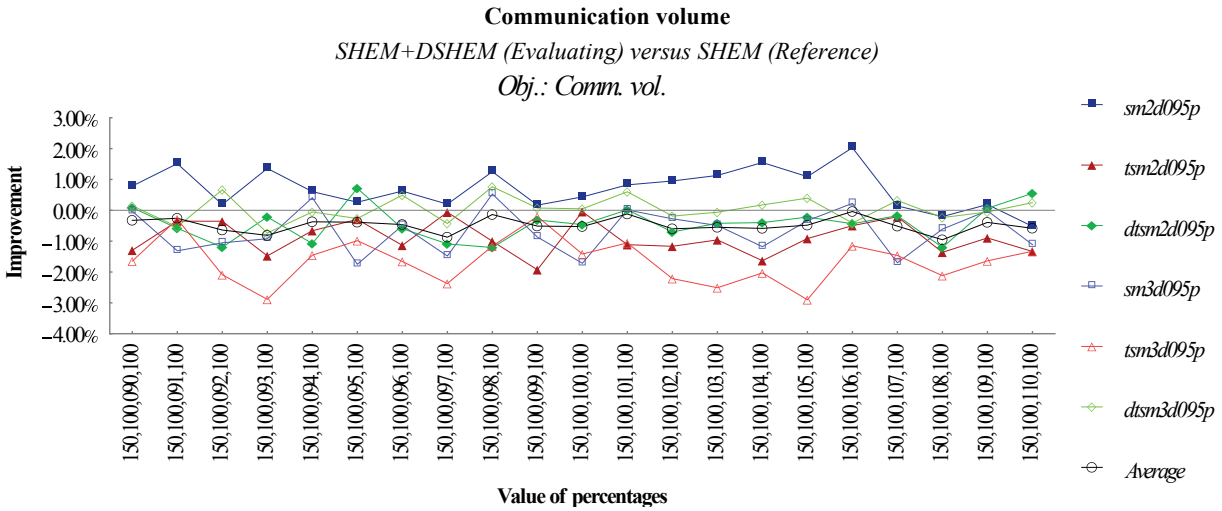


**Figure 9.42. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with synthetic graphs and communication volume as partitioning objective.**

The communication volume is also affected by the irregularity introduced to the graphs. Again, DSHEM+SHEM is more stable and predictable than SHEM+DSHEM; see Figure 9.41 and Figure 9.42. Further away from the value 100 the results are less predictable. Based on the results from previous experiments, it is still possible to see that for some types of graphs, such as the 2D square graph (*sm2d095p*), a lower value for the percentage *dshem_p2* may produce better results when nested DSHEM is executed as DSHEM+SHEM.

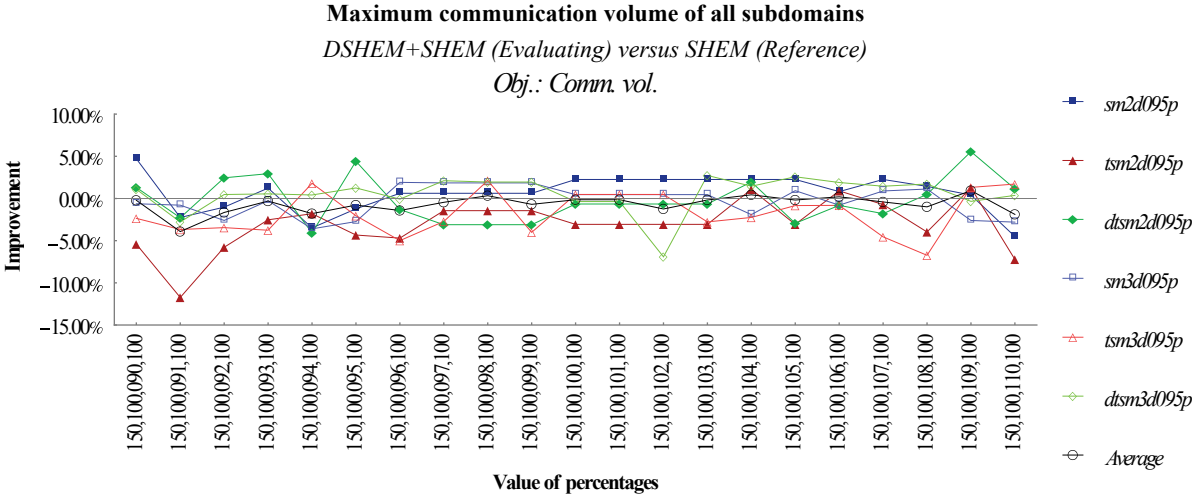


**Figure 9.43. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

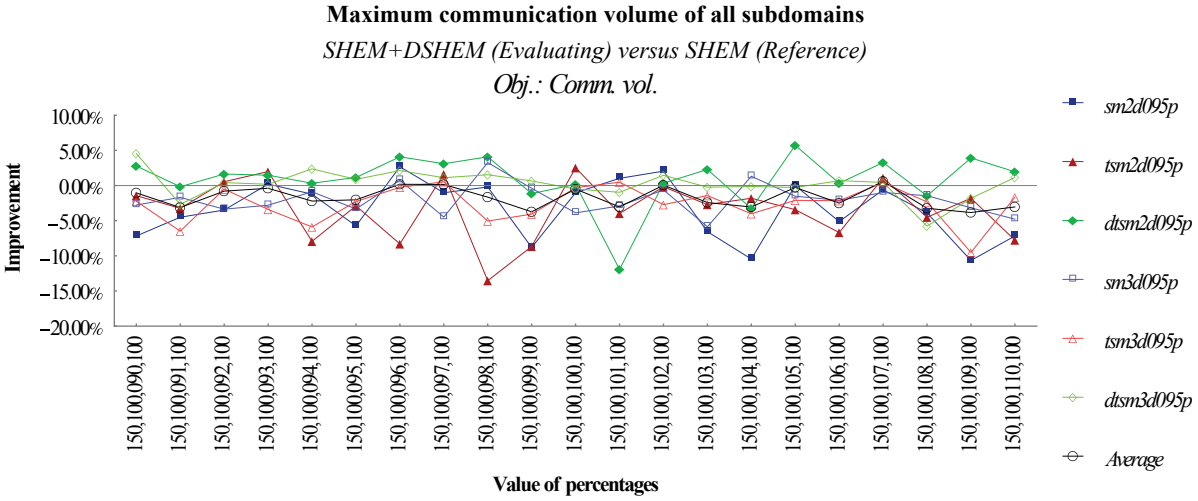


**Figure 9.44. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Figure 9.43 and Figure 9.44 shows that the maximum communication volume of all subdomains is also affected by the irregularity introduced to the synthetic graphs. The original pattern is modified in the same way as described before. This situation leads as to think that values closer to 100 bring a more predictable result with this type of graph.

## Graph Irregularity

The performance of DSHEM is studied with irregular graphs. This set of experiments utilizes only graphs with 95% of the edges.

### Refinement

To analyze the effect of the refinement process on the partitions produced by nested DSHEM, METIS is executed without it. When the refinement process is not performed, no objective is optimized.

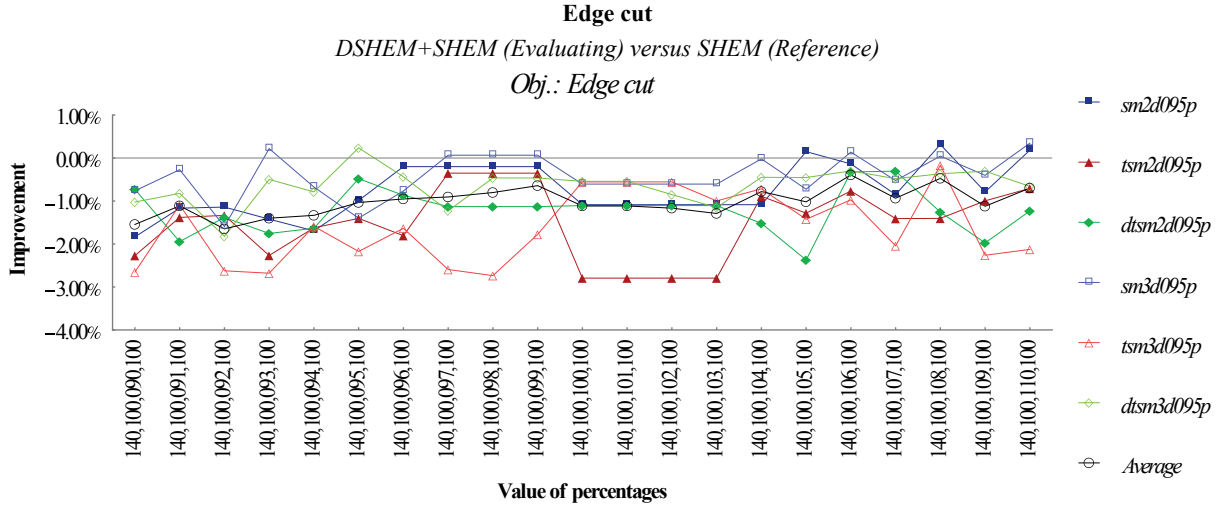


**Figure 9.45. DSHEM+SHEM vs. SHEM: effect of refinement on the edge cut with synthetic graphs and edge cut as partitioning objective.**

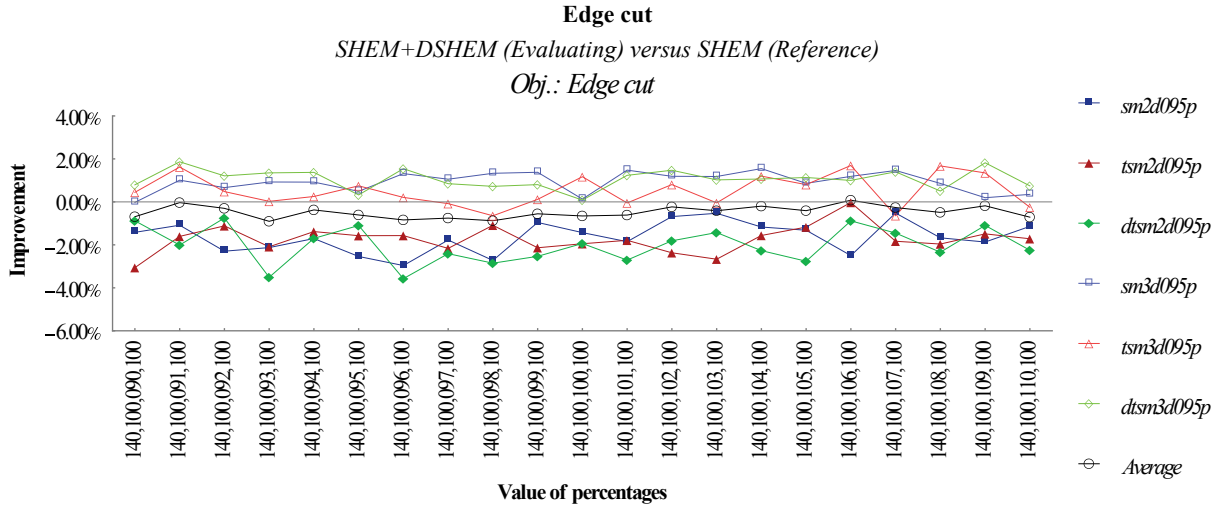


**Figure 9.46. SHEM+DSHEM vs. SHEM: effect of refinement on the edge cut with synthetic graphs and edge cut as partitioning objective.**

When the edge cut is evaluated, Figure 9.45 and Figure 9.46 show that nested DSHEM executed as SHEM+DSHEM produces better results. Again, the irregularity introduced to the synthetic graphs affects the behavior of the percentage *-dshem_p2* when nested DSHEM is executed as DSHEM+SHEM. The 3D square graph (*sm3d095p*) is the type of graph with the best results, followed by the 3D dense triangular square graph (*dtsm3d095p*).

With the total communication volume in mind, nested DSHEM, executed as DSHEM+SHEM, produces better results with the 2D and 3D square graphs (*sm2d095p* and *sm3d095p*). If executed as SHEM+DSHEM, the 2D and 3D square graphs (*sm2d095p* and *sm3d095p*) and the 3D triangular square graph (*tsm3d095p*) show improvements. This is depicted in Figure 9.47 and Figure 9.48 where the irregularity of the graphs affects the final results.

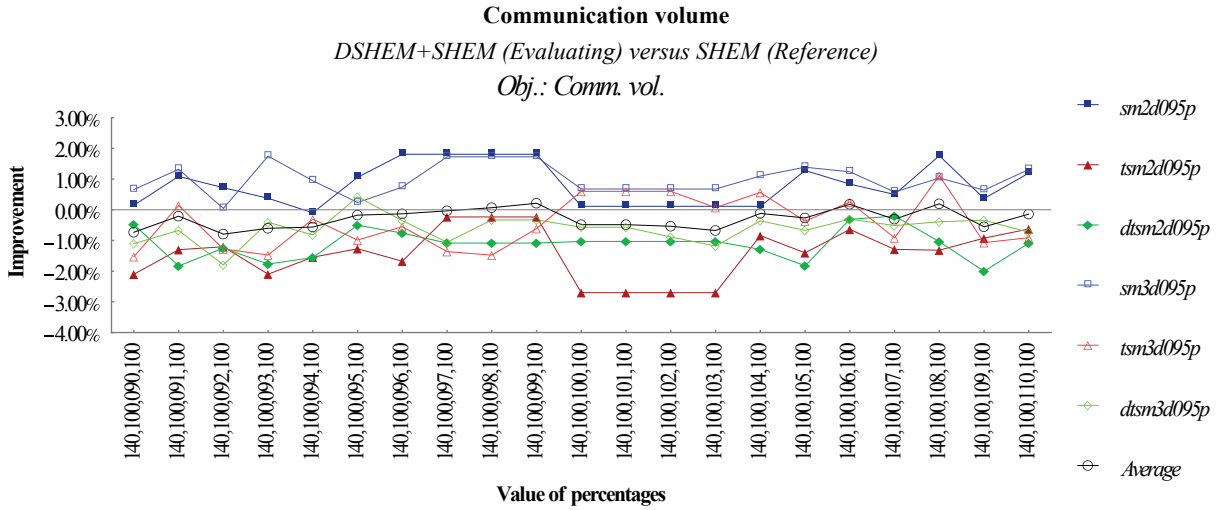


**Figure 9.47. DSHEM+SHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

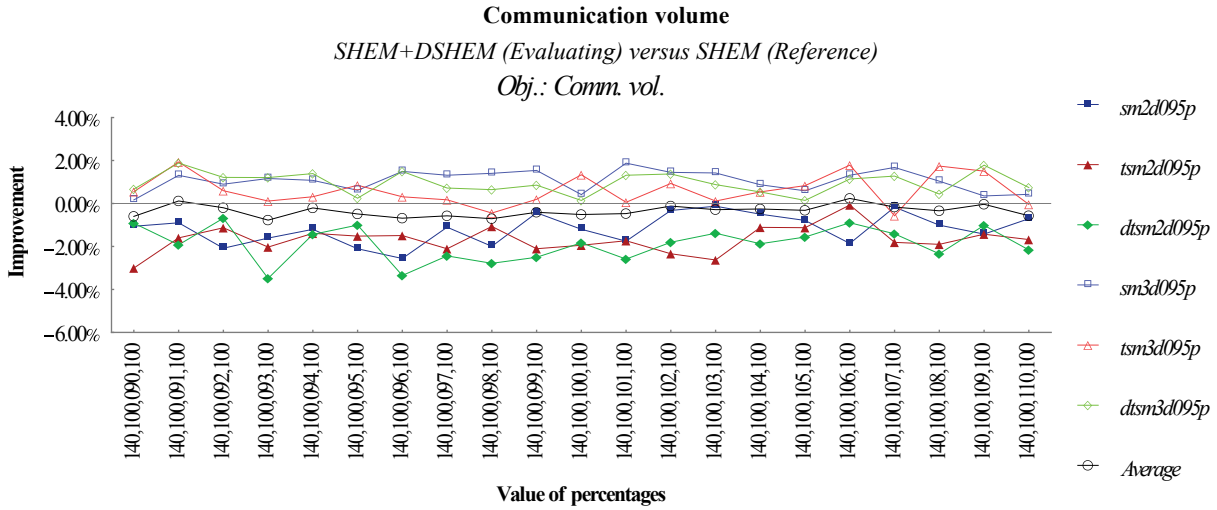


**Figure 9.48. SHEM+DSHEM vs. SHEM: effect of refinement on the communication volume with synthetic graphs and communication volume as partitioning objective.**

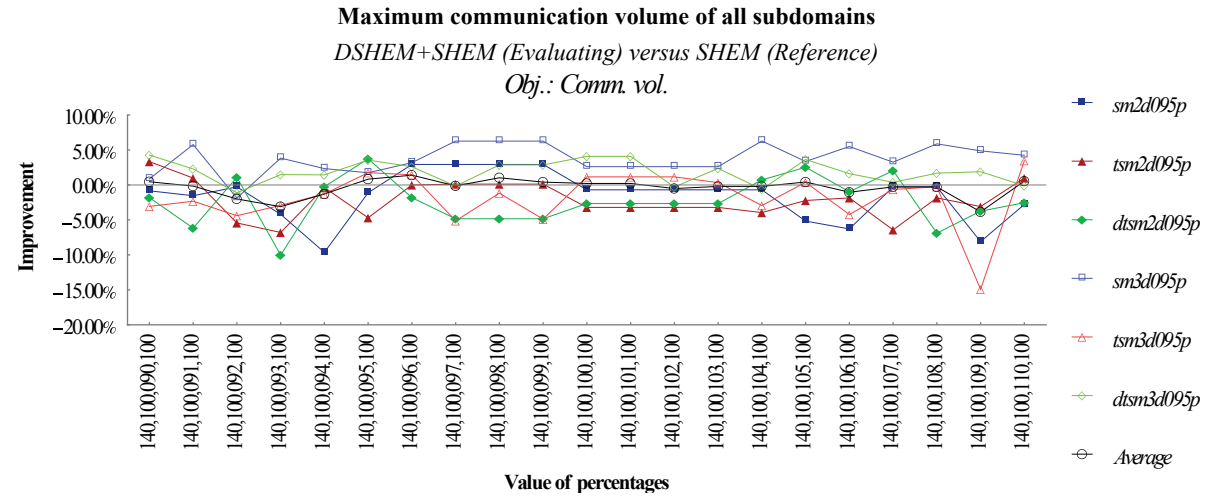


**Figure 9.49. DSHEM+SHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

Figure 9.49 and Figure 9.50 present the effect of the refinement process when the maximum communication volume of all subdomains is evaluated. The graphs with quadrangular geometries show better results in general; this is consistent with the results from previous experiments. However, the degree of irregularity in the graphs creates a different scenario of that from the second set of experiments where the results are more stable and predictable in both cases, DSHEM+SHEM and SHEM+DSHEM.

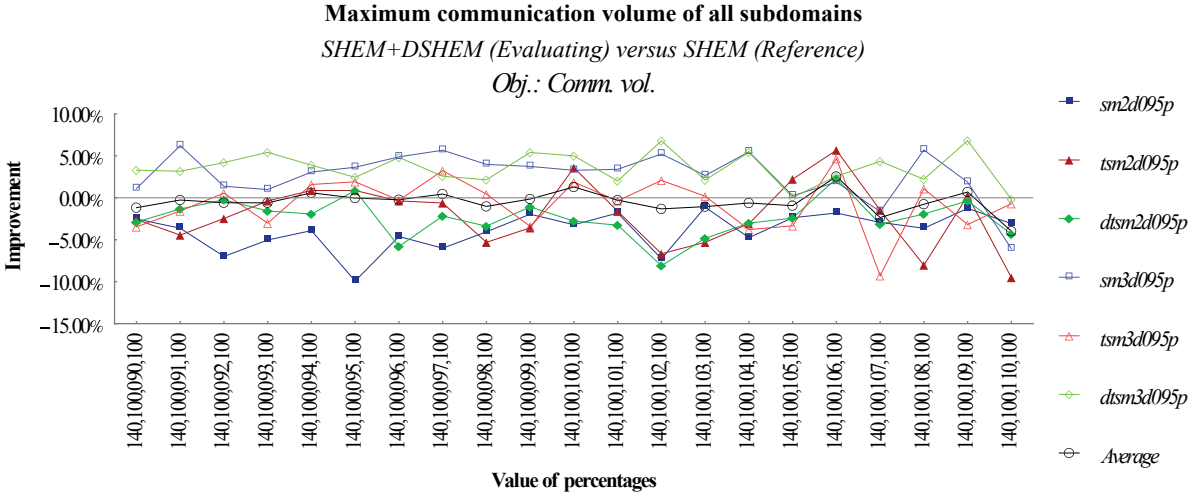


**Figure 9.50. SHEM+DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with synthetic graphs and communication volume as partitioning objective.**

In the case of DSHEM+SHEM, the 3D square graph (*sm3d095p*) and 3D dense triangular square graph (*dtsm3d095p*) have improvement up to 9%; in contrast, the 2D square graph (*sm2d095p*) presents a degradation of up to 10%, see Figure 9.50.

## Execution Time

When METIS is executed without the refinement process, nested DSHEM is slower compared to SHEM due to the extra time spend on the search of the opposite edges. Nevertheless, the difference is minimal, only a couple seconds. In a normal execution, when the partitioning time reaches minutes or hours, the *disadvantage* of DSHEM can be disregarded. The execution time of nested DSHEM executed as DSHEM+SHEM and SHEM+DSHEM are very similar; only the partitioning times of DSHEM+SHEM are presented in here.

Based on the results presented in Figure 9.51 and Figure 9.52, the execution time of nested DSHEM is similar to that of the full implementation of DSHEM. The graphs with quadrangular geometries show a bigger gap when the refinement process optimizes the edge cut or the communication volume. In most of the cases, nested DSHEM is faster than SHEM or Random.

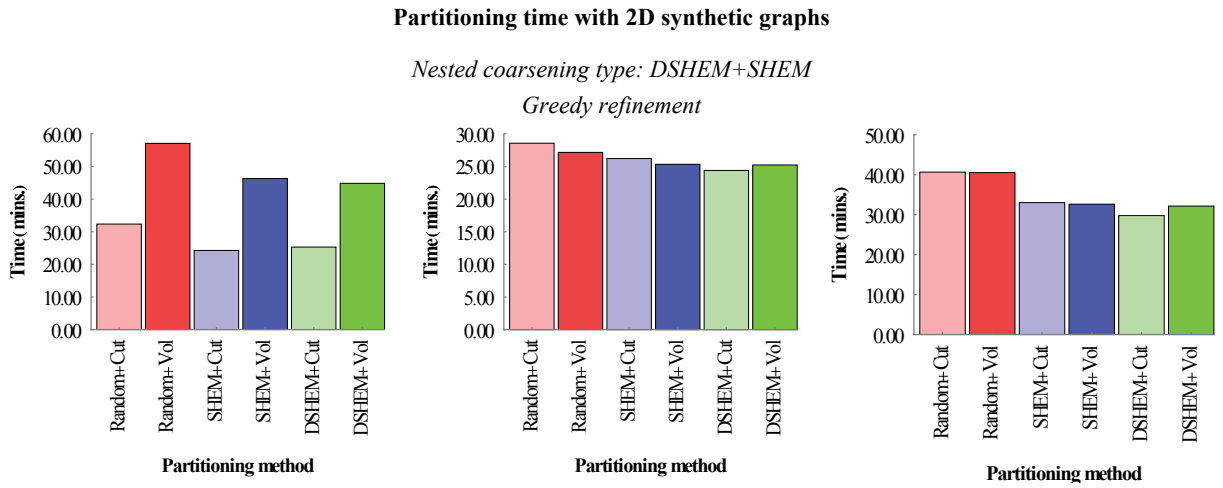


**Figure 9.51. Partitioning time of 2D synthetic graphs with 64 subdomains and greedy refinement. Type of graph: square on the left, triangular square on the middle, dense triangular square on the right. Nested DSHEM is executed as DSHEM+SHEM.**

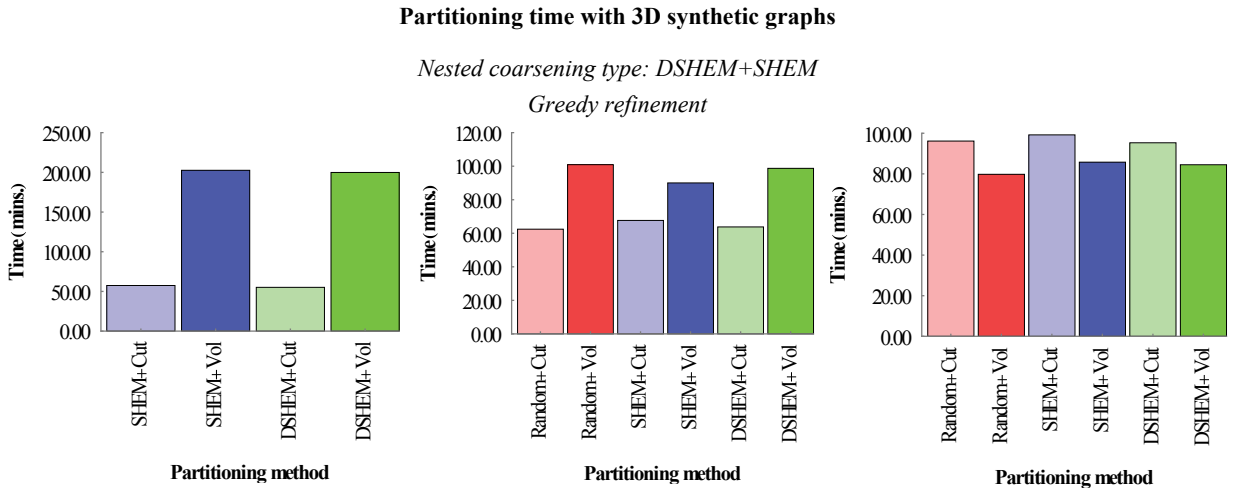


**Figure 9.52. Partitioning time of 3D synthetic graphs with 64 subdomains and greedy refinement. Type of graph: square on the left, triangular square on the middle, dense triangular square on the right. Nested DSHEM is executed as DSHEM+SHEM.**

## 9.5. Experiments on Real Life Graphs

This particular set of experiments uses the graphs presented in Table 7.9 of Chapter 7. It is a set of real life graphs chosen to evaluate the performance of nested DSHEM and compare it with SHEM and Random.

### 9.5.1. Execution Parameters

Five main parameters are used to tune up nested DSHEM, namely *-nctype*, *-maxvwtm*, *-dshem_p1*, *-dshem_p2*, and *-dshem_p3*. The values chosen for the fourth set of experiments are presented in Table 9.4. This particular set produces 3430 different combinations of values, giving a wide view of the performance of nested DSHEM. They are based on the results from previous experimental results, and used to confirm the findings with the synthetic graphs.

**Table 9.4: Nested DSHEM parameters for the set of real life graphs.**

| *-nctype* | *-maxvwtm* | *-dshem_p1* | *-dshem_p2* | *-dshem_p3* |
|---|---|---|---|---|
| dshem+shem<br>shem+dshem | 140 to 160, step 5 | 91 to 109, step 3 | 91 to 109, step 3 | 91 to 109, step 3 |

## 9.5.2. Analysis of Results

The experimental results presented in this section are organized in a manner to understand how the different execution parameters affect the partitions. First, the effect of the multiplier *-maxvwtm* is evaluated. Next, the three percentages *-dshem_p1*, *-dshem_p2*, and *-dshem_p3* are examined to understand their influence. The values used for this particular set of experiments are based on the results of the previous three sets; they help validate the initial findings. The refinement and its influence on nested DSHEM are also studied to confirm the previous results obtained from the synthetic graphs. Finally, the execution time is also examined to estimate the degradation, if any, brought by nested DSHEM.

The analysis is carried out with the two partitioning objectives available in METIS: *cut* and *vol*; the edge cut and the total communication volume respectively. Only three metrics are presented in this thesis: total edge cut, total communication volume, and maximum communication volume of all subdomains.

### Multiplier *-maxvwtm*

The multiplier *-maxvwtm* limits the size of vertices during the coarsening process. Reducing its value produces more balanced initial partitions and the refinement process is also optimized. However, a low value may have also undesired effects such as the inability to match vertices that could lead to an infinite loop trying to contract the graph without success.

Based on the results obtained from the experiments with synthetic graphs, the values chosen for the multiplier *-maxvwtm* are designed to match those of the first set. They provide a wide view and reduce the number of experiments in the set.

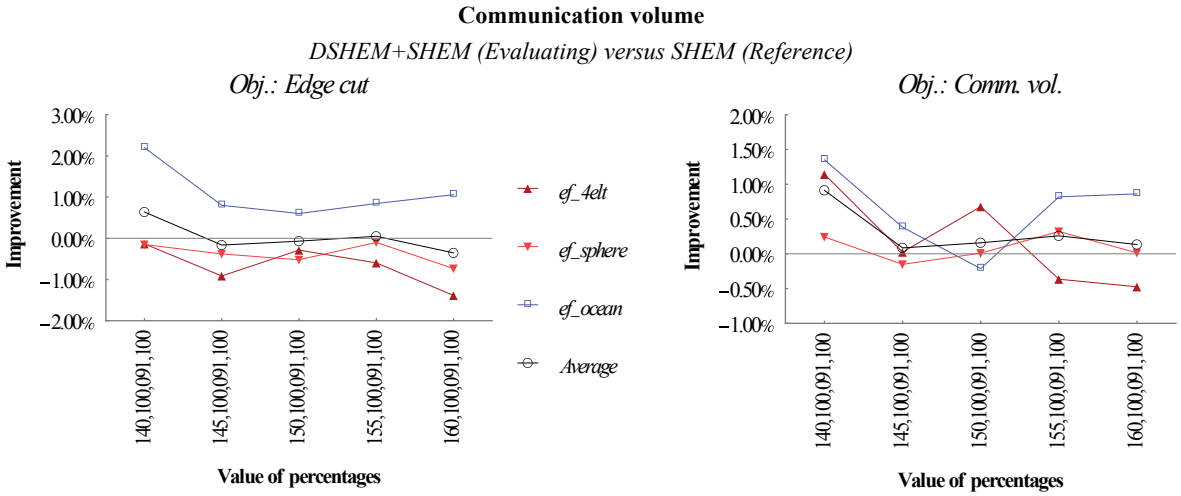


**Figure 9.53. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

The total communication volume is improved when nested DSHEM is executed as DSHEM+SHEM with the graph *ef_ocean*, see Figure 9.53. However, the value of multiplier *-maxvwtm* does not play a predictable role; it depends on the instance of the problem. Results are poor for the two graphs with triangular-like geometries. When nested DSHEM is executed as SHEM+DSHEM, see Figure 9.54, the results are inferior; as found with the synthetic graphs.

The maximum communication volume of all subdomains presents a similar performance as the total communication volume, see Figure 9.55 and Figure 9.56; DSHEM+SHEM produces better results than SHEM+DSHEM with the graph *ef_ocean*. The quality of the partition is heavily degraded with the graph *ef_4elt*; it is not clear why this degradation as the synthetic graphs performed better. Nonetheless, the triangular geometries do not perform as well as the quadrangular ones.

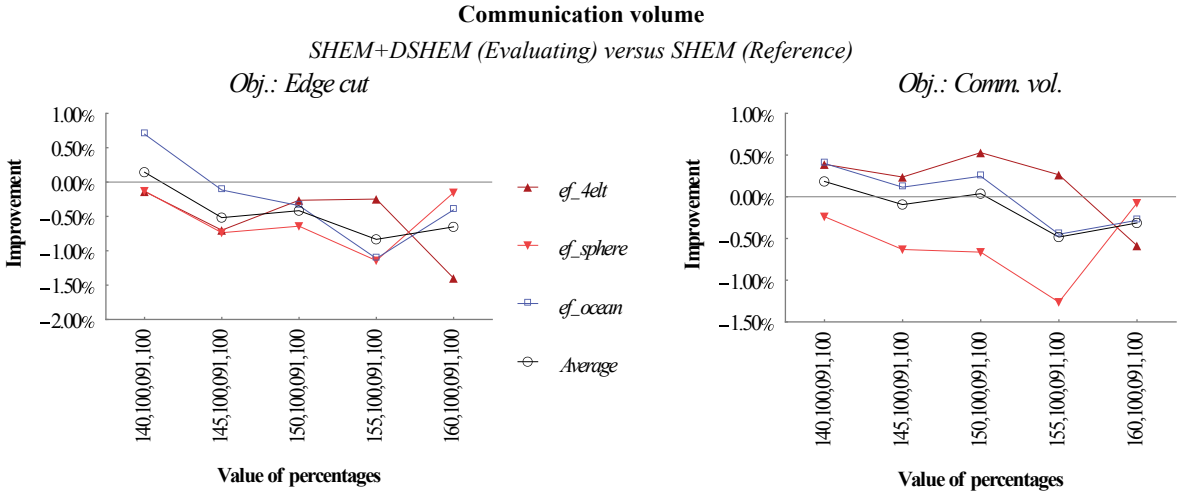


**Figure 9.54. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

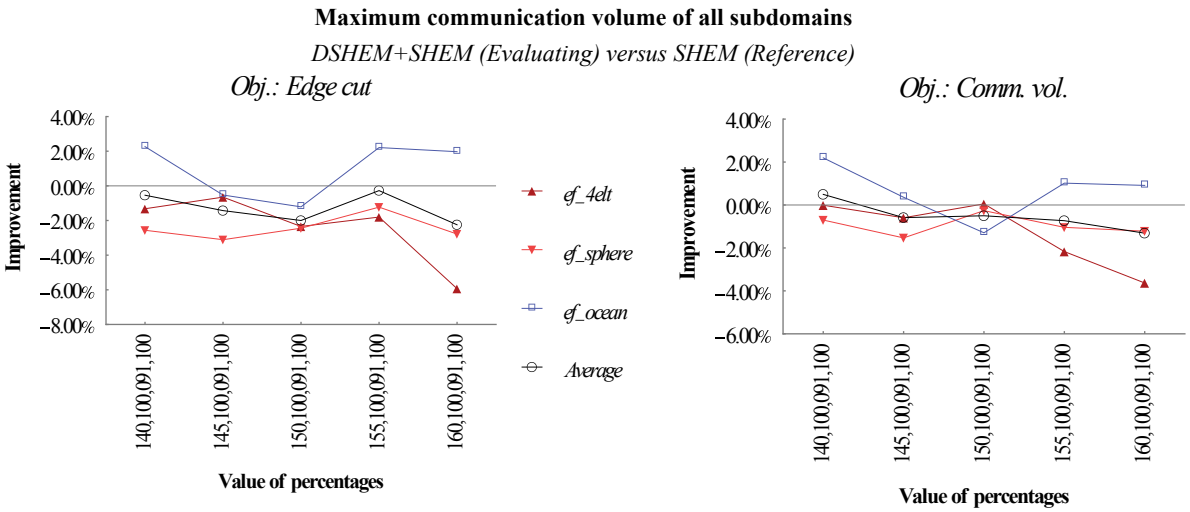


**Figure 9.55. DSHEM+SHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

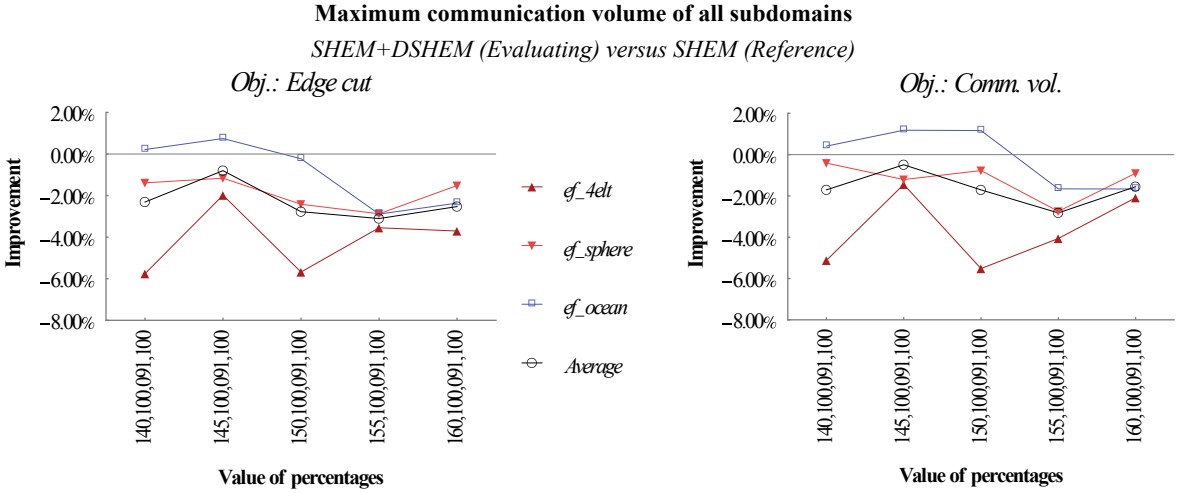


**Figure 9.56. SHEM+DSHEM vs. SHEM: effect of *-maxvwtm* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

### Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

Percentages *-dshem_p1* and *-dshem_p3* are excluded from the analysis as they do not play any role in the partitioning process. Percentage *-dshem_p2* is used to modify the behavior of the cost function in nested DSHEM and improve the partition.

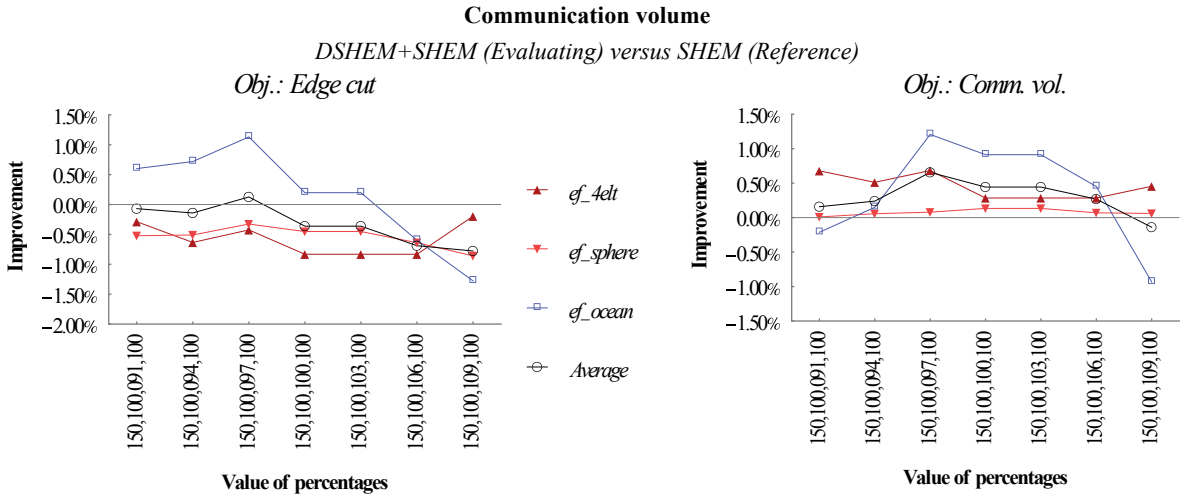


**Figure 9.57. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

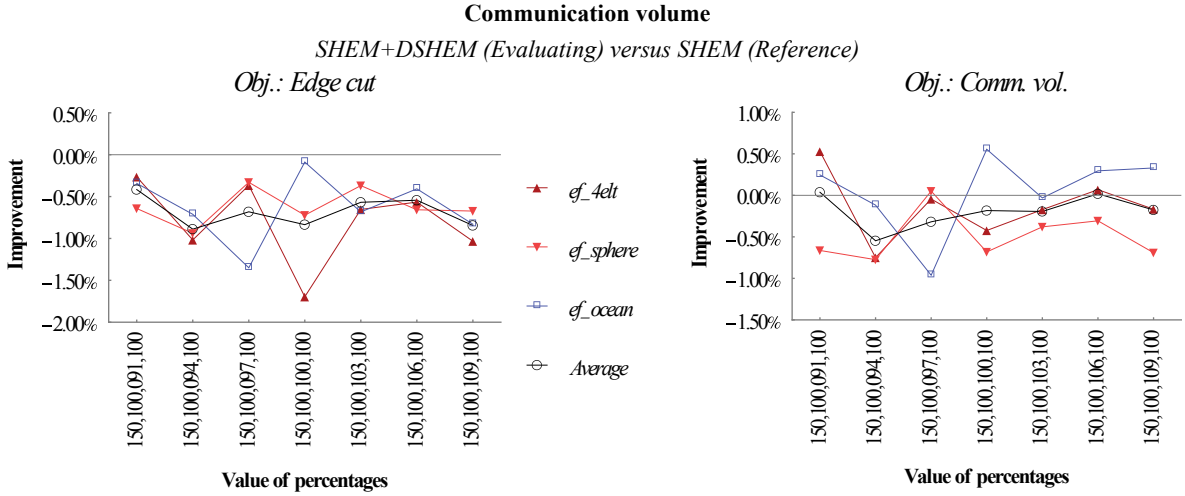


**Figure 9.58. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

From Figure 9.57 and Figure 9.58 it is possible to confirm that DSHEM executed as DSHEM+SHEM performs better and more stable than SHEM+DSHEM when the total communication volume is evaluated. Also, when the optimization objective is *vol*, the communication volume, the results are better as expected.

Again, the graph *ef_ocean* has better results, than the other two, with the maximum communication volume of all subdomains, see Figure 9.59 and Figure 9.60. Nested DSHEM, executed as DSHEM+SHEM, improves the quality of the partition when the percentage *-dshem_p2* is lower than 100 for the synthetic 3D square graph, however, it seems to be the opposite for the graph *ef_ocean*, with a similar geometry.

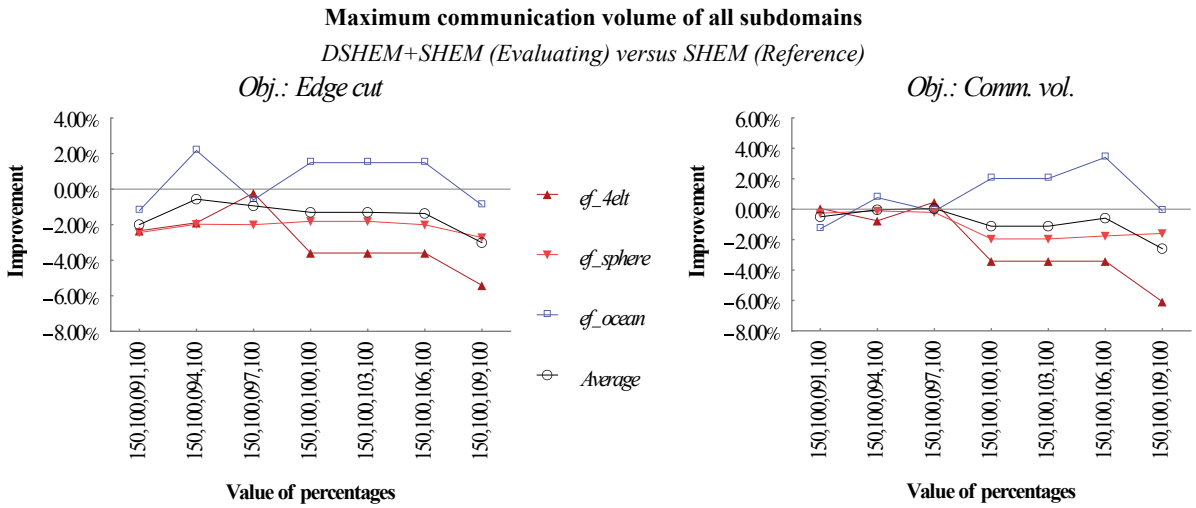


**Figure 9.59. DSHEM+SHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

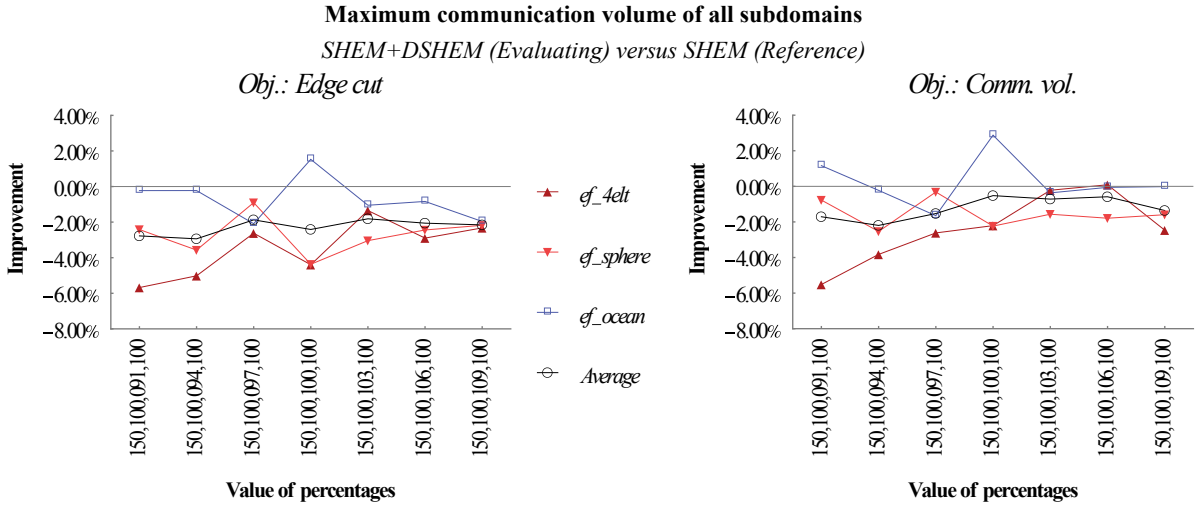


**Figure 9.60. SHEM+DSHEM vs. SHEM: effect of *-dshem_p2* on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

Still, nested DSHEM executed as DSHEM+SHEM seems to produce better, a more stable, results than SHEM+DSHEM according to the results.

## Refinement

METIS is executed without refinement for both, SHEM and nested DSHEM, to analyze its effect on the partitions. This process is responsible of the majority of the execution time; it improves the initial partition according to the partitioning objective.

The refinement plays a significant role in the partitioning process. It is evident, from the results in Figure 9.61, that nested DSHEM executed as DSHEM+SHEM improves the quality of the partitions when the total communication volume is evaluated. However, once the refinement is included, the results are degraded. The results produced by SHEM+DSHEM are more stable, but with lower quality, see Figure 9.62. From Figure 9.63 and Figure 9.64, it is possible to see that the maximum communication volume of all subdomains presents a similar picture to the total communication volume.

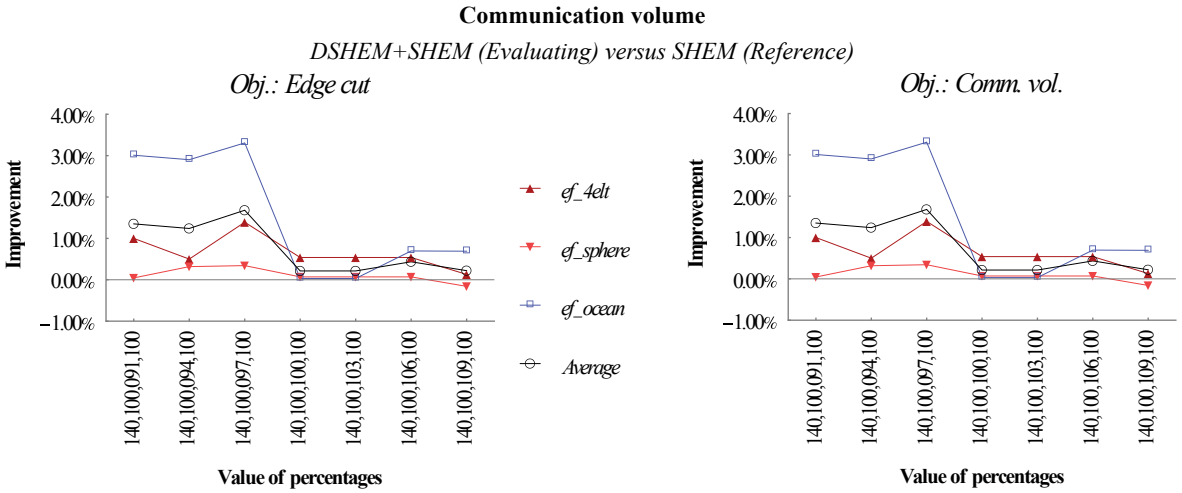


**Figure 9.61. DSHEM+SHEM vs. SHEM: effect of refinement on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

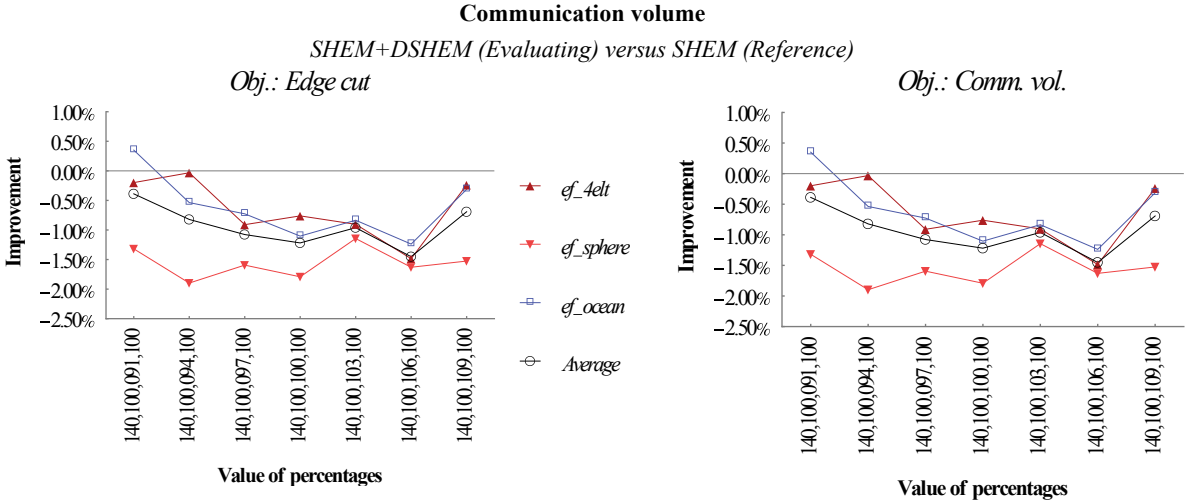


**Figure 9.62. SHEM+DSHEM vs. SHEM: effect of refinement on the communication volume with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

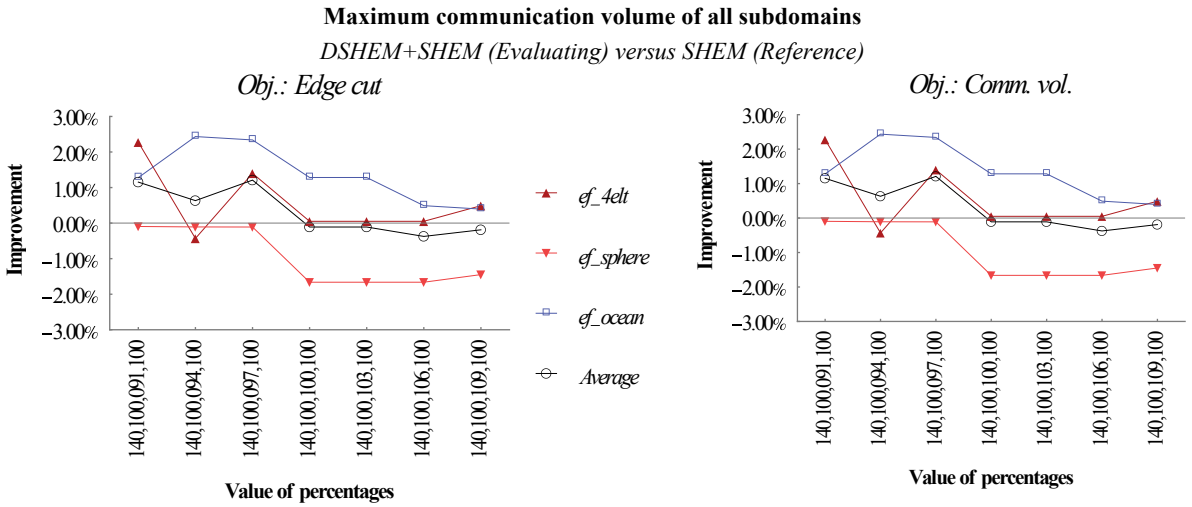


**Figure 9.63. DSHEM+SHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

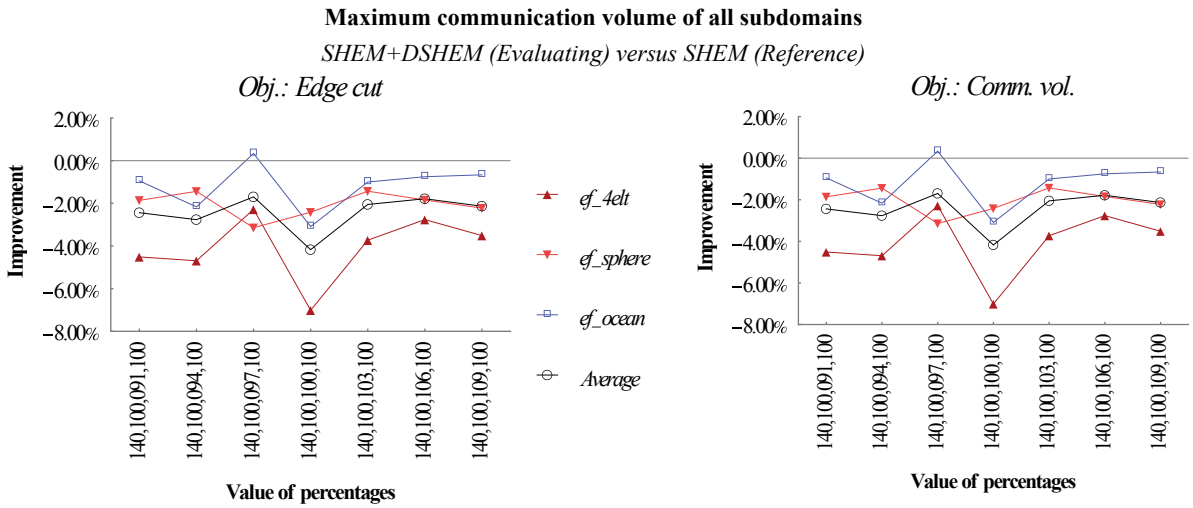


**Figure 9.64. SHEM+DSHEM vs. SHEM: effect of refinement on the maximum communication volume of all subdomains with real life graphs. Partitioning objective: edge cut on the left, communication volume on the right.**

## Execution Time

The evaluation of nested DSHEM provides similar results as the full implementation of DSHEM. There is virtually no difference in the execution time when nested DSHEM is used with the graph *ef_4elt*, as depicted by Figure 9.65. Again, there is a connection between the number of subdomains and the partitioning time; the more subdomains the longer to partition the graph.

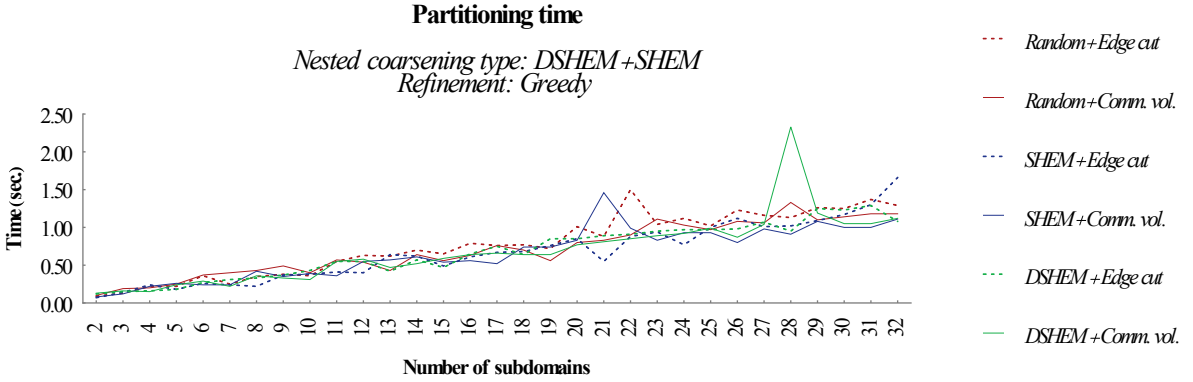


**Figure 9.65. Partitioning time with graph *ef_4elt* greedy refinement. Nested DSHEM is executed as DSHEM+SHEM.**

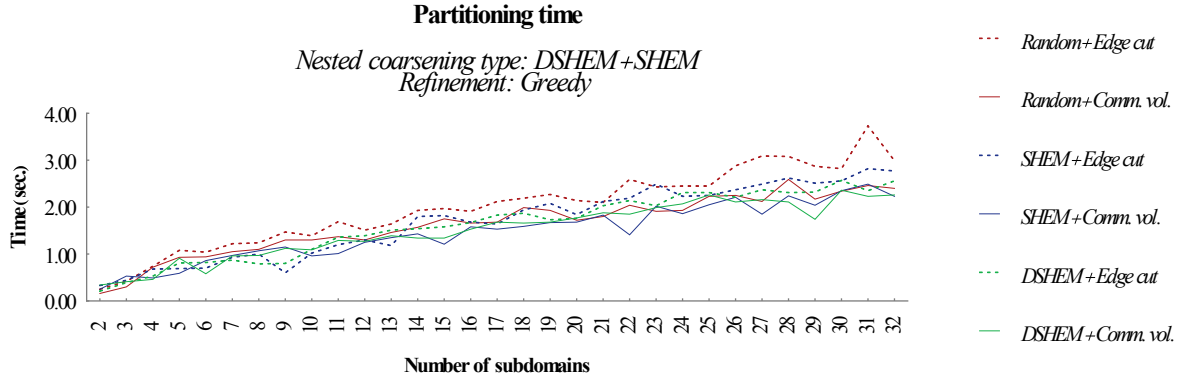


**Figure 9.66. Partitioning time with graph *ef_sphere* and greedy refinement. Nested DSHEM is executed as DSHEM+SHEM.**

Figure 9.66 and Figure 9.67 also confirm the results from the full implementation of DSHEM. The

triangular geometries keep the partitioning time balanced whether the refinement process optimizes the edge cut or the communication volume. SHEM and nested DSHEM keep a close gap in terms of partitioning time.

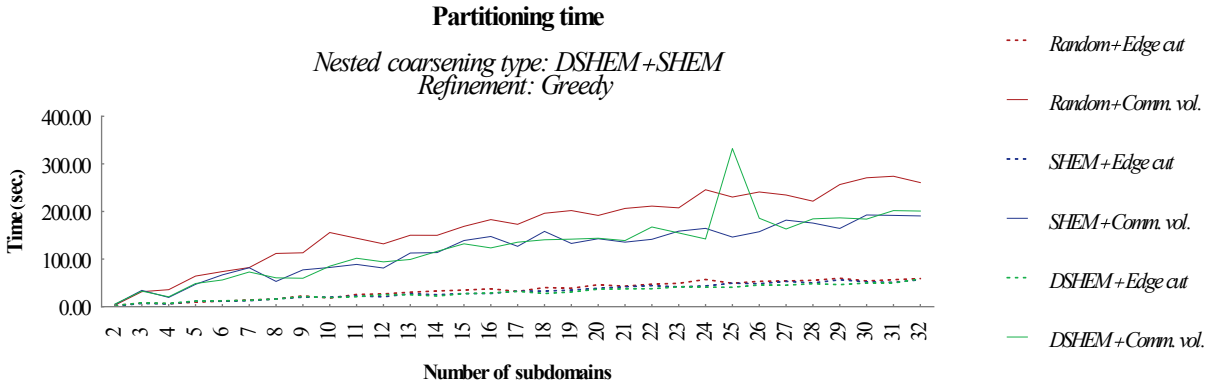


**Figure 9.67. Partitioning time with graph *ef_ocean* and greedy refinement. Nested DSHEM is executed as DSHEM+SHEM.**

The execution time without the refinement process is not included in this section. It proves that nested DSHEM is slower than SHEM, but a few extra seconds have no influence when the overall partitioning time is in the range of minutes.

## 9.6. Discussion

DSHEM was first evaluated with a full implementation using the original idea. The initial results are promising; however, the refinement process is far from perfect and needs to be adapted for the new type of graph generated by DSHEM. A new idea to overcome this problem is the nested DSHEM that combines both strategies, SHEM and DSHEM, to try to improve the partitions. The general discussion presented in this section is divided in the same way as the experiments; it helps to understand the results.

### 9.6.1. Impact of Multiplier *-maxvwtm*

Whether nested DSHEM is executed as DSHEM+SHEM or SHEM+DSHEM the influence of the multiplier *-maxvwtm* is invariant. Similar to the full implementation of DSHEM, the multiplier modifies the execution and brings different results, but no clear pattern can be inferred from the results. It is also clear that it is more frequent to obtain better results when the value of the multiplier *-maxvwtm* is reduced from the default 150.

The quality of the partition also depends on the type of graph, for example the 2D triangular square graph (*tsm2d100p*) benefits from the multiplier when its value is smaller than 150 and nested DSHEM is executed as SHEM+DSHEM. When executed as DSHEM+SHEM, the quality of the partition for this type of graph remains constant with different values for the multiplier. The results show that the real life graphs have also a similar behavior when the value of the multiplier is modified; they follow the same trends according to the geometry of the graph.

The execution of nested DSHEM as DSHEM+SHEM gives more stable results when the communication volume and the maximum communication volume of all subdomains are evaluated; SHEM+DSHEM is steadier when the edge cut is evaluated.

### 9.6.2. Impact of Percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3*

The percentages *-dshem_p1*, *-dshem_p2* and *-dshem_p3* present a similar behavior with the full implementation of DSHEM and nested DSHEM; *-dshem_p1*, and *-dshem_p3* do not have any influence in the execution of nested DSHEM as well. The impact of percentage *-dshem_p2* is more evident when nested DSHEM is executed as DSHEM+SHEM; this is due to the fact that the first strategy has more influence in the outcome.

Yet again, an inflexion point is visible when the value of the percentage *-dshem_p2* is set to 100. When nested DSHEM is executed as SHEM+DSHEM, this point fades out as the influence of SHEM dominates the final partition. The percentage *-dshem_p2* has a stable and predictable impact on DSHEM when the refinement is not part of the partitioning process.

The graphs with a quadrangular-like geometry benefits from the percentage *-dshem_p2* and values smaller than 100 when DSHEM is executed as DSHEM+SHEM; triangular geometries with SHEM+DSHEM.

### 9.6.3. Impact of Graph Irregularity on Nested DSHEM

The results of nested DSHEM confirm those from the full implementation of DSHEM; there is not a clear correlation between the degree of irregularity introduced to the synthetic graphs and the quality of the partitions generated by nested DSHEM. All three algorithms, Random, SHEM and DSHEM are affected in the same way and the results suggest that the quality of the partition depends on greater measure on the particular instance of the graph.

Another conclusion can be established from the results, the performance of nested DSHEM is not affected when the degree of irregularity does not drastically modify the original geometry of the graph. It is the same conclusion with the full implementation of DSHEM based on the previous results. DSHEM, and nested DSHEM, will provide the same partition quality with the different geometries when the degree of irregularity introduced to the graphs remains reasonable.

### 9.6.4. Impact of Refinement on Nested DSHEM

When the refinement process is not part of the partitioning process, METIS does not optimize the objectives edge cut or communication volume; the quality of the partition is then defined solely by the coarsening strategy. To evaluate the impact of nested DSHEM on the partition, the refinement process is taken out of the partitioning process. Based on the results from the full implementation of DSHEM and nested DSHEM, it is clear that in most cases the refinement process is much slower when the optimization objective is the communication volume. A rational assumption dictates that the reason for this disparity is because METIS was originally designed to reduce the edge cut; the coarsening and refinement process were designed with this in mind. Later, the optimization of the communication volume was introduced to the refinement process, but the coarsening process still focuses on the edge cut. Now, with the introduction of DSHEM optimizing the communication volume within the coarsening process, the refinement process needs to be redesigned to take advantage of the new type of coarse graph.

The results obtained with the synthetic graphs show that nested DSHEM can improve the quality of the partitions when the graph has a quadrangular-like geometry, and in certain circumstances with triangular-like geometries. The improvement is more evident when nested DSHEM is executed as DSHEM+SHEM; it is more irregular when executed as SHEM+DSHEM.

Some patterns emerge from the synthetic graphs that can be used to ensure a higher quality in the partition according to the type of graph, but once the refinement process is included those patterns are distorted. The same conclusion can be attaint; the refinement process does not efficiently interact with the coarsening process and needs to be redesigned to effectively optimize the communication volume.

### 9.6.5. Impact of Nested DSHEM on the Execution Time

The results show that there is virtually no difference between the full implementation of DSHEM and nested DSHEM with respect to the execution time. From the initial results with the full implementation of DSHEM, it becomes evident that the time spent in the coarsening process is minimal and can even been discarded from the overall execution time; conversely, the refinement process dictates the final partitioning time.

Whether nested DSHEM is executed as DSHEM+SHEM or SHEM+DSHEM, the execution time has essentially no variations. The same patterns emerge with nested DSHEM: optimizing the edge cut is up to 4 times faster than the communication volume with quadrangular geometries. Similar to the full implementation of DSHEM, nested DSHEM could be faster than SHEM with some types of graphs.

### 9.6.6. Global Evaluation of Nested DSHEM

The full implementation of DSHEM and nested DSHEM are different in two key aspects: nested DSHEM combines the graphs generated by SHEM and DSHEM to create the partition, and nested DSHEM requires a second data structure to store the second graph. This makes nested DSHEM more expensive in memory consumption. Concerning the execution time, nested DSHEM remains as competitive as the full implementation of DSHEM.

A general conclusion can be achieved from the experimental analysis: nested DSHEM can improve the quality of partitions under certain circumstances when the communication volume is considered. The type of geometry present in the graph has a great influence on the final partition and nested DSHEM, as well as the full implementation of DSHEM, provides better results with quadrangular geometries.

The refinement process impacts the performance of nested DSHEM in a similar way as the full implementation of DSHEM. The results suggest that the refinement process is not well suited for the type of graph generated by DSHEM, and its performance is degraded when the communication volume is optimized. Further work on the refinement process is necessary to ensure that it will improve the partitions instead of degrade them, when the communication volume is optimized.

Nested DSHEM is slower than SHEM due to its design; nonetheless, the impact on the overall execution time is irrelevant as the coarsening process takes a negligible part of it. The refinement process is affected by the coarsest graph generated by SHEM or DSHEM; it may increase the time spent on the refinement process. If the series of coarse graphs generated during the coarsening process have a low quality, the refinement process will take longer to optimize the partitioning objective and generate a

good partition at the end of the process.

Similar to the full implementation of DSHEM, the irregularity introduced to the graphs has no clear effect on the partitions generated by the nested DSHEM. The results remain steady when the degree or irregularity is reasonable and the geometry of the graph does not change.

The analysis of nested DSHEM with the real life graphs confirms the conclusions with the synthetic ones. Nested DSHEM can bring benefits with certain types of graphs.

# Part IV.

# Discussion

# Chapter 10.

# Conclusions and Future Perspectives

Current FEM applications heavily rely on large computational, memory and communication costs to provide high accurate solutions. Much work has been done to improve the efficiency on parallel systems; however, new hardware architectures bring new challenges for parallel applications. This chapter presents the final conclusions of this research work, as well as future perspectives; it also summarizes the highlights of the research.

## 10.1. Conclusions about the Research Questions

This work was motivated by the need of more efficient FEM simulations on parallel systems. The main objectives are planned around two central approaches: a high and a low level method to improve the efficiency. The high level approach involves a new multilevel hierarchical load balancer designed to dynamically balance the load of parallel FEM applications. The low level approach focuses on the graph partitioning problem which changes the paradigm with the aim to improve the partitions when the communication volume is considered.

### 10.1.1. Load Balancing

The first questions are related to the load balancing problem on FEM simulations. This problem is not new and, with emergent technologies, keeps evolving during the years. New approaches to existing problems are necessary to improve the efficiency of parallel FEM applications. Some questions arise since the beginning of this work, and based on the subsequent study and research, they need to be addressed.

#### Can a generic load balancer fulfill the current requirements of FEM applications? And if so, how can it be achieved?

The initial focus of the research is the load balancing problem for FEM simulations. Many of the previous approaches use methods optimized for specific scenarios to increase its performance; they are efficient but cannot address other use cases due to its nature. Generic load balancers may be less effective in specific cases, but provide good results for a wide range of problems.

The hierarchical approach on many problems, such as in graph partitioning, has proven to be highly efficient and fast; it is natural to assume that a similar approach could be beneficial for the load balancing problem. The multilevel hierarchical load balancer, presented in Section 10.4.2, provides an

insight on how a generic load balancer could be designed to efficiently distribute the load on parallel FEM applications. Further investigation is necessary to assess its viability and effectiveness.

**Many FEM applications rely on dynamic meshes to improve the accuracy of the results without incurring in extra computational cost. Can a similar approach be used in the design of a load balancer?**

A dynamic behavior within a load balancer can improve its efficiency, but at a cost; dynamic problems are often more difficult to handle. Despite this disadvantage, a simple approach can be used to minimize the cost and improve the load distribution in parallel FEM applications. Section 10.4.2 describes how the load balancer can dynamically move elements during the processing steps during the computations of a parallel FEM simulation.

The first level of the hierarchical load balancer generates the main distribution of the load during the load balancing steps of the parallel FEM simulation. Throughout the computational step, the second level of the load balancer is active and detects the imbalance according to the status of all processors. It is then able to move mesh elements from heavily loaded processors to those becoming idle.

### 10.1.2. Graph Partitioning

The main topic of the research work is the graph partitioning problem and how it can improve the distribution of the load in parallel FEM applications. Much effort has been focused on the graph partitioning problem to address the load balancing problem. Some areas of improvement have been identified throughout this work and some questions arise.

**Can a graph partition be improved and is it worth the effort?**

Hierarchical graph partitioning is fast and generates competitive partitions compared with more expensive methods. It is obvious that this approach is a good candidate to improve the partitions without the extra computational cost seen with other methods. The proposed algorithm DSHEM is an example of how the quality of partitions can be improved without incurring in extra computational costs. While the algorithm requires additional time to contract the graph, this is only a minimal percentage of the overall partitioning time. Still, work is necessary to improve further the partitions when the communication volume is important; the refinement process requires additional examination to adapt it to the new partitioning objective. The initial results are promising and suggest that the effort to continue the research is worth the final results.

**If communication costs are important while generating a distribution of the load, can it be included in the partitioning process of the graph that represents the mesh of information? And how this new objective will affect the partitioning process?**

Different types of graphs can be used to characterize the mesh of information used by the FEM applications; some provide higher accuracy, but require more computational power to be processed. With the appropriate graph, the computational cost can be reduced while the quality of the partition can

be kept under control. Nonetheless, it is necessary to find a compromise between the type of graph and the partitioning method to achieve the best results.

DSHEM demonstrates that with the same type of graph, or input data, the partitions can be improved without an increase on the overall partitioning time. With a precise change in the way the weight of the edges is calculated, it is possible to better emulate the communication costs of the different subdomains. This approach has an effect on the refinement process which was originally designed to optimize the edge cut. Further study is necessary in order to improve the interaction of the coarsening and refinement process and enhance the quality of partitions.

## 10.2. Conclusions about the Research Problem

In this thesis, we have presented an overview of improvements on graph partitioning techniques for load balancing, and efficiency of FEM simulations on large-scale parallel machines. Much work has been done in the field, but requirements of emerging technologies are not fulfill by state-of-the-art libraries. We also introduced a new vertex matching model called DSHEM aimed to reduce the communication volume during FEM simulations. It is based on the sorted heavy-edge matching implemented in the graph partitioning library METIS. We also introduced a new hierarchical load balancer for parallel FEM applications which can improve their efficiency. Our new model can be successfully used as a starting point for a more complex strategy. The study can be extended to a number of directions including the inclusion of other cost functions and multi-objective cost functions.

DSHEM improves partitions generated from undirected graphs if the communication volume is considered. The extra time needed to generate the partition is negligible compared to the original algorithm SHEM. No use of extra memory is needed to emulate the directed graph. DSHEM can improve the system efficiency of parallel FEM computations under certain circumstances.

We continue the development and implementation of the proposed vertex matching model to improve its performance regarding running times and quality of the partition. The refinement process needs to be further studied and adapted to the new data generated by DSHEM. From the results of the full implementation of DSHEM, and nested DSHEM, it is evident that the refinement process can hide the advantages brought by DSHEM.

## 10.3. Implications and Limitations

To fully exploit the benefits of any piece of technology is important to understand its strengths and limitations. This also applies to the problem in hand; if not understood correctly, the proposed solutions may not fulfill all expectations. Important considerations that should be taken into account while using DSHEM to partition a graph are provided next.

Although the implementation of DSHEM is mature enough and the experimental analysis proves its efficiency, several factors heavily influence its performance. We would not advice to blindly use DSHEM without prior examination of the use case.

Important insights come from the evaluation of the algorithm. The geometry of the graph affects the quality of the partition; however, tuning up the execution of DSHEM helps to improve it. It is advised to *experiment* with the execution parameters of METIS to identify the set of values that provide the best

results. Default values may be good in general, but specific use cases can benefit from custom values.

It is also evident that DSHEM inherits METIS characteristics as it is based on SHEM and implemented within the partition library. The use cases suitable for METIS are consequently appropriate for DSHEM. It is up to the user to decide when and how to utilize the algorithms in METIS for better results.

### 10.3.1. Graph Type

The type of the graph is an important consideration while selecting a graph partitioning tool. Each graph type is designed for a particular set of requirements and several tools may not be suitable, or even capable, to partition them. A complete understanding of the graph and partitioning tool is essential for obtaining high quality results.

DSHEM, as part of METIS, shares all capabilities and limitations of this library. It is designed to partition irregular graphs and meshes that arise from FEM simulations. In addition, the format of the graph is also important; METIS uses undirected graphs as input. We refer the reader to [36], [37] for more details.

### 10.3.2. Graph Geometry

The graph geometry plays an important role in the quality of the partition. While some algorithms are optimized for a particular set of graphs, they may not perform well with others.

DSHEM is designed for partitioning graphs arising from FEM simulations. The experimental analysis shows that the quality of the partitions is higher with certain graph geometries; however, it is not degraded with the rest.

### 10.3.3. Current Implementation

Though the current implementation is fully operational, it is not ready for production use. There exists debugging code to monitor and control its behavior during execution. The most significant disadvantage is its execution time which is increased by the extra code. However, it is not considerably larger compared with the original SHEM algorithm. It would be incorrect to state that DSHEM is in disadvantage do to the longer processing time required to contract the graph; the structure of the contracted graph affects the refinement process too and the overall execution time is not affected.

## 10.4. Future Research

The research presented in this thesis focuses on a new matching strategy to produce optimized partitions in terms of communication volume. However, METIS is a collection of different algorithms used in the different stages of the partitioning process. A closer inspection of the effects of the refinement algorithms on DSHEM is still needed.

Much work has already been done in the field, nonetheless the requirements of new emerging technologies are not entirely fulfill by current state-of-the-art libraries. We propose a multilevel

hierarchical load balancer which improves the local load imbalance [14]. It is based on graph partitioning algorithms and takes into account the hardware architecture of the underlying system. The enhancement to the cost function presented by Olas et al. [29], including new information, helps to better approximate the computations and load balancing costs of the next FEM computation step. The new model can successfully be used as a starting point for a more complex load balancing strategy. The research can be extended to a number of directions including the development of a more intricate cost function, and prediction model into the multilevel load balancer.

### 10.4.1. Effect of the Refinement Process on DSHEM

METIS uses a multilevel approach to generate the partition of a graph and every phase of the process affects the overall result. The current implementation of DSHEM focuses on the matching phase, during the contraction of the graph. Several KL based algorithms [68] help to improve the partition during the uncoarsening phase. This set of algorithms is not designed to use the new directional communication generated by DSHEM; they have only been adapted to use the information in the *new format*.

It is an interesting direction for the research the effect of the refinement process of METIS on DSHEM. The quality of partitions can improve if optimized versions of the refinement algorithms are implemented. These algorithms need to include the new model of the graph into their cost function in order to make proper decisions.

### 10.4.2. Multilevel Hierarchical Load Balancer

As previously stated, new hardware architectures bring new capabilities and new problems in resource management. New approaches and algorithms have to be developed in order to overcome these issues. To this end, we propose a new multilevel load balancing model, which aims to reduce the local imbalance, while tries to reduce the global communication overhead. The use of resource information and a cost function is important to achieve a good load balance.

Compute time has to scale linearly with respect to the problem and the number of processors. Additionally, local memory requirements should only depend on the local, not the global problem size. To efficiently distribute data on the underlying system, we need to gather information about the computing environment (e.g., processors, network topology and memory). A perfect balanced partition is worthless if it cannot be efficiently mapped. Such partitions have to be computed based on the knowledge of the system. A non balanced partition could fit better to specific hardware architectures (e.g., when processor speeds differ between them). The system information is gathered before the actual FEM simulation begins using a configuration step. In case of dynamic resources, this step has to be performed before each computation step within the simulation. There exist libraries, such as LINPACK [162], that can be used for this purpose.

Our model works as follows. The first level is responsible for the main load balancing steps. It performs the load distribution over the entire system, such as traditional models, before each computation step. We use additional information to compute the mesh partitioning and mapping. A graph is built from the available hardware information which represents the underlying system. Vertices represent processors and edges network links; both can be weighted to mimic the heterogeneity.

Therefore, we use two graphs, one representing the mesh, and one the system. With the extra information, a partition that better fits the system can be found. In this way, we are able to better distribute the load among the processors using well-known libraries such as METIS in combination with the Directed Sorted Heavy Edge Matching.

A similar cost model to the one proposed by Olas et al. [29] can be used to determine if a balancing step is required or not. If the time required by the load balancing step is smaller to the time that will be saved with a new distribution, then it is performed. We enhanced the model by adding additional information and handling the system heterogeneity. Instead of computing the communication time by only multiplying the amount of data to be transferred and the network speed, we take into account the speed of each network link independently; the same is applied to the computing time. In this way, we have a more accurate prediction, and, thus, the second level of load balancing will provide better results.

The second level uses hardware information to perform a local load balancing. It is not a separate step; instead it is performed during computations. First, we identify clusters of processors (groups of processors joint by high speed network links). This can be done during the configuration step before the FEM simulation (or during each configuration step before each computation step in a dynamic system). Second, we identify the mesh cells with numbers. These numbers represent the gain of moving the cell to a neighbor processor in the case of imbalance. This is done during the last global load balancing step when the partition is refined. We keep these values and use them to improve local imbalance in this balancing level. As previously mentioned, the graph model does not represent the exact real workload. Thus, the imbalance may become evident during a computation step. According to the progress in solving PDEs by each processor, we can decide to move some mesh cells to a neighboring processor within the cluster of processors with high speed network links. Overloaded processors migrate mesh cells to neighbors during the computation step. This is done only if local predictions assure a gain in performance. As these communications are done concurrently and locally, the performance of the whole system is not degraded.

This approach solves some of the problems we have described before. We believe that tuning-up the cost functions, used in predictions during the simulation, we can achieve better results. Including more information in the partitioning process may add complexity to the problem; but if used efficiently, a good improvement in performance can be achieved.

A typical HPC environment for FEM simulations may contain thousands of processors with around 8 GB of memory and 20 GB of HDD per core. InfiniBand is widely used to interconnect nodes within the system. Our model also takes into account the Grid computing model which enables the use of geographically distributed systems as a single resource.

# Bibliography


[1] T. Olas, K. Karczewski, A. Tomas, and R. Wyrzykowski, "FEM computations on clusters using different models of parallel programming," in *Parallel Processing and Applied Mathematics*, vol. 2328, no. 2006, R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waśniewski, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 170–182.

[2] O. C. Zienkiewicz and R. L. Taylor, *The finite element method: The basis*, 5th ed., vol. 1. Oxford, United Kingdom: Butterworth-Heinemann, 2000.

[3] S. Blazy, W. Borchers, and U. Dralle, "Parallelization methods for a characteristic's pressure correction scheme," in *Flow Simulation with High-Performance Computers: II*, E. H. Hirschel, Ed. Braunschweig/Wiesbaden, Germany: Friedrich Vieweg & Sohn Verlagsgesellschaft mbH, 1996, p. 576.

[4] Y. Saad, *Iterative methods for sparse linear systems*, 2nd ed. Philadelphia, PA, United States of America: Society for Industrial and Applied Mathematics, 2003.

[5] R. Verfürth, "A posteriori error estimation and adaptive mesh-refinement techniques," *J. Comput. Appl. Math.*, vol. 50, no. 1–3, pp. 67–83, 1994.

[6] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC'74)*, 1974, pp. 47–63.

[7] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

[8] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco, CA, United States of America: W. H. Freeman and Company, 1979.

[9] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *Computer (Long. Beach. Calif).*, vol. 25, no. 12, pp. 33–44, 1992.

[10] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering: Selected Results and Surveys*, vol. 9220, L. Kliemann and P. Sanders, Eds. Springer-Verlag, 2016, pp. 117–158.

[11] P.-O. Fjällström, "Algorithms for graph partitioning: A survey," *Linköping Electron. Artic. Comput. Inf. Sci.*, vol. 3, no. 1998, Sep. 1998.

[12] J. Chen and V. E. Taylor, "Mesh partitioning for efficient use of distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 1, pp. 67–79, 2002.

[13] J. L. González García, R. Yahyapour, and A. Tchernykh, “Load balancing for parallel computations with the finite element method,” in *3rd International Supercomputing Conference in Mexico*, 2012, p. 9.

[14] J. L. González García, R. Yahyapour, and A. Tchernykh, “Load balancing for parallel computations with the finite element method,” *Comput. y Sist.*, vol. 3, no. 17, pp. 299–316, Sep. 2013.

[15] B. J. Lint and T. K. M. Agerwala, “Communication issues in the design and analysis of parallel algorithms,” *IEEE Trans. Softw. Eng.*, vol. SE-7, no. 2, pp. 174–188, 1981.

[16] S. Moore and D. Greenfield, “The next resource war: computation vs. communication,” in *Proceedings of the 2008 international workshop on System level interconnect prediction*, 2008, pp. 81–86.

[17] E. D. Jensen, “The Honeywell experimental distributed processor – An overview,” *Computer (Long. Beach. Calif).*, vol. 11, no. 1, pp. 28–38, Jan. 1978.

[18] V. Bouchitté, P. Boulet, A. Darte, and Y. Robert, “Evaluating array expressions on massively parallel machines with communication/computation overlap,” *Int. J. High Perform. Comput. Appl.*, vol. 9, no. 3, pp. 205–219, Sep. 1995.

[19] F. Desprez, S. Domas, and B. Tourancheau, “Optimization of the ScaLAPACK LU factorization routine using communication/computation overlap,” in *Euro-Par’96 Parallel Processing*, 1996, pp. 1–10.

[20] M. J. Quinn and P. J. Hatcher, “On the utility of communication–computation overlap in data-parallel programs,” *J. Parallel Distrib. Comput.*, vol. 33, no. 2, pp. 197–204, Mar. 1996.

[21] A. K. Somani and A. M. Sansano, “Minimizing overhead in parallel algorithms through overlapping communication/computation,” Hampton, VA, United States of America, 1997.

[22] A. Danalis, L. Pollock, M. Swany, and J. Cavazos, “MPI-aware compiler optimizations for improving communication-computation overlap,” in *Proceedings of the 23rd international conference on Supercomputing*, 2009, pp. 316–325.

[23] G. A. Chadwick, “Communication centric, multi-core, fine-grained processor architecture,” Cambridge, United Kingdom, 2013.

[24] H. Cota de Freitas and P. O. A. Navaux, “Evaluating on-chip interconnection architectures for parallel processing,” in *2008 11th IEEE International Conference on Computational Science and Engineering - Workshops*, 2008, pp. 188–193.

[25] W. Heirman, J. Dambre, D. Stroobandt, and J. Van Campenhout, “Rent’s rule and parallel programs: characterizing network traffic behavior,” in *Proceedings of the 2008 international workshop on System level interconnect prediction*, 2008, pp. 87–94.

[26] T. Heister, M. Kronbichler, and W. Bangerth, “Massively parallel finite element programming,” in *Recent Advances in the Message Passing Interface*, vol. 6305, no. 2010, R. Keller, E. Gabriel, M. Resch, and J. Dongarra, Eds. Springer Berlin Heidelberg, 2010, pp. 122–131.

[27] C. Burstedde, M. Burtscher, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "ALPS: A framework for parallel adaptive PDE solution," *J. Phys.*, vol. 180, no. 1, p. 8, 2009.

[28] C. Burstedde, L. C. Wilcox, and O. Ghattas, "p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees," *SIAM J. Sci. Comput.*, vol. 33, no. 3, pp. 1103–1133, 2011.

[29] T. Olas, R. Leśniak, R. Wyrzykowski, and P. Gepner, "Parallel adaptive finite element package with dynamic load balancing for 3D thermo-mechanical problems," in *Parallel Processing and Applied Mathematics*, vol. 6067, no. 2010, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, Eds. Springer Berlin Heidelberg, 2010, pp. 299–311.

[30] R. Wyrzykowski, T. Olas, and N. Sczygiol, "Object-oriented approach to finite element modeling on clusters," in *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, vol. 1947, no. 2001, T. Sørevik, F. Manne, A. H. Gebremedhin, and R. Moe, Eds. Berlin, Germany: Springer-Verlag, 2001, pp. 250–257.

[31] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, "New challenges in dynamic load balancing," *Appl. Numer. Math.*, vol. 52, no. 2–3, pp. 133–152, Feb. 2005.

[32] S. Sinha and M. Parashar, "Adaptive system sensitive partitioning of AMR applications on heterogeneous clusters," *Cluster Comput.*, vol. 5, no. 4, pp. 343–352, 2002.

[33] C. H. Walshaw and M. Cross, "Multilevel mesh partitioning for heterogeneous communication networks," *Futur. Gener. Comput. Syst.*, vol. 17, no. 5, pp. 601–623, 2001.

[34] T. Minyard and Y. Kallinderis, "Parallel load balancing for dynamic execution environments," *Comput. Methods Appl. Mech. Eng.*, vol. 189, no. 4, pp. 1295–1309, 2000.

[35] J. D. Teresco, M. W. Beall, J. E. Flaherty, and M. S. Shephard, "A hierarchical partition model for adaptive finite element computation," *Comput. Methods Appl. Mech. Eng.*, vol. 184, no. 2–4, pp. 269–285, 2000.

[36] G. Karypis and V. Kumar, "METIS - Serial graph partitioning and fill-reducing matrix ordering," 2012. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview. [Accessed: 16-Jan-2018].

[37] G. Karypis, "METIS A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," Minneapolis, MN, United States of America, 2011.

[38] Y. Chen, B. L. Nobley, and R. D. Chamberlain, "Comparing edge-cuts to communications volume in parallel VLSI logic simulation," in *Proc. of the 8th IASTED International Conference on Parallel and Distributed Computing and Systems*, 1996, pp. 481–484.

[39] J. L. González García, R. Yahyapour, and A. Tchernykh, "Graph Partitioning for FEM Applications: Reducing the Communication Volume with DSHEM (under review)," in *Submitted to HPCS 2019 - The 2019 International Conference on High Performance Computing & Simulation*, 2019.

[40] J. L. González García, R. Yahyapour, and A. Tchernykh, “Graph Partitioning for FEM Applications: Reducing the Communication Volume with Nested DSHEM (under review),” in *Submitted to PPAM 2019 - 13th International Conference on Parallel Processing and Applied Mathematics*, 2019.

[41] R. Diekmann, U. Dralle, F. Neugebauer, and T. Römke, “PadFEM: A portable parallel FEM-tool,” in *High-Performance Computing and Networking*, vol. 1067, no. 1996, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, Eds. Berlin, Germany: Springer Berlin / Heidelberg, 1996, pp. 580–585.

[42] F. Hülsemann, M. Kowarschik, M. Mohr, and U. Rüde, “Parallel geometric multigrid,” in *Numerical Solution of Partial Differential Equations on Parallel Computers*, vol. 51, A. M. Bruaset and A. Tveito, Eds. Springer Berlin Heidelberg, 2006, pp. 165–208.

[43] K. Ho-Le, “Finite element mesh generation methods: A review and classification,” *Comput. Des.*, vol. 20, no. 1, pp. 27–38, 1988.

[44] A. Ghavidel, S. R. Mousavi, and M. Rashki, “The effect of FEM mesh density on the failure probability analysis of structures,” *KSCE J. Civ. Eng.*, pp. 1–13, Sep. 2017.

[45] S. A. Ashford and N. Sitar, “Effect of element size on the static finite element analysis of steep slopes,” *Int. J. Numer. Anal. Methods Geomech.*, vol. 25, no. 14, pp. 1361–1376, Sep. 2001.

[46] D. N. Dyck, D. A. Lowther, and S. J. McFee, “Determining an approximate finite element mesh density using neural network techniques,” *IEEE Trans. Magn.*, vol. 28, no. 2, pp. 1767–1770, Mar. 1992.

[47] Z. Li, M. W. Kindig, D. Subit, and R. W. Kent, “Influence of mesh density, cortical thickness and material properties on human rib fracture prediction,” *Med. Eng. Phys.*, vol. 32, no. 9, pp. 998–1008, Nov. 2010.

[48] A. Perillo Marcone, A. Alonso Vázquez, and M. Taylor, “Assessment of the effect of mesh density on the material property discretisation within QCT based FE models: A practical example using the implanted proximal tibia,” *Comput. Methods Biomech. Biomed. Engin.*, vol. 6, no. 1, pp. 17–26, 2003.

[49] Y. Liu and G. Glass, “Effects of mesh density on finite element analysis,” in *SAE Technical Papers*, 2013.

[50] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, and J. F. O’Brien, “Dynamic local remeshing for elastoplastic simulation,” *ACM Trans. Graph.*, vol. 29, no. 4, Jul. 2010.

[51] O. C. Zienkiewicz and R. L. Taylor, *The finite element method: Fluid dynamics*, 5th ed., vol. 1. Oxford, United Kingdom: Butterworth-Heinemann, 2000.

[52] B. Hudson, “Dynamic mesh refinement,” ProQuest Dissertations Publishing, Pittsburgh, PA, United States of America, 2007.

[53] B. S. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, “libMesh: A C++ library for parallel adaptive mesh refinement/coarsening simulations,” *Eng. Comput.*, vol. 22, no. 3–4, pp. 237–

254, 2006.

[54] T. Heister, “A massively parallel finite element framework with application to incompressible flows,” Georg-August-Universität Göttingen, Göttingen, Niedersachsen, Germany, 2011.

[55] R. Löhner, *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*, 2nd ed. John Wiley & Sons, Ltd., 2008.

[56] T. Plewa, T. Linde, and V. G. Weirs, Eds., *Adaptive Mesh Refinement - Theory and Applications*, vol. 41. Springer Berlin Heidelberg, 2003.

[57] P. M. Pauskar, K. Sawamiphakdi, and D. Q. Jin, “Static implicit vs. dynamic explicit finite element analysis for ring rolling process modeling,” in *Proceedings of the 8th International Conference on Numerical Methods in Industrial Forming Processes*, 2004, vol. 712, pp. 412–417.

[58] A. A. Johnson, “Dynamic-mesh CFD and its application to flapping-wing micro-air vehicles,” in *25th Army Science Conference*, 2006.

[59] B. S. Kirk, “Adaptive finite element simulation of flow and transport applications on parallel computers,” ProQuest Information and Learning Company, Austin, TX, United States of America, 2007.

[60] S. Plimpton, S. Attaway, B. A. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner, “Parallel transient dynamics simulations: Algorithms for contact detection and smoothed particle hydrodynamics,” *J. Parallel Distrib. Comput.*, vol. 50, no. 1–2, pp. 104–122, 1998.

[61] C. H. Walshaw, M. Cross, and K. McManus, “Multiphase mesh partitioning,” *Appl. Math. Model.*, vol. 25, no. 2, pp. 123–140, Dec. 2000.

[62] O. C. Zienkiewicz and R. L. Taylor, *The finite element method: Solid mechanics*, 5th ed., vol. 1. Oxford, United Kingdom: Butterworth-Heinemann, 2000.

[63] R. Diekmann, D. Meyer, and B. Monien, “Parallel decomposition of unstructured FEM-meshes,” in *Parallel Algorithms for Irregularly Structured Problems*, vol. 980, no. 1995, A. Ferreira and J. Rolim, Eds. Springer Berlin Heidelberg, 1995, pp. 199–215.

[64] T. N. Bui and C. Jones, “Finding good approximate vertex and edge partitions is NP-hard,” *Inf. Process. Lett.*, vol. 42, no. 3, pp. 153–159, May 1992.

[65] R. Diekmann, B. Monien, and R. Preis, “Using helpful sets to improve graph bisections,” in *Interconnection networks and mapping and scheduling parallel computations*, vol. 21, D. F. Hsu, A. L. Rosenberg, and D. Sotteau, Eds. American Mathematical Society, 1995, pp. 57–73.

[66] C. Farhat, “A simple and efficient automatic FEM domain decomposer,” *Comput. Struct.*, vol. 28, no. 5, pp. 579–602, 1988.

[67] B. A. Hendrickson and R. Leland, “An improved spectral graph partitioning algorithm for mapping parallel computations,” *SIAM J. Sci. Comput.*, vol. 16, no. 2, pp. 452–469, 1995.

[68] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1999.

[69] A. Pothen, H. D. Simon, and K.-P. P. Liou, “Partitioning sparse matrices with eigenvectors of graphs,” *SIAM J. Sci. Comput.*, vol. 11, no. 3, pp. 430–452, 1990.

[70] H. D. Simon, “Partitioning of unstructured problems for parallel processing,” *Comput. Syst. Eng.*, vol. 2, no. 2–3, pp. 135–148, 1991.

[71] C. M. Fiduccia and R. M. Mattheyses, “A linear-time heuristic for improving network partitions,” in *19th Conference on Design Automation*, 1982, pp. 175–181.

[72] B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[73] L. Oliker and R. Biswas, “PLUM : Parallel load balancing for adaptive unstructured meshes,” *J. Parallel Distrib. Comput.*, vol. 52, no. 2, pp. 150–177, 1998.

[74] R. Diekmann, R. Preis, F. Schlimbach, and C. H. Walshaw, “Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM,” *Parallel Comput.*, vol. 26, no. 12, pp. 1555–1581, 2000.

[75] M. H. Willebeek-LeMair and A. P. Reeves, “Strategies for dynamic load balancing on highly parallel computers,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979–993, 1993.

[76] B. N. Delaunay, “Sur la sphère vide,” in *Proceedings of the International Mathematical Congress*, 1924, vol. 1, pp. 695–700.

[77] B. N. Delaunay, “Sur la sphère vide,” *Bull. l’Académie des Sci. l’URSS. Cl. des Sci. mathématiques Nat.*, no. 6, pp. 793–800, 1934.

[78] H. Meyerhenke, B. Monien, and S. Schamberger, “Accelerating shape optimizing load balancing for parallel FEM simulations by algebraic multigrid,” in *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium*, 2006, p. 10 pp.

[79] C. H. Walshaw and M. Cross, “Mesh partitioning: A multilevel balancing and refinement algorithm,” *SIAM J. Sci. Comput.*, vol. 22, no. 1, pp. 63–80, Jun. 2000.

[80] C. H. Walshaw and M. Cross, “Parallel optimisation algorithms for multilevel mesh partitioning,” *Parallel Comput.*, vol. 26, no. 12, pp. 1635–1660, Nov. 2000.

[81] C.-E. Bichot, “Élaboration d’une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission. Application au découpage de l’espace aérien,” Institut National Polytechnique de Toulouse, Toulouse, Languedoc-Roussillon-Midi-Pyrénées, France, 2007.

[82] D. Vanderstraeten and R. Keunings, “Optimized partitioning of unstructured finite element meshes,” *Int. J. Numer. Methods Eng.*, vol. 38, no. 3, pp. 433–450, 1995.

[83] C. Farhat, S. Lanteri, and H. D. Simon, “TOP/DOMDEC - A software tool for mesh partitioning and parallel processing,” *Comput. Syst. Eng.*, vol. 6, no. 1, pp. 13–26, Feb. 1995.

[84] C.-E. Bichot, “Metaheuristics versus spectral and multilevel methods applied on an air traffic control problem,” in *12th IFAC Symposium on Information Control Problems in Manufacturing*, 2006, vol. 39, no. 3, pp. 493–498.

[85] S. T. Barnard and H. D. Simon, “Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems,” *Concurr. Pract. Exp.*, vol. 6, no. 2, pp. 101–117, Apr. 1994.

[86] C. H. Walshaw, M. Cross, and M. G. Everett, “Parallel dynamic graph partitioning for adaptive unstructured meshes,” *J. Parallel Distrib. Comput.*, vol. 47, no. 2, pp. 102–108, 1997.

[87] Y. F. Hu and R. J. Blake, “An Optimal dynamic load balancing algorithm,” Daresbury, United Kingdom, 1995.

[88] G. Cybenko, “Dynamic load balancing for distributed memory multiprocessors,” *J. Parallel Distrib. Comput.*, vol. 7, no. 2, pp. 279–301, 1989.

[89] C. H. Walshaw, M. Cross, and M. G. Everett, “A localized algorithm for optimizing unstructured mesh partitions,” *Int. J. High Perform. Comput. Appl.*, vol. 9, no. 4, pp. 280–295, 1995.

[90] G. Karypis and V. Kumar, “Multilevel k-way partitioning scheme for irregular graphs,” *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 96–129, Jan. 1998.

[91] A. Abou-Rjeili and G. Karypis, “Multilevel algorithms for partitioning power-law graphs,” in *International Parallel and Distributed Processing Symposium*, 2006, p. 10 pp.

[92] A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J.-M. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose, and C. H. Walshaw, “Dynamic load-balancing of finite element applications with the DRAMA library,” *Appl. Math. Model.*, vol. 25, no. 2, pp. 83–98, Dec. 2000.

[93] B. A. Hendrickson, “Graph partitioning and parallel solvers: Has the emperor no clothes?,” in *Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, 1998, pp. 218–225.

[94] Ü. V. Çatalyürek and C. Aykanat, “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 7, pp. 673–693, 1999.

[95] B. Vastenhouw and R. H. Bisseling, “A two-dimensional data distribution method for parallel sparse matrix-vector multiplication,” *SIAM Rev.*, vol. 47, no. 1, pp. 67–95, 2005.

[96] C. Chang, T. Kurc, A. Sussman, Ü. V. Çatalyürek, and J. Saltz, “A hypergraph-based workload partitioning strategy for parallel data aggregation,” in *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, 2001.

[97] M. M. Ozdal and C. Aykanat, “Hypergraph models and algorithms for data-pattern-based clustering,” *Data Min. Knowl. Discov.*, vol. 9, no. 1, pp. 29–57, 2004.

[98] A. E. Caldwell, A. B. Kahng, and I. L. Markov, “Design and implementation of move-based heuristics for VLSI hypergraph partitioning,” *J. Exp. Algorithmics*, vol. 5, no. 2000, 2000.

[99] P. Miettinen, M. Honkala, and J. Roos, “Using METIS and hMETIS algorithms in circuit partitioning,” Espoo, Finland, 2006.

[100] H. Meyerhenke, B. Monien, and S. Schamberger, “Graph partitioning and disturbed diffusion,” *Parallel Comput.*, vol. 35, no. 10–11, pp. 544–569, 2009.

[101] P. Sanders and C. Schulz, “High quality graph partitioning,” in *Graph partitioning and graph clustering*, vol. 588, D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, Eds. Providence, RI, United States of America: American Mathematical Society, 2013, pp. 1–18.

[102] G. Karypis and V. Kumar, “Multilevel graph partitioning schemes,” in *Proceedings of the 1995 International Conference on Parallel Processing*, 1995, pp. 113–122.

[103] G. Karypis and V. Kumar, “A parallel algorithm for multilevel graph partitioning and sparse matrix ordering,” *J. Parallel Distrib. Comput.*, vol. 48, no. 1998, pp. 71–85, 1998.

[104] B. A. Hendrickson and T. G. Kolda, “Graph partitioning models for parallel computing,” *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, Nov. 2000.

[105] S. Rajamanickam and E. G. Boman, “An evaluation of the Zoltan parallel graph and hypergraph partitioners,” in *10th DIMACS Implementation Challenge*, 2012.

[106] C. Schulz, “High quality graph partitioning,” Karlsruher Instituts für Technologie, Karlsruhe, Baden-Württemberg, Deutschland, 2013.

[107] J. Kim, I. Hwang, Y.-H. Kim, and B.-R. Moon, “Genetic approaches for graph partitioning: A survey,” in *GECCO ’11 Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 473–480.

[108] H. Meyerhenke and S. Schamberger, “Balancing parallel adaptive FEM computations by solving systems of linear equations,” in *Euro-Par 2005 Parallel Processing*, vol. 3648, no. 2005, J. Cunha and P. Medeiros, Eds. Springer Berlin / Heidelberg, 2005, pp. 624–624.

[109] G. Horton, “A multi-level diffusion method for dynamic load balancing,” *Parallel Comput.*, vol. 19, no. 2, pp. 209–218, 1993.

[110] S. Schamberger, “A shape optimizing load distribution heuristic for parallel adaptive FEM computations,” in *Parallel Computing Technologies*, vol. 3606, no. 2005, V. Malyshkin, Ed. Springer Berlin / Heidelberg, 2005, pp. 263–277.

[111] C.-J. Liao, “Efficient partitioning and load-balancing methods for finite element graphs on distributed memory multicomputers,” Feng Chia University, Seatwen, Taichung, Taiwan, 1999.

[112] R. Elsässer, B. Monien, and R. Preis, “Diffusion schemes for load balancing on heterogeneous networks,” *Theory Comput. Syst.*, vol. 35, no. 3, pp. 305–320, 2002.

[113] S. Schamberger, “On partitioning FEM graphs using diffusion,” in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004, p. 277.

[114] A. Heirich and S. Taylor, “A parabolic load balancing method,” Pasadena, CA, United States of America, 1994.

[115] B. A. Hendrickson and R. Leland, “A multilevel algorithm for partitioning graphs,” in *Supercomputing ’95 Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, 1995, p. Article no. 28.

[116] B. A. Hendrickson and R. Leland, “Chaco: Software for partitioning graphs,” 2015. [Online]. Available: https://cfwebprod.sandia.gov/cfdocs/CompResearch/templates/insert/softwre.cfm?sw=36. [Accessed: 16-Jan-2018].

[117] B. A. Hendrickson and R. Leland, “The Chaco user’s guide: Version 2.0,” Albuquerque, NM, United States of America, 1995.

[118] T. Goehring and Y. Saad, “Heuristic algorithms for automatic graph partitioning,” Minneapolis, MN, United States of America, 1994.

[119] Y. Linde, A. Buzo, and R. M. Gray, “An algorithm for vector quantizer design,” *IEEE Trans. Commun.*, vol. 28, no. 1, pp. 84–95, 1980.

[120] M. T. Heath and P. Raghavan, “A cartesian parallel nested dissection algorithm,” *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 1, pp. 235–253, 1995.

[121] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, “Automatic mesh partitioning,” in *Graphs Theory and Sparse Matrix Computation*, vol. 56, A. George, J. R. Gilbert, and J. W. H. Liu, Eds. New York, NY, United States of America: Springer New York, 1993, pp. 57–84.

[122] M. J. Berger and S. H. Bokhari, “A partitioning strategy for nonuniform problems on multiprocessors,” *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 570–580, May 1987.

[123] V. E. Taylor and B. Nour-Omid, “A study of the factorization fill-in for a parallel implementation of the finite element method,” *Int. J. Numer. Methods Eng.*, vol. 37, no. 22, pp. 3809–3823, 1994.

[124] G. Karypis and V. Kumar, “Analysis of multilevel graph partitioning,” in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, 1995, p. 29.

[125] G. Karypis and V. Kumar, “Analysis of multilevel graph partitioning,” Minneapolis, MN, United States of America, 1995.

[126] H. D. Simon and S.-H. Teng, “How good is recursive bisection,” Moffett Field, CA, United States of America, 1993.

[127] FIZ Karlsruhe and Zuse-Institut Berlin, “swMATH,” 2011. [Online]. Available: http://www.swmath.org/. [Accessed: 23-Jun-2018].

[128] M. A. Bhandarkar and L. V. Kalé, “A parallel framework for explicit FEM,” in *Proceedings of the 7th International Conference on High Performance Computing*, 2000, pp. 385–394.

[129] J. R. Stewart and H. C. Edwards, “The SIERRA framework for developing advanced parallel mechanics applications,” in *Large-Scale PDE-Constrained Optimization*, vol. 30, no. 2003, L. T. Biegler, M. Heinkenschloss, O. Ghattas, and B. van B. Waanders, Eds. Berlin, Germany: Springer Berlin Heidelberg, 2003, pp. 301–315.

[130] Sandia National Laboratories, “Trilinos,” 2011. [Online]. Available: https://trilinos.org/. [Accessed: 19-Jan-2018].

[131] A. Logg and G. N. Wells, “DOLFIN: Automated finite element computing,” *ACM Trans. Math.*

*Softw.*, vol. 37, no. 2, p. 28, Apr. 2010.

[132] S. Turek, D. Göddeke, C. Becker, S. H. M. Buijssen, and H. Wobker, "FEAST - Realization of hardware-oriented numerics for HPC simulations with finite elements," *Concurr. Comput. Pract. Exp.*, vol. 22, no. 16, pp. 2247–2265, 2010.

[133] H. P. Langtangen, *Computational partial differential equations: Numerical methods and Diffpack programming*, 2nd ed., vol. 1. Berlin, Germany: Springer-Verlag, 2003.

[134] K. Long, R. Kirby, J. Benk, B. van B. Waanders, and P. Boggs, "Sundance," 2011. [Online]. Available: http://www.math.ttu.edu/~kelong/Sundance/html/. [Accessed: 19-Jan-2018].

[135] P. Dular and C. Geuzaine, "GetDP: A general environment for the treatment of discrete problems," 2011. [Online]. Available: http://getdp.info/. [Accessed: 19-Jan-2018].

[136] O. Pironneau, F. Hecht, A. Le Hyaric, and J. Morice, "FreeFEM++," 2011. [Online]. Available: http://www.freefem.org/. [Accessed: 19-Jan-2018].

[137] C. Prud'homme, V. Chabannes, and Feel++ Group, "Feel++," 2011. [Online]. Available: https://forge.imag.fr/projects/life/. [Accessed: 19-Jan-2018].

[138] C. Prud'homme, "Life: Overview of a unified C++ implementation of the finite and spectral element methods in 1D, 2D and 3D," in *Applied Parallel Computing. State of the Art in Scientific Computing*, vol. 4699, no. 2007, B. Kågström, E. Elmroth, J. Dongarra, and J. Waśniewski, Eds. Berlin, Germany: Springer-Verlag, 2007, pp. 712–721.

[139] X. Jiao, X.-Y. Li, and X. Ma, "SIFFEA: Scalable integrated framework for finite element analysis," in *Computing in Object-Oriented Parallel Environments*, vol. 1732, no. 1999, S. Matsuoka and M. Tholburn, Eds. Berlin, Germany: Springer-Verlag, 1999, pp. 84–95.

[140] A. M. Bruaset and H. P. Langtangen, "A comprehensive set of tools for solving partial differential equations; Diffpack," in *Numerical Methods and Software Tools in Industrial Mathematics*, M. Dæhlen and A. Tveito, Eds. Boston, MA, United States of America: Birkhäuser Boston, 1997, pp. 61–90.

[141] Y. Renard and J. Pommier, "GetFEM++," 2012. [Online]. Available: http://getfem.org/. [Accessed: 19-Jan-2018].

[142] B. Patzák and Z. Bittnar, "Design of object oriented finite element code," *Adv. Eng. Softw.*, vol. 32, no. 10–11, pp. 759–767, 2001.

[143] A. Logg, "Automating the finite element method," *Arch. Comput. Methods Eng.*, vol. 14, no. 2, pp. 93–138, 2007.

[144] F. Hecht, "New development in FreeFEM++," *J. Numer. Math.*, vol. 20, no. 3–4, pp. 251–265, Dec. 2012.

[145] W. Bangerth, R. Hartmann, and G. Kanschat, "deal.II - A general-purpose object-oriented finite element library," *ACM Trans. Math. Softw.*, vol. 33, no. 4, Aug. 2007.

[146] L. V. Kalé, R. D. Skeel, M. A. Bhandarkar, R. K. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "NAMD2: Greater scalability for parallel

molecular dynamics," *J. Comput. Phys.*, vol. 151, no. 1, pp. 283–312, May 1999.

[147] S. Turek, *Efficient solvers for incompressible flow problems: An algorithmic and computational approach*, 1st ed. Berlin, Germany: Springer-Verlag, 1999.

[148] inuTech GmbH, "Diffpack," 2018. [Online]. Available: http://diffpack.de/. [Accessed: 25-Jun-2018].

[149] inuTech GmbH, "inuTech," 2018. [Online]. Available: http://inutech.de/. [Accessed: 25-Jun-2018].

[150] ANSANSYS Inc., "ANSYS - Simulation Driven Product Development," 2012. [Online]. Available: http://www.ansys.com/. [Accessed: 19-Jan-2018].

[151] ESI Group, "ESI Group," 2012. [Online]. Available: http://www.esi-group.com/. [Accessed: 18-Jan-2018].

[152] TRANSVALOR S.A., "Transvalor: Material forming simulation," 2012. [Online]. Available: http://www.transvalor.com/. [Accessed: 18-Jan-2018].

[153] Cuylaerts Engineering, "Analysis3D Structural Engineering Software," 2018. [Online]. Available: http://www.cuylaerts.net/Analysis3D.html. [Accessed: 25-Jun-2018].

[154] University of Utah and Columbia University, "FEBio software suite," 2005. [Online]. Available: https://febio.org/. [Accessed: 25-Jun-2018].

[155] Precise Simulation Ltd., "FEATool Multiphysics," 2013. [Online]. Available: https://www.featool.com/. [Accessed: 25-Jun-2018].

[156] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale, "DRAMA: A library for parallel dynamic load balancing of finite element applications," in *Euro-Par 1999 Parallel Processing*, vol. 1685, no. 1999, P. Amestoy, P. Berger, M. Daydé, D. Ruiz, I. Duff, V. Frayssé, and L. Giraud, Eds. Springer Berlin / Heidelberg, 1999, pp. 313–316.

[157] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, and C. Vaughan, "Zoltan data management services for parallel dynamic applications," *Comput. Sci. Eng.*, vol. 4, no. 2, pp. 90–96, 2002.

[158] Sandia National Laboratories, "Zoltan: Parallel partitioning, load balancing and data-management services," 2012. [Online]. Available: http://www.cs.sandia.gov/Zoltan/. [Accessed: 18-Jan-2018].

[159] J. Faik, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco, "DRUM: The dynamic resource utilization model," 2012. [Online]. Available: http://j.teresco.org/research/drum/. [Accessed: 18-Jan-2018].

[160] J. Faik, "A model for resource-aware load balancing on heterogeneous and non-dedicated clusters," Rensselaer Polytechnic Institute, Troy, NY, United States of America, 2005.

[161] W. C. Chu, D.-L. Yang, J.-C. Yu, and Y.-C. Chung, "UMPAL An unstructured mesh partitioner and load balancer on world wide web," *J. Inf. Sci. Eng.*, vol. 17, no. 4, pp. 595–614, 2001.

[162] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK User's guide*, 1st ed. Philadelphia, PA, United States of America: Society for Industrial and Applied Mathematics, 1979.

[163] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, "LINPACK," 2018. [Online]. Available: http://www.netlib.org/linpack/. [Accessed: 18-Jan-2018].

[164] C. H. Walshaw, "The serial JOSTLE library user guide : Version 3.0," London, United Kingdom, 2002.

[165] F. Pellegrini, "SCOTCH: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package," 2012. [Online]. Available: http://www.labri.u-bordeaux.fr/perso/pelegrin/scotch/. [Accessed: 18-Jan-2018].

[166] F. Pellegrini, "Scotch and libScotch 5.1 user's guide," Talence, France, 2010.

[167] F. Pellegrini and J. Roman, "SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *High-Performance Computing and Networking*, vol. 1067, no. 1996, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, Eds. Springer Berlin Heidelberg, 1996, pp. 493–498.

[168] R. Baños Navarro and C. Gil Montoya, "Graph and mesh partitioning: An overview of the current state-of-the-art," in *Mesh Partitioning Techniques and Domain Decomposition Methods*, F. Magoulès, Ed. Stirlingshire, United Kingdom: Saxe-Coburg Publications, 2007, pp. 1–26.

[169] R. Battiti and A. A. Bertossi, "Greedy, prohibition, and reactive heuristics for graph partitioning," *IEEE Trans. Comput.*, vol. 48, no. 4, pp. 361–385, Apr. 1999.

[170] C. H. Walshaw, "JOSTLE - Graph partitioning software," 2012. [Online]. Available: http://staffweb.cms.gre.ac.uk/~wc06/jostle/. [Accessed: 16-Jan-2018].

[171] C. H. Walshaw and M. Cross, "JOSTLE: Parallel multilevel graph-partitioning software - an overview," in *Mesh partitioning techniques and domain decomposition methods*, F. Magoulès, Ed. Stirlingshire, United Kingdom: Saxe-Coburg Publications, 2007, pp. 27–58.

[172] C. H. Walshaw, "FocusWare NetWorks MNO (Mobile Network Optimisation)," 2018. [Online]. Available: http://staffweb.cms.gre.ac.uk/~wc06/focusware/. [Accessed: 16-Jan-2018].

[173] R. Preis, "PARTY Partitioning library," 2018. [Online]. Available: http://www2.cs.uni-paderborn.de/cs/ag-monien/PERSONAL/ROBSY/party.html. [Accessed: 16-Jan-2018].

[174] R. Preis, "The PARTY Graphpartitioning - Library - User manual - Version 1.99," Paderborn, Germany, 1998.

[175] R. Preis and R. Diekmann, "PARTY - A software library for graph partitioning," in *Advances in Computational Mechanics with Parallel and Distributed Processing*, 1997, pp. 63–71.

[176] R. Diekmann, B. Monien, and R. Preis, "Using helpful sets to improve graph bisections," Paderborn, Germany, 1994.

[177] J. Hromkovič and B. Monien, "The bisection problem for graphs of degree 4 (configuring transputer systems)," in *Mathematical Foundations of Computer Science 1991*, vol. 520, A.

Tarlecki, Ed. Springer Berlin / Heidelberg, 1991, pp. 211–220.

[178] B. Monien and R. Preis, “Upper bounds on the bisection width of 3- and 4-regular graphs,” in *Mathematical Foundations of Computer Science 2001*, vol. 2136, no. 2001, J. Sgall, A. Pultr, and P. Kolman, Eds. Springer Berlin Heidelberg, 2001, pp. 524–536.

[179] F. Pellegrini, “Static mapping by dual recursive bipartitioning of process and architecture graphs,” in *Proceedings of the 1994 Scalable High-Performance Computing Conference*, 1994, pp. 486–493.

[180] G. Karypis and V. Kumar, “Multilevel algorithms for multi-constraint graph partitioning,” in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, 1998.

[181] Y. F. Hu, R. J. Blake, and D. R. Emerson, “An optimal migration algorithm for dynamic load balancing,” *Concurr. Pract. Exp.*, vol. 10, no. 6, pp. 467–483, 1998.

[182] G. Karypis, “hMETIS - Hypergraph & circuit partitioning,” 2012. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/hmetis/overview. [Accessed: 16-Jan-2018].

[183] G. Karypis and V. Kumar, “hMETIS - A hypergraph partitioning package - Version 1.5.3,” Minneapolis, MN, United States of America, 1998.

[184] Ü. V. Çatalyürek, “PaToH v3.2,” 2012. [Online]. Available: http://bmi.osu.edu/~umit/software.html. [Accessed: 19-Jan-2018].

[185] Ü. V. Çatalyürek and C. Aykanat, “PaToH: Partitioning tool for hypergraphs,” Columbus, OH, United States of America, 2011.

[186] R. H. Bisseling, “Mondriaan for sparse matrix partitioning,” 2012. [Online]. Available: http://www.staff.science.uu.nl/~bisse101/Mondriaan/mondriaan.html. [Accessed: 19-Jan-2018].

[187] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, “Multilevel hypergraph partitioning: Application in VLSI domain,” in *34th Design and Automation Conference*, 1997, pp. 526–529.

[188] Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar, “UMPa: A multi-objective, multi-level partitioner for communication minimization,” in *Graph partitioning and graph clustering*, vol. 588, D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, Eds. Providence, RI, United States of America: American Mathematical Society, 2013, pp. 53–66.

[189] M. Deveci, K. Kaya, B. Uçar, and Ü. V. Çatalyürek, “Hypergraph partitioning for multiple communication cost metrics: Model and methods,” *J. Parallel Distrib. Comput.*, vol. 77, pp. 69–83, 2015.

[190] D. P. Woodruff and Q. Zhang, “When distributed computation is communication expensive,” in *Distributed Computing*, vol. 8205, Y. Afek, Ed. Springer Berlin Heidelberg, 2013, pp. 16–30.

[191] Texas A&M University, “The SuiteSparse matrix collection,” 2018. [Online]. Available: https://sparse.tamu.edu/. [Accessed: 22-May-2018].

[192] Center for Discrete Mathematics and Theoretical Computer Science, “Graphs from numerical simulations,” 2011. [Online]. Available: https://www.cc.gatech.edu/dimacs10/index.shtml. [Accessed: 22-May-2018].

[193] S. Y. Chan, T. C. Ling, and E. Aubanel, “The impact of heterogeneous multi-core clusters on graph partitioning: An empirical study,” *Cluster Comput.*, vol. 15, no. 3, pp. 281–302, 2012.

[194] D. Beckett and University of Kent at Canterbury, “Internet parallel computing archive,” 2000. [Online]. Available: http://wotug.org/parallel/. [Accessed: 23-May-2018].

[195] C. H. Walshaw, “The graph partitioning archive,” 2016. [Online]. Available: http://chriswalshaw.co.uk/partition/. [Accessed: 23-May-2018].

[196] E. N. Gilbert, “Random plane networks,” *J. Soc. Ind. Appl. Math.*, vol. 9, no. 4, pp. 533–543, Dec. 1961.

[197] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science (80-. ).*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[198] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” *Rev. Mod. Phys.*, vol. 74, no. 1, pp. 47–97, Jan. 2002.

[199] G. U. Yule, “A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F.R.S.,” *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 213, no. 402–410, pp. 21–87, Jan. 1925.

[200] H. A. Simon, “On a class of skew distribution functions,” *Biometrika*, vol. 42, no. 3/4, pp. 425–440, Dec. 1955.

[201] M. E. J. Newman, D. J. Watts, and S. H. Strogatz, “Random graph models of social networks,” in *Proceedings of the National Academy of Sciences*, 2002, vol. 99, no. Supplement 1, pp. 2566–2572.

[202] E. N. Gilbert, “Random graphs,” *Ann. Math. Stat.*, vol. 30, no. 4, pp. 1141–1144, 1959.

[203] P. Erdős and A. Rényi, “On random graphs I,” *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[204] University of Florida, “University of Florida sparse matrix collection,” 2015. [Online]. Available: http://faculty.cse.tamu.edu/davis/research.html. [Accessed: 23-May-2018].

# Appendix A.

# HEM, SHEM and DSHEM

HEM, SHEM and DSHEM are all algorithms designed to generate a maximal matching of a graph. They have many points in common since DSHEM is based on SHEM, which in turn is based in HEM. A number of algorithms were implemented in the first release of METIS; HEM among them. Theoretical and experimental studies have shown the efficiency of HEM and SHEM making them the two main algorithms in later versions of METIS. Finally, SHEM demonstrated its superiority over HEM and has been the de facto algorithm in METIS. DSHEM takes the advantages of SHEM and incorporates new ideas to trick the matching process in order to obtain better partitions for FEM applications. It considers bidirectional edges to represent more accurately the communication involved during FEM simulations. The rest of this section describes in detail all three algorithms.

## A.1. HEM Detailed Algorithm

HEM is simple, effective and very fast; important characteristics in graph partitioning software. It was implemented in METIS since its inception and produced good results during the matching process. It was part of the partition library until version 5.0 when it was finally replaced by the enhanced version of SHEM, the next generation of the algorithm. SHEM produces consistently better partitions compared to its predecessor; the reason why HEM was abandoned. The next paragraphs describe, in detail, the HEM algorithm in Table A.1.

**Table A.1: HEM detailed algorithm**

| **Algorithm** | Heavy Edge Matching |
|---|---|
| **Input:** | Maximum weight allowed for a vertex $maxvwgt$ |
| | Structure with information of the graph $graph$ |
| **Output:** | Array with maximal matching of the graph $match$ |
| | Number of new coarse vertices $cnvtxs$ |

```
1:  procedure MATCH_HEM(maxvwgt, graph)
2:    nvtxs ← graph.nvtxs
3:    xadj ← graph.xadj
4:    vwgt ← graph.vwgt
5:    adjncy ← graph.adjncy
6:    adjwgt ← graph.adjwgt
7:    match ← INITIALIZE(nvtxs, UNMATCHED)
```

```
8:    perm ← RANDOM_PERMUTE(nvtxs)
9:    cnvtxs ← 0
10:   for ii ← 0 to nvtxs − 1 step 1 do
11:     i ← perm[ii]
12:     if match[i] = UNMATCHED then
13:       maxidx ← i
14:       maxwgt ← 0
15:       for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
16:         k ← adjncy[j]
17:         if match[k] = UNMATCHED and
18:         maxwgt < adjwgt[j] and
19:         (vwgt[i] + vwgt[k]) ≤ maxvwgt then
20:           maxidx ← k
21:           maxwgt ← adjwgt[j]
22:         end if
23:       end for
24:       cnvtxs ← cnvtxs + 1
25:       match[i] ← maxidx
26:       match[maxidx] ← i
27:     end if
28:   end for
29:   return match, cnvtxs
30: end procedure
```

Two main arguments are received by the algorithm; namely $maxvwgt$ and $graph$. $maxvwgt$ specifies the maximum weight allowed for a coarse vertex; two vertices will only be matched if their combined weight does not exceed $maxvwgt$. $graph$ is a structure which contains all information related to the graph, such as the number of vertices and edges, their weights and adjacency information, among others. The algorithm returns the array $match$ with the matching information and $cnvtxs$ with the number of coarse vertices.

The first part of the algorithm is for initialization purposes. Lines 2 to 6 initialize local variables with information of the graph; $nvtxs$ contains the number of vertices, $xadj$ and $adjncy$ the adjacency information, $vwgt$ the weights of the vertices and $adjwgt$ the weights of the edges. Line 7 initializes the array $match$ with $UNMATCHED$; all vertices are unmatched at this point of the execution. Then the array $perm$ is also initialized, in line 8; it contains a random permutation used to visit all vertices in that particular sequence. Finally, the last initialization is that of $cnvtxs$ that will contain the number of coarse vertices after the matching process.

The for loop of lines 10 to 28 visits the vertices of the graph according to the random permutation in $perm$. If vertex $i$ is $UNMATCHED$, in line 12, then an adjacent vertex is going to be chosen to match with it. Vertex $i$ is first matched to itself in lines 13 and 14, in case no other option is available; this is, vertex $i$ is provisionally matched to vertex $maxidx$ (itself) and the weight of the edge between them $maxwgt$ is 0 (as no edge links the vertex to itself). The adjacent vertices of vertex $i$ are then visited,

using the for loop in lines 15 to 23, in order to select the adjacent vertex $k$ with the heaviest edge to match with vertex $i$. In lines 17 to 19 a decision is made, if adjacent vertex $k$ is $UNMATCHED$ and the edge $(i,k)$ is heavier than the current weight in $maxwgt$ and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed specified by $maxvwgt$ then vertex $i$ is matched to vertex $k$ in lines 20 and 21. The final and definitive matching of vertex $i$ is done after all its adjacent vertices have been evaluated in the loop; the number of coarse vertices is updated in line 24, vertex $i$ is matched to the vertex in $maxidx$, and vice versa, in lines 25 and 26. Finally, line 29 returns the array with the matching information and the number of coarse vertices in the resulting coarse graph.

## A.2. SHEM Detailed Algorithm

SHEM has been improved over the time on every release of METIS. It is a modified version of HEM with one key difference: the vertices are visited in a sorted manner according to their degree. The very first implementation of SHEM visited the vertices with the highest degree first, whereas more recent versions do it in the opposite order. Upgrades to the algorithm have been gradually implemented to boost the quality of the partition. The *original version* of SHEM, depicted in Table A.2, is in fact a newer version of the algorithm; however, more suited for the purpose of describing SHEM and DSHEM. This version is referred as the original version of the SHEM algorithm throughout this work. The computation of vertex degrees, the subsequent sorting of the vertices, and the matching of islands are the core differences with HEM. The light gray text in Table A.2 is the shared code between HEM and SHEM, the black text represents the new or modified code present in SHEM. Lines 8 to 17 calculate the vertex degrees and sort the vertices using that information. Lines 19 to 38 find suitable matches for the island vertices. The next paragraphs describe, in detail, the original version of the SHEM algorithm.

**Table A.2: SHEM detailed algorithm (original version)**

| **Algorithm** | Sorted Heavy Edge Matching (original version) |
|---|---|
| **Input:** | Maximum weight allowed for a vertex $maxvwgt$ |
| | Structure with information of the graph $graph$ |
| **Output:** | Array with maximal matching of the graph $match$ |
| | Number of new coarse vertices $cnvtxs$ |

```
1:  procedure MATCH_SHEM(maxvwgt, graph)
2:    nvtxs ← graph.nvtxs
3:    xadj ← graph.xadj
4:    vwgt ← graph.vwgt
5:    adjncy ← graph.adjncy
6:    adjwgt ← graph.adjwgt
7:    match ← INITIALIZE(nvtxs, UNMATCHED)
8:    tperm ← RANDOM_PERMUTE(nvtxs)
9:    avgdegree ← 0.7 * (xadj[nvtxs]/nvtxs)
10:   for i ← 0 to nvtxs − 1 step 1 do
11:     if (xadj[i + 1] − xadj[i]) > avgdegree then
```

```
12:       degrees[i] ← avgdegree
13:     else
14:       degrees[i] ← xadj[i + 1] − xadj[i]
15:     end if
16:   end for
17:   perm ← BUCKET_SORT_KEYS_INC(nvtxs, degrees, tperm)
18:   cnvtxs ← 0
19:   for ii ← 0 to nvtxs − 1 step 1 do
20:     i ← perm[ii]
21:     if match[i] = UNMATCHED then
22:       if xadj[i] < xadj[i + 1] then
23:         break
24:       end if
25:       maxidx ← i
26:       for j ← nvtxs − 1 to ii + 1 step −1 do
27:         k ← perm[j]
28:         if match[k] = UNMATCHED and
29:         xadj[k] < xadj[k + 1] then
30:           maxidx ← k
31:           break
32:         end if
33:       end for
34:       cnvtxs ← cnvtxs + 1
35:       match[i] ← maxidx
36:       match[maxidx] ← i
37:     end if
38:   end for
39:   for ii ← ii to nvtxs − 1 step 1 do
40:     i ← perm[ii]
41:     if match[i] = UNMATCHED then
42:       maxidx ← i
43:       maxwgt ← 0
44:       for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
45:         k ← adjncy[j]
46:         if match[k] = UNMATCHED and
47:         maxwgt < adjwgt[j] and
48:         (vwgt[i] + vwgt[k]) ≤ maxvwgt then
49:           maxidx ← k
50:           maxwgt ← adjwgt[j]
51:         end if
52:       end for
```

```
53:          cnvtxs ← cnvtxs + 1
54:          match[i] ← maxidx
55:          match[maxidx] ← i
56:        end if
57:      end for
58:      return match, cnvtxs
59:  end procedure
```

Similar to HEM, two main arguments are received by SHEM; namely $maxvwgt$ and $graph$. $maxvwgt$ specifies the maximum weight allowed for a coarse vertex; two vertices will only be matched if their combined weight does not exceed $maxvwgt$. Nevertheless, the islands are excluded from this restriction. $graph$ is a structure which contains all information related to the graph, such as the number of vertices and edges, their weights and adjacency information, among others. The algorithm returns the array $match$ with the matching information and $cnvtxs$ with the number of coarse vertices.

The first part of the algorithm is for initialization purposes. Lines 2 to 6 initialize local variables with information of the graph; $nvtxs$ contains the number of vertices, $xadj$ and $adjncy$ the adjacency information, $vwgt$ the weights of the vertices and $adjwgt$ the weights of the edges. Line 7 initializes the array $match$ with $UNMATCHED$; all vertices are unmatched at this point of the execution. Then the temporal array $tperm$ is also initialized, in line 8; it contains a random permutation used to sort the vertices later in line 17. The constrained average degree in $avgdegree$ is limited to 70% of the actual value; all vertices with a degree over $avgdegree$ are treated equally. The for loop, in lines 10 to 16, computes the degrees for all vertices limiting their values to a maximum of $avgdegree$. Then, in line 17, the array $perm$ is populated with a sorted permutation of the vertices according to their degree previously computed. Finally, the last initialization, in line 18, is that of $cnvtxs$ that will contain the number of coarse vertices after the matching process.

The for loop of lines 19 to 38 treats the islands if they exist; i.e., vertices without edges. Contrary to HEM, these vertices are now properly matched leading to better partitions. The islands would be located in the first positions of the array $perm$. Line 21 checks if vertex $i$ is $UNMATCHED$, if so, a vertex with high degree will be selected to match with it. If vertex $i$ is not an island, i.e., its degree is not 0, then the for loop ends as the rest of the vertices have higher degrees in the array $perm$; this verification is performed in lines 22 to 24. Vertex $i$ is first matched to itself in line 25, in case no other option is available; in other words, vertex $i$ is provisionally matched to vertex $maxidx$ (itself). The vertices are then visited, using the for loop in lines 26 to 33, in order to select a vertex $k$ with high degree to match with vertex $i$. The vertices with the highest degrees are located in the last positions of array $perm$. Lines 28 and 29 verify that vertex $k$ is $UNMATCHED$ and not an island; there is no restriction of the combined weight of vertices $i$ and $k$. Once this is confirmed, vertex $i$ is matched to vertex $k$ in line 30 and the for loop is terminated in line 31. The final and definitive matching of vertex $i$ is done after the loop; the number of coarse vertices is updated in line 34, vertex $i$ is matched to the vertex in $maxidx$, and vice versa, in lines 35 and 36.

The for loop of lines 39 to 57 visits the vertices of the graph according to the sorted permutation in $perm$, it starts from the first non island vertex. If vertex $i$ is $UNMATCHED$, in line 41, then an adjacent vertex is going to be chosen to match with it. Vertex $i$ is first matched to itself in lines 42 and 43, in case

no other option is available; this is, vertex $i$ is provisionally matched to vertex $maxidx$ (itself) and the weight of the edge between them $maxwgt$ is 0 (as no edge links the vertex to itself). The adjacent vertices of vertex $i$ are then visited, using the for loop in lines 44 to 52, in order to select the adjacent vertex $k$ with the heaviest edge to match with vertex $i$. In lines 46 to 48 a decision is made, if adjacent vertex $k$ is $UNMATCHED$ and the edge $(i, k)$ is heavier than the current weight in $maxwgt$ and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed specified by $maxvwgt$ then vertex $i$ is matched to vertex $k$ in lines 49 and 50. The final and definitive matching of vertex $i$ is done after all its adjacent vertices have been examined in the loop; the number of coarse vertices is updated in line 53, vertex $i$ is matched to the vertex in $maxidx$, and vice versa, in lines 54 and 55. Finally, line 58 returns the array with the matching information and the number of coarse vertices in the resulting coarse graph.

Improvements were made when SHEM was rewritten in version 5 of METIS; the entire partitioning library was redesigned. The process to match islands was optimized; multi-constraint support and 2-hop matching were added to improve SHEM as well. Multi-constraint means that a vertex has multiple weights and 2-hop matching is used to match a vertex with a non-adjacent one. Not much of the original code was preserved, however, the core idea behind SHEM remains. The light gray text in Table A.3 is the shared code between the original and the enhanced version of SHEM, the black text represents the new or modified code present in the enhanced version of SHEM. Lines 9 to 17 calculate the 2-hop keys for the vertices. Lines 36 to 45 find suitable matches for the islands. The rest of the code is divided into two main sections, one for a single constraint and the other for multiple constraints; the code of these two sections is almost identical. The next paragraphs describe, in detail, the enhanced version of the SHEM algorithm.

**Table A.3: SHEM detailed algorithm (enhanced version)**

| **Algorithm** | Sorted Heavy Edge Matching (enhanced version) |
|---|---|
| **Input:** | Maximum weight allowed for a vertex $maxvwgt$ |
| | Structure with information of the graph $graph$ |
| **Output:** | Array with maximal matching of the graph $match$ |
| | Number of new coarse vertices $cnvtxs$ |

```
1:  procedure MATCH_SHEM(maxvwgt, graph)
2:    nvtxs ← graph.nvtxs
3:    ncon ← graph.ncon
4:    xadj ← graph.xadj
5:    vwgt ← graph.vwgt
6:    adjncy ← graph.adjncy
7:    adjwgt ← graph.adjwgt
8:    match ← INITIALIZE(nvtxs, UNMATCHED)
9:    vkeys ← INITIALIZE(nvtxs, 0)
10:   for i ← 0 to nvtxs − 1 step 1 do
11:     if (xadj[i + 1] − xadj[i]) > MAXDEGREEFOR2HOP then
12:       continue
```

```
13:       end if
14:       for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
15:         vkeys[i] ← vkeys[i] + adjncy[j]
16:       end for
17:     end for
18:     tperm ← RANDOM_PERMUTE(nvtxs)
19:     avgdegree ← 0.7 ∗ (xadj[nvtxs]/nvtxs)
20:     for i ← 0 to nvtxs − 1 step 1 do
21:       if (xadj[i + 1] − xadj[i]) > avgdegree then
22:         degrees[i] ← avgdegree
23:       else
24:         degrees[i] ← xadj[i + 1] − xadj[i]
25:       end if
26:     end for
27:     perm ← BUCKET_SORT_KEYS_INC(nvtxs, degrees, tperm)
28:     cnvtxs ← 0
29:     last_unmatched ← 0
30:     for pi ← 0 to nvtxs − 1 step 1 do
31:       i ← perm[pi]
32:       if match[i] = UNMATCHED then
33:         maxidx ← i
34:         maxwgt ← −1
35:         if ALL_LESS_THAN(ncon, vwgt[i ∗ ncon], maxvwgt) then
36:           if xadj[i] = xadj[i + 1] then
37:             last_unmatched ← 1 + MAX(pi, last_unmatched)
38:             for last_unmatched ← last_unmatched to nvtxs − 1 step 1 do
39:               j ← perm[last_unmatched]
40:               if match[j] = UNMATCHED then
41:                 maxidx ← j
42:                 break
43:               end if
44:             end for
45:           else /*Current vertex is not island*/
46:             if ncon = 1 then
47:               for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
48:                 k ← adjncy[j]
49:                 if match[k] = UNMATCHED and
50:                 maxwgt < adjwgt[j] and
51:                 (vwgt[i] + vwgt[k]) ≤ maxvwgt[0] then
52:                   maxidx ← k
53:                   maxwgt ← adjwgt[j]
```

```
54:             end if
55:           end for /*Goes through all adjacent vertices*/
56:           if maxidx = i and
57:           (xadj[i + 1] − xadj[i]) ≤ MAXDEGREEFOR2HOP then
58:             for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
59:               ii ← adjncy[j]
60:               if (xadj[ii + 1] − xadj[ii]) < 50 then
61:                 jjinc ← 1
62:               else
63:                 jjinc ← 1 + RANDOM((xadj[ii + 1] − xadj[ii])/10)
64:               end if
65:               for jj ← xadj[ii] to xadj[ii + 1] − 1 step jjinc do
66:                 k ← adjncy[jj]
67:                 if k ≠ i and
68:                 match[k] = UNMATCHED and
69:                 vkeys[i] = vkeys[k] and
70:                 (xadj[i + 1] − xadj[i]) = (xadj[k + 1] − xadj[k]) and
71:                 (vwgt[i] + vwgt[k]) ≤ maxvwgt[0] then
72:                   maxidx ← k
73:                   break
74:                 end if
75:               end for
76:               if maxidx ≠ i then
77:                 break
78:               end if
79:             end for /*Goes through all adjacent vertices*/
80:           end if /*2-hop matching*/
81:         else /*Multiple constraints*/
82:           for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
83:             k ← adjncy[j]
84:             if match[k] = UNMATCHED and
85:             (maxwgt < adjwgt[j] or
86:             (maxwgt = adjwgt[j] and
87:             BETTER_BALANCE(ncon, vwgt[i ∗ ncon],
88:             vwgt[maxidx ∗ ncon], vwgt[k ∗ ncon])) and
89:             ALL_SUMS_LESS_OR_EQUAL_TO(ncon,
90:             vwgt[i ∗ ncon], vwgt[k ∗ ncon], maxvwgt) then
91:               maxidx ← k
92:               maxwgt ← adjwgt[j]
93:             end if
94:           end for /*Goes through all adjacent vertices*/
```

```
95:             if maxidx = i and
96:             (xadj[i + 1] − xadj[i]) ≤ MAXDEGREEFOR2HOP then
97:               for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
98:                 ii ← adjncy[j]
99:                 if (xadj[ii + 1] − xadj[ii]) < 50 then
100:                  jjinc ← 1
101:                else
102:                  jjinc ← 1 + RANDOM((xadj[ii + 1] − xadj[ii])/10)
103:                end if
104:                for jj ← xadj[ii] to xadj[ii + 1] − 1 step jjinc do
105:                  k ← adjncy[jj]
106:                  if k ≠ i and
107:                  match[k] = UNMATCHED and
108:                  vkeys[i] = vkeys[k] and
109:                  (xadj[i + 1] − xadj[i]) = (xadj[k + 1] − xadj[k]) and
110:                  ALL_SUMS_LESS_OR_EQUAL_TO(ncon,
111:                  vwgt[i ∗ ncon], vwgt[k ∗ ncon], maxvwgt) then
112:                    maxidx ← k
113:                    break
114:                  end if
115:                end for
116:                if maxidx ≠ i then
117:                  break
118:                end if
119:              end for /*Goes through all adjacent vertices*/
120:            end if /*2-hop matching*/
121:          end if /*One constraint*/
122:        end if /*Current vertex is island*/
123:      end if /*Current vertex weight not big*/
124:      cnvtxs ← cnvtxs + 1
125:      match[i] ← maxidx
126:      match[maxidx] ← i
127:    end if /*Current vertex not matched*/
128:  end for /*Goes through all vertices*/
129:  return match, cnvtxs
130: end procedure
```

Two main arguments are received by the enhanced version of SHEM; namely $maxvwgt$ and $graph$. $maxvwgt$ specifies the maximum weight allowed for a coarse vertex with single or multi-constraint; two vertices will only be matched if their combined weight does not exceed $maxvwgt$. However, the islands are partially excluded from this restriction. $graph$ is a structure which contains all information related to the graph, such as the number of vertices and edges, their weights and adjacency information,

among others. The algorithm returns the array $match$ with the matching information and $cnvtxs$ with the number of coarse vertices.

The first part of the algorithm is for initialization purposes. Lines 2 to 7 initialize local variables with information of the graph; $nvtxs$ contains the number of vertices, $ncon$ contains the number of constraints, $xadj$ and $adjncy$ the adjacency information, $vwgt$ the weights of the vertices and $adjwgt$ the weights of the edges. Line 8 initializes the array $match$ with $UNMATCHED$; all vertices are unmatched at this point of the execution. Line 9 initializes the array $vkeys$ used during the 2-hop matching. The for loop, in lines 10 to 17, computes the 2-hop keys for each vertex if their degree does not exceed the value established by $MAXDEGREEFOR2HOP$, i.e., if the vertices meet the requirements for 2-hop matching. Then the temporal array $tperm$ is also initialized, in line 18; it contains a random permutation used to sort the vertices later in line 27. The constrained average degree in $avgdegree$ is limited to 70% of the actual value; all vertices with a degree over $avgdegree$ are treated equally. The for loop, in lines 20 to 26, computes the degrees for all vertices limiting their values to a maximum of $avgdegree$. Then, in line 27, the array $perm$ is populated with a sorted permutation of the vertices according to their degree previously computed. $cnvtxs$ is initialized in line 28; it will contain the number of coarse vertices after the matching process. Finally, the last initialization, in line 29, is that of $last\_unmatched$ that will keep track of the last unmatched non-island vertex during the matching process of island vertices.

The for loop of lines 30 to 128 visits the vertices of the graph according to the sorted permutation in $perm$. If vertex $i$ is $UNMATCHED$, in line 32, then it is first matched to itself in lines 33 and 34, in case no other option is available; this is, vertex $i$ is provisionally matched to vertex $maxidx$ (itself) and the weight of the edge between them $maxwgt$ is set to $-1$ (as no edge links the vertex to itself). If the weight of vertex $i$ does not exceed the maximum allowed for a coarse vertex, in line 35, then the process to find a suitable vertex to match with it continues; if vertex $i$ is *too big*, it matches to itself to prevent oversized coarse vertices. This verification handles vertices with one or multiple constraints. The lines 36 to 45 match the islands if they exist, i.e., vertices without edges; the search for a match starts from the last unmatched vertex, defined line 37. The for loop finds the first available vertex $j$, lines 38 to 44; vertex $i$ is matched to vertex $j$ if the latter is $UNMATCHED$ and the search ends, lines 40 to 43. There is no restriction of the combined weight of vertices $i$ and $j$. The code from lines 46 to 121 is divided into two main sections: lines 47 to 80 for single constraint and 82 to 120 for multiple constraints. The two sections follow the same logic and share most of the code; the single constraint is described next.

The adjacent vertices of vertex $i$ are visited, using the for loop in lines 47 to 55, in order to select the adjacent vertex $k$ with the heaviest edge to match with vertex $i$. In lines 49 to 51 a decision is made, if adjacent vertex $k$ is $UNMATCHED$ and the edge $(i,k)$ is heavier than the current weight in $maxwgt$ and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed specified by $maxvwgt$ then vertex $i$ is matched to vertex $k$ in lines 52 and 53. If no match was found with the normal process, then 2-hop matching is employed when vertex $i$ meets the requirements; the verification is done in lines 56 and 57, vertex $i$ is matched to itself and its degree does not exceed $MAXDEGREEFOR2HOP$. The adjacent vertices of vertex $i$ are visited again, using the for loop in lines 58 to 79, in order to select a vertex $k$ suited to match with vertex $i$. Vertex $k$ is adjacent to vertex $ii$, which in turn is adjacent to vertex $i$; this is the idea behind 2-hop matching. Lines 60 to 64 define how

many vertices $k$ are visited according to the degree of vertex $ii$. The adjacent vertices of vertex $ii$ are then visited, using the for loop in lines 65 to 75, in order to select a vertex $k$ to match with vertex $i$. If vertex $k$ is not vertex $i$, and vertex $k$ is $UNMATCHED$, and both vertices have the same 2-hop key, and they have the same degree, and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed specified by $maxvwgt$ then vertex $i$ is matched to vertex $k$ and the search ends in lines 67 to 74. Finally, in lines 76 to 78, if vertex $i$ is not matched to itself anymore the 2-hop matching process is terminated.

The multi-constraint section is similar to that of a single constraint described earlier. The difference lies in the two conditions of lines 84 to 90 and 106 to 111. The first condition matches vertex $k$ to vertex $i$ if adjacent vertex $k$ is $UNMATCHED$, and the edge $(i,k)$ is heavier than the current weight in $maxwgt$, or the edge has the same weight but it leads to a better balance, and the combined weight of vertices $i$ and $k$ does not exceed the maximum allowed specified by $maxvwgt$ for all constraints. The second condition is practically similar; it just considers the multiple constraints of vertices.

The final and definitive matching of vertex $i$ is done after all its adjacent vertices have been analyzed and the 2-hop matching process completed; the number of coarse vertices is updated in line 124, vertex $i$ is matched to the vertex in $maxidx$, and vice versa, in lines 125 and 126. Finally, line 129 returns the array with the matching information and the number of coarse vertices in the resulting coarse graph.

## A.3. DSHEM Detailed Algorithm

DSHEM is based on the enhanced version of SHEM; they share the majority of the code. The key difference is the condition used to decide when vertex $k$ is matched to vertex $i$. Much of the new code in DHSEM gathers information, which is then used in the condition mentioned above. DSHEM takes advantage of the data structures, utilized by METIS to store the graph information, in order to simulate the bidirectional communication that would rise during the FEM simulations. It introduces the concept of *sources* and, in combination with the new directed edges, generates a better approximation of the real communication between the different parts.

The detailed version of DSHEM depicted in Table A.4 differentiates the new or modified code, in black, from the inherited SHEM code, in light gray. We refer the reader to the SHEM Detailed Algorithm in the previous section of the appendix for the description of this share code.

The for loop in lines 14 to 22 computes the number of positive and negative edges incident to every vertex in the graph. This information is later used to calculate the number of sources produced if vertex $i$ is matched to vertex $k$; with fewer sources the total amount of communication is also reduced. The for loop in lines 64 to 68 finds the edge $(k,i)$, i.e., edge $(i,k)$ in the opposite direction. These edges have different values in DSHEM as opposed to SHEM where their values are always equal. Lines 69 to 99 compute the number of sources for the case of vertex $k$ matching vertex $i$. The conditional in lines 101 to 112 decides whether vertex $k$ is matched to vertex $i$ taking into consideration the new gathered information. The same process is also implemented for the multi-constraint section of the algorithm.

Three main arguments are received by DSHEM; namely $maxvwgt$, $graph$ and $percentages$. $maxvwgt$ specifies the maximum weight allowed for a coarse vertex with single or multi-constraint; two vertices will only be matched if their combined weight does not exceed $maxvwgt$, but the islands are partially excluded from this restriction. $graph$ is a structure which contains all information related to

the graph, such as the number of vertices and edges, their weights and adjacency information, among others. $percentages$ contains three percentage values employed to fine tune the algorithm. The algorithm returns the array $match$ with the matching information and $cnvtxs$ with the number of coarse vertices.

**Table A.4: DSHEM detailed algorithm**

| **Algorithm** | Directed Sorted Heavy Edge Matching |
|---|---|
| **Input:** | Maximum weight allowed for a vertex $maxvwgt$ |
| | Structure with information of the graph $graph$ |
| | Structure with matching percentages for DSHEM $percentages$ |
| **Output:** | Array with maximal matching of the graph $match$ |
| | Number of new coarse vertices $cnvtxs$ |

```
1:  procedure MATCH_DSHEM(maxvwgt, graph, percentages)
2:    nvtxs ← graph.nvtxs
3:    ncon ← graph.ncon
4:    xadj ← graph.xadj
5:    vwgt ← graph.vwgt
6:    adjncy ← graph.adjncy
7:    adjwgt ← graph.adjwgt
8:    p1 ← percentages.r_dshem_p1/100
9:    p2 ← percentages.r_dshem_p2/100
10:   p3 ← percentages.r_dshem_p3/100
11:   match ← INITIALIZE(nvtxs, UNMATCHED)
12:   vtxadjp ← INITIALIZE(nvtxs, 0)
13:   vtxadjn ← INITIALIZE(nvtxs, 0)
14:   for i ← 0 to nvtxs − 1 step 1 do
15:     for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
16:       if adjncy[i] ≥ 0 then
17:         vtxadjp[i] ← vtxadjp[i] + 1
18:       else
19:         vtxadjn[i] ← vtxadjn[i] + 1
20:       end if
21:     end for
22:   end for
23:   vkeys ← INITIALIZE(nvtxs, 0)
24:   for i ← 0 to nvtxs − 1 step 1 do
25:     if (xadj[i + 1] − xadj[i]) > MAXDEGREEFOR2HOP then
26:       continue
27:     end if
28:     for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
29:       vkeys[i] ← vkeys[i] + adjncy[j]
```

```
30:     end for
31:   end for
32:   tperm ← RANDOM_PERMUTE(nvtxs)
33:   avgdegree ← 0.7 * (xadj[nvtxs]/nvtxs)
34:   for i ← 0 to nvtxs − 1 step 1 do
35:     if (xadj[i + 1] − xadj[i]) > avgdegree then
36:       degrees[i] ← avgdegree
37:     else
38:       degrees[i] ← xadj[i + 1] − xadj[i]
39:     end if
40:   end for
41:   perm ← BUCKET_SORT_KEYS_INC(nvtxs, degrees, tperm)
42:   cnvtxs ← 0
43:   last_unmatched ← 0
44:   for pi ← 0 to nvtxs − 1 step 1 do
45:     i ← perm[pi]
46:     if match[i] = UNMATCHED then
47:       maxidx ← i
48:       maxwgt ← −1
49:       minsources ← 5
50:       if ALL_LESS_THAN(ncon, vwgt[i * ncon], maxvwgt) then
51:         if xadj[i] = xadj[i + 1] then
52:           last_unmatched ← 1 + MAX(pi, last_unmatched)
53:           for last_unmatched ← last_unmatched to nvtxs − 1 step 1 do
54:             j ← perm[last_unmatched]
55:             if match[j] = UNMATCHED then
56:               maxidx ← j
57:               break
58:             end if
59:           end for
60:         else /*Current vertex is not island*/
61:           if ncon = 1 then
62:             for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
63:               k ← adjncy[j]
64:               for iadj ← xadj[k] to xadj[k + 1] − 1 step 1 do
65:                 if adjncy[iadj] = i then
66:                   break
67:                 end if
68:               end for
69:               nsources ← 0
70:               if adjwgt[j] ≥ 0 then
```

```
71:             if vtxadjp[i] > 1 then
72:               nsources ← nsources + 1
73:             end if
74:             if vtxadjn[i] > 0 then
75:               nsources ← nsources + 1
76:             end if
77:           else
78:             if vtxadjn[i] > 1 then
79:               nsources ← nsources + 1
80:             end if
81:             if vtxadjp[i] > 0 then
82:               nsources ← nsources + 1
83:             end if
84:           end if
85:           if adjwgt[iadj] ≥ 0 then
86:             if vtxadjp[k] > 1 then
87:               nsources ← nsources + 1
88:             end if
89:             if vtxadjn[k] > 0 then
90:               nsources ← nsources + 1
91:             end if
92:           else
93:             if vtxadjn[k] > 1 then
94:               nsources ← nsources + 1
95:             end if
96:             if vtxadjp[k] > 0 then
97:               nsources ← nsources + 1
98:             end if
99:           end if
100:          bidiradjwgt ← ABS(adjwgt[j]) + ABS(adjwgt[iadj])
101:          if match[k] = UNMATCHED and
102:          ((nsources < minsources and
103:          (maxwgt * p1) < bidiradjwgt) or
104:          (nsources = minsources and
105:          (maxwgt * p2) < bidiradjwgt) or
106:          (nsources > minsources and
107:          (maxwgt * p3) < bidiradjwgt)) and
108:          (vwgt[i] + vwgt[k]) ≤ maxvwgt[0] then
109:            maxidx ← k
110:            maxwgt ← bidiradjwgt
111:            minsources ← nsources
```

```
112:              end if
113:            end for /*Goes through all adjacent vertices*/
114:            if maxidx = i and
115:            (xadj[i + 1] − xadj[i]) ≤ MAXDEGREEFOR2HOP then
116:              for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
117:                ii ← adjncy[j]
118:                if (xadj[ii + 1] − xadj[ii]) < 50 then
119:                  jjinc ← 1
120:                else
121:                  jjinc ← 1 + RANDOM((xadj[ii + 1] − xadj[ii])/10)
122:                end if
123:                for jj ← xadj[ii] to xadj[ii + 1] − 1 step jjinc do
124:                  k ← adjncy[jj]
125:                  if k ≠ i and
126:                  match[k] = UNMATCHED and
127:                  vkeys[i] = vkeys[k] and
128:                  (xadj[i + 1] − xadj[i]) = (xadj[k + 1] − xadj[k]) and
129:                  (vwgt[i] + vwgt[k]) ≤ maxvwgt[0] then
130:                    maxidx ← k
131:                    break
132:                  end if
133:                end for
134:                if maxidx ≠ i then
135:                  break
136:                end if
137:              end for /*Goes through all adjacent vertices*/
138:            end if /*2-hop matching*/
139:          else /*Multiple constraints*/
140:            for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
141:              k ← adjncy[j]
142:              for iadj ← xadj[k] to xadj[k + 1] − 1 step 1 do
143:                if adjncy[iadj] = i then
144:                  break
145:                end if
146:              end for
147:              nsources ← 0
148:              if adjwgt[j] ≥ 0 then
149:                if vtxadjp[i] > 1 then
150:                  nsources ← nsources + 1
151:                end if
152:                if vtxadjn[i] > 0 then
```

```
153:              nsources ← nsources + 1
154:            end if
155:          else
156:            if vtxadjn[i] > 1 then
157:              nsources ← nsources + 1
158:            end if
159:            if vtxadjp[i] > 0 then
160:              nsources ← nsources + 1
161:            end if
162:          end if
163:          if adjwgt[iadj] ≥ 0 then
164:            if vtxadjp[k] > 1 then
165:              nsources ← nsources + 1
166:            end if
167:            if vtxadjn[k] > 0 then
168:              nsources ← nsources + 1
169:            end if
170:          else
171:            if vtxadjn[k] > 1 then
172:              nsources ← nsources + 1
173:            end if
174:            if vtxadjp[k] > 0 then
175:              nsources ← nsources + 1
176:            end if
177:          end if
178:          bidiradjwgt ← ABS(adjwgt[j]) + ABS(adjwgt[iadj])
179:          if match[k] = UNMATCHED and
180:          (((nsources < minsources and
181:          (maxwgt * p1) < bidiradjwgt) or
182:          (nsources = minsources and
183:          (maxwgt * p2) < bidiradjwgt) or
184:          (nsources > minsources and
185:          (maxwgt * p3) < bidiradjwgt)) or
186:          ((nsources = minsources and
187:          maxwgt = bidiradjwgt) and
188:          BETTER_BALANCE(ncon, vwgt[i * ncon],
189:          vwgt[maxidx * ncon], vwgt[k * ncon]))) and
190:          ALL_SUMS_LESS_OR_EQUAL_TO(ncon,
191:          vwgt[i * ncon], vwgt[k * ncon], maxvwgt) then
192:            maxidx ← k
193:            maxwgt ← bidiradjwgt
```

```
194:              minsources ← nsources
195:            end if
196:          end for /*Goes through all adjacent vertices*/
197:          if maxidx = i and
198:          (xadj[i + 1] − xadj[i]) ≤ MAXDEGREEFOR2HOP then
199:            for j ← xadj[i] to xadj[i + 1] − 1 step 1 do
200:              ii ← adjncy[j]
201:              if (xadj[ii + 1] − xadj[ii]) < 50 then
202:                jjinc ← 1
203:              else
204:                jjinc ← 1 + RANDOM((xadj[ii + 1] − xadj[ii])/10)
205:              end if
206:              for jj ← xadj[ii] to xadj[ii + 1] − 1 step jjinc do
207:                k ← adjncy[jj]
208:                if k ≠ i and
209:                match[k] = UNMATCHED and
210:                vkeys[i] = vkeys[k] and
211:                (xadj[i + 1] − xadj[i]) = (xadj[k + 1] − xadj[k]) and
212:                ALL_SUMS_LESS_OR_EQUAL_TO(ncon,
213:                vwgt[i ∗ ncon], vwgt[k ∗ ncon], maxvwgt) then
214:                  maxidx ← k
215:                  break
216:                end if
217:              end for
218:              if maxidx ≠ i then
219:                break
220:              end if
221:            end for /*Goes through all adjacent vertices*/
222:          end if /*2-hop matching*/
223:        end if /*One constraint*/
224:      end if /*Current vertex is island*/
225:    end if /*Current vertex weight not big*/
226:    cnvtxs ← cnvtxs + 1
227:    match[i] ← maxidx
228:    match[maxidx] ← i
229:  end if /*Current vertex not matched*/
230: end for /*Goes through all vertices*/
231: return match, cnvtxs
232: end procedure
```

The first part of the algorithm is for initialization purposes. Lines 8 to 10 initializes the three percentages $p1$, $p2$ and $p3$. These percentages are used to fine tune how much the weight of the edge,

combined with the number of sources, affects the decision to match vertex $k$ to vertex $i$. The lines 12 and 13 initialize the two arrays used to store the number of positive and negative edges that each vertex has. The for loop of lines 14 to 22 visits all the vertices of the graph, while the for loop of lines 15 to 21 visits the adjacent vertices of the current vertex $i$ of the outer loop. The amount of positive and negative edges of current vertex $i$ is then computed in lines 16 to 20. When vertex $i$ is matched with some vertex $k$, the number of sources produced by that match has to be kept to a minimum.

Once in the matching process, the variable $minsources$ is initialized every time a new vertex $i$ is going to be treated; this is done in line 49. $minsources$ is set to 5 due to the nature of the matching process; a maximum of 4 sources could result from any matching of two vertices. Further in the code, the same two sections from SHEM are present: the first one for the single constraint case, lines 62 to 138, and the second one for the multi-constraint case, lines 140 to 222. These two sections share most of the code and only the single constraint is described here; the multi-constraint section uses a similar approach. It is important to note that DSHEM keeps the same code for islands and 2-hop matching.

The edge $(i, k)$ is already available to DSHEM when vertex $i$ is treated and its adjacent vertex $k$ is visited; however, the edge $(k, i)$ is yet to be found. DSHEM uses directed edges to better approximate the communication volume, the reason why both edges may have different values. SHEM uses undirected edges, both always with the same value, without the need to find that second edge. The for loop of lines 64 to 68 visits the adjacent vertices of vertex $k$ and once the location of vertex $i$ is found, the edge $(k, i)$ is available. With both edges, $(i, k)$ and $(k, i)$, the number of sources can be computed. First, the number of sources in $nsources$ is initialized in line 69. Then the two conditionals, lines 70 to 84 and 85 to 99, calculate the number of sources for the provisional matching of vertices $i$ and vertex $k$. The total communication going through the edge linking both vertices is calculated next; line 100 computes the bidirectional communication. Note that the minus sign (-) in the weights of edges is used to indicate the origin of that communication and not its amount.

With the new data obtained previously, DSHEM can now make a more informed decision whether vertices $i$ and $k$ should be matched together. It takes into consideration the three different scenarios that are possible: the number of sources is reduced, preserved or increased. It depends on the weight of the edge, in combination with the percentages, if increasing the number of sources is preferred than reducing them. It could be more beneficial to remove an over weighted edge, at the cost of increasing the sources, than removing a very light edge. Removing heavy edges in the coarsening process also reduces the overall communication in the coarsest graph. The conditional in lines 101 to 108 is that of SHEM with the addition of the three different scenarios described earlier. Instead of considering only the weight of the edge (i.e., $maxwgt < adjwgt[j]$), the number of sources and percentage are included (e.g., $nsources < minsources$ **and** $(maxwgt * p1) < bidiradjwgt$).

The multi-constraint section is similar to that of a single constraint described earlier. The difference lies in the conditional of lines 179 to 191 which considers the multiple constraints of vertices.

The final and definitive matching of vertex $i$ is done after all its adjacent vertices have been analyzed and the 2-hop matching process completed; the number of coarse vertices is updated in line 226, vertex $i$ is matched to the vertex in $maxidx$, and vice versa, in lines 227 and 228. Finally, line 231 returns the array with the matching information and the number of coarse vertices in the resulting coarse graph.

# Appendix B.

# Graphs

It is important to understand the geometry of the set of graphs used for the experimentation and how it affects the performance of the algorithms under investigation. For that purpose, a more detailed description of the real life and synthetic graphs is provided in this appendix. The creation of the synthetic graphs is also explained here.

## B.1. Real Life Graphs

The collection of real life graphs represents the most common types of geometries found in FE meshes. Their size is also considered to measure the performance degradation, if any, of the algorithm under test. Small and medium size graphs have been used for the experimentation phase. Coordinate information is only used to plot the graphs and not for the purpose of the partitioning process. This information helps understand the geometry of the graph by visual inspection.

### B.1.1. 2D Graphs

One 2D triangular graph has been chosen. It is a small graph with 15 thousand vertices and 45 thousand edges.

#### ef_4elt

This small 2D graph is based on a triangular mesh around an airfoil with front slat and rear flaps, see Figure B.1.

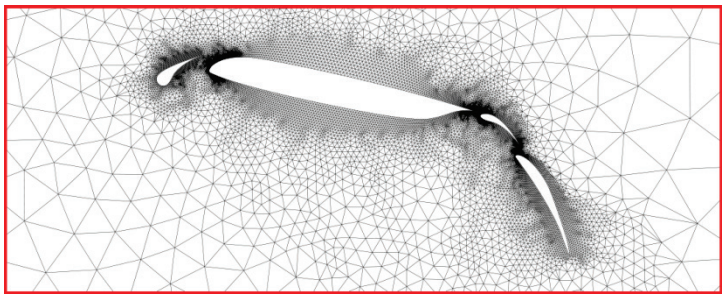

**Figure B.1. Graph of an airfoil with flaps.**

The darker areas around the airfoil, slat and flaps represent the concentration of elements of the mesh, as shown in detail in Figure B.2.

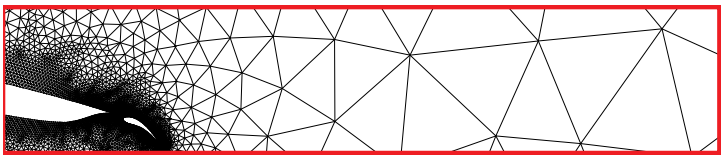

**Figure B.2. Detailed view of the graph of an airfoil with flaps.**

The mesh has an irregular distribution of the elements as depicted in Figure B.3; the wing is located on the center of the graph. It can be seen that the mesh elements are limited to a close area around the airfoil.

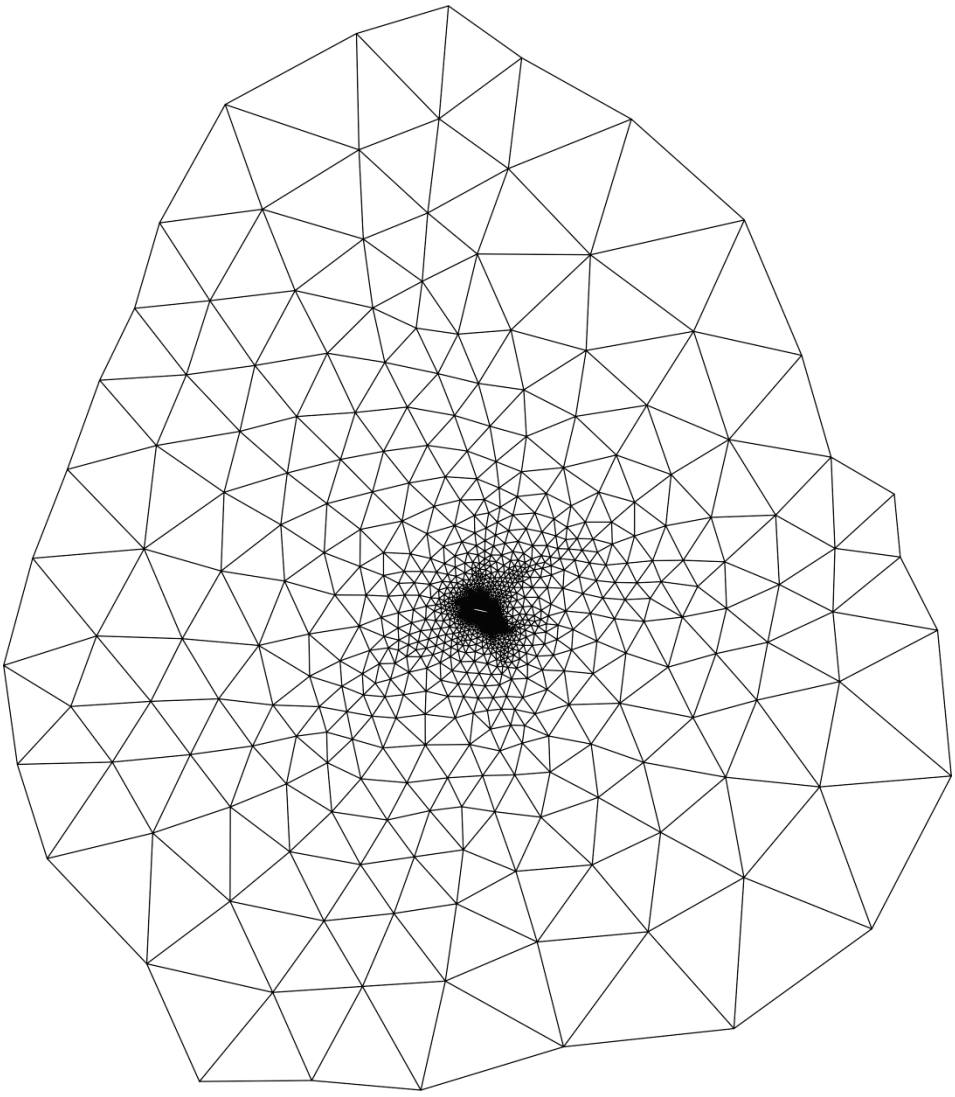

**Figure B.3. Full graph of an airfoil with flaps.**

When the graph is plotted without coordinate information, the elements of the mesh are distributed uniformly making more evident the concentration of elements around the wing; see Figure B.4.

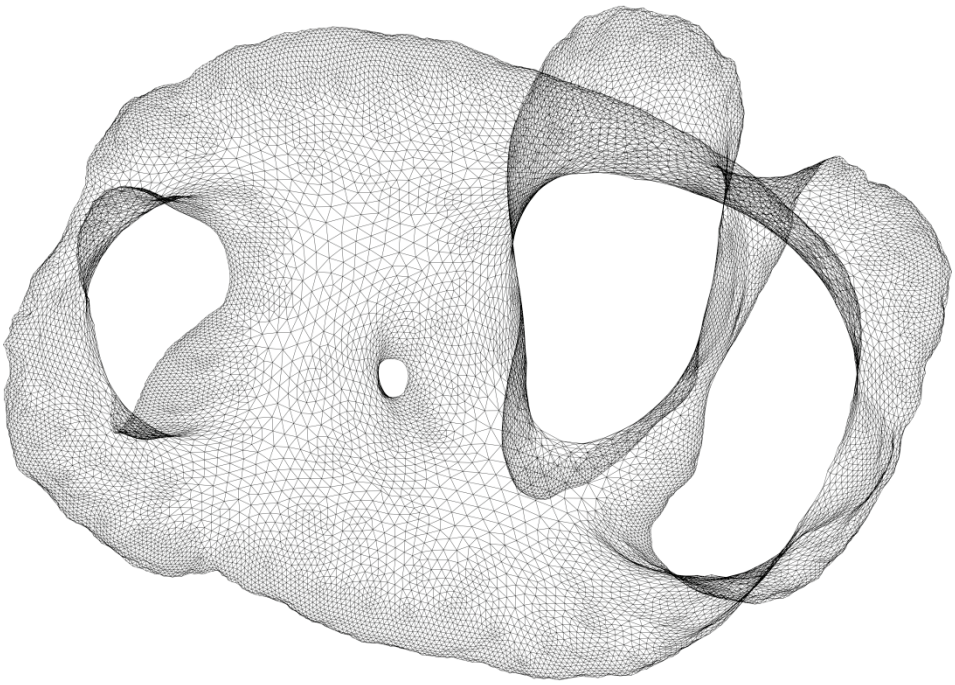

**Figure B.4. Full graph of an airfoil with flaps without coordinate information.**

A large percentage of the edges are located around the wing and flaps, close to 90% of them.

### B.1.2. 3D Graphs

Two 3D graphs have been used for the experiments; one medium size quadrangular graph with 140 thousand vertices and 400 thousand edges, and a small triangular graph with 16 thousand vertices and 49 thousand edges.

#### ef_ocean

This 3D graph represents the oceans of the world, the empty areas belong to the continents; Figure B.5 shows a small section of the entire plot.

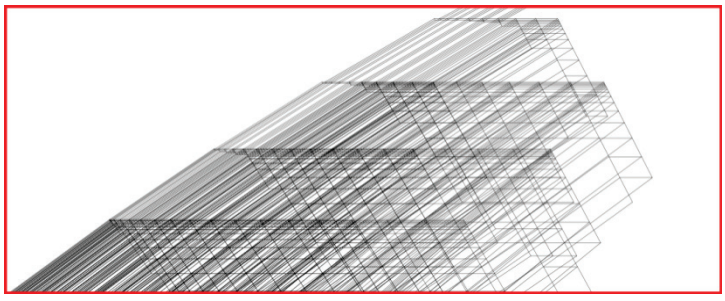

**Figure B.5. Graph of the ocean.**

A detailed view of the graph is depicted in Figure B.6.

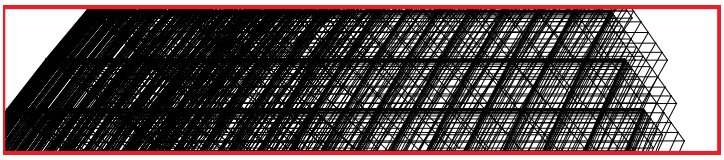

**Figure B.6. Detailed view of the graph of the ocean.**

This graph has all its elements distributed in a regular manner, see Figure B.7; the darker areas are due to the overlay of the different sections of the graph in the 2D image. It can be seen that the size of the mesh elements vary according to their proximity to the poles.

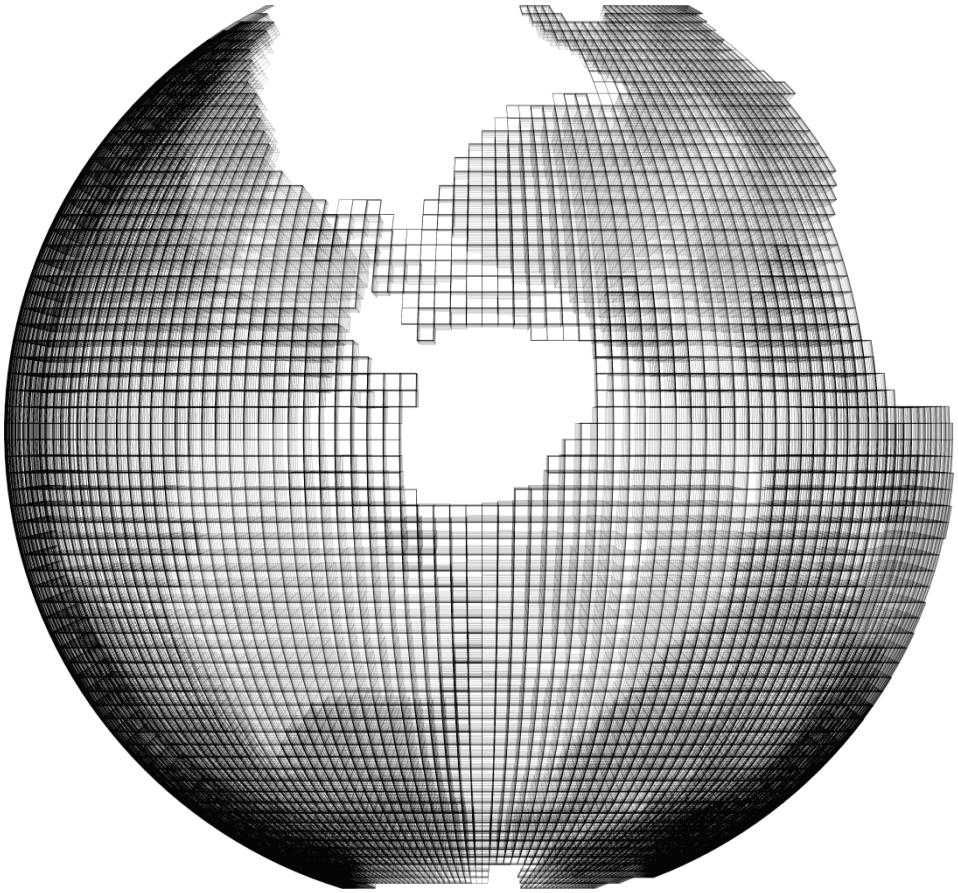

**Figure B.7. Full graph of the ocean.**

When the graph is plotted without coordinate information, the irregularity shown in Figure B.8 comes from the size of the mesh elements.

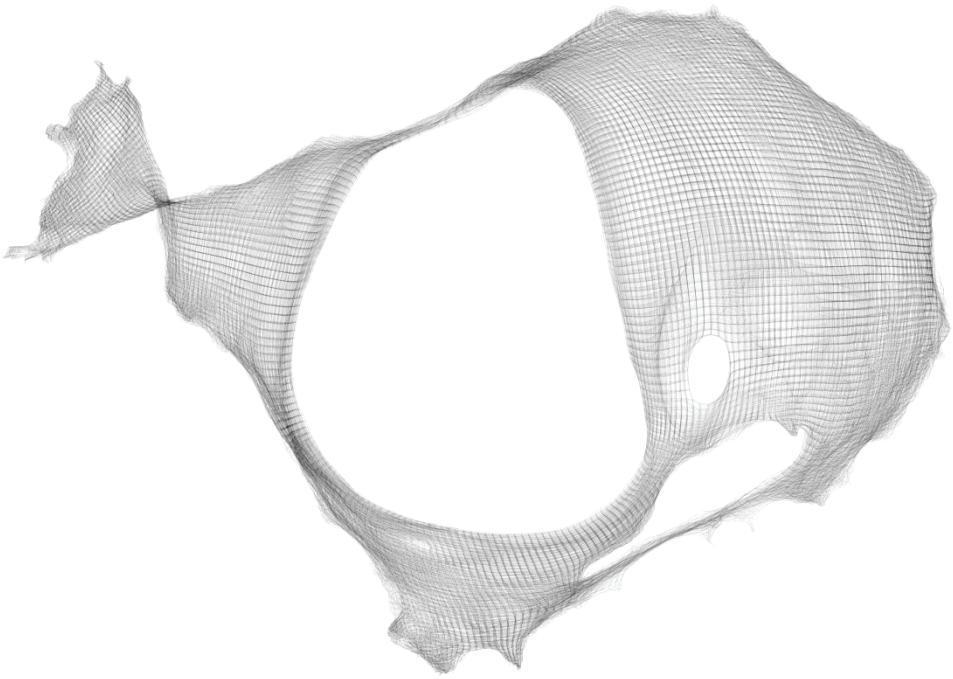

**Figure B.8. Full graph of the ocean without coordinate information.**

### ef_sphere

This 3D graph represents a sphere with one layer of mesh elements. Figure B.9 shows a small section of the entire plot.

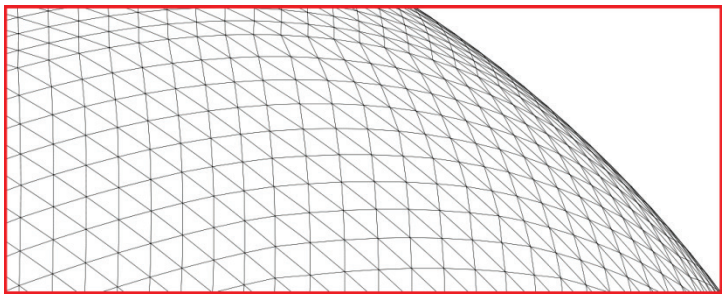

**Figure B.9. Graph of a sphere.**

A detailed view of the graph is depicted in Figure B.10.

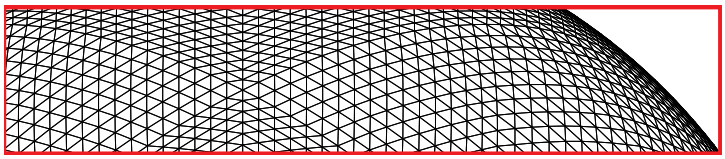

**Figure B.10. Detailed view of the graph of a sphere.**

This graph has all its elements distributed in a regular manner, see Figure B.11; the darker areas are due to the overlay of the different sections of the graph in the 2D image.

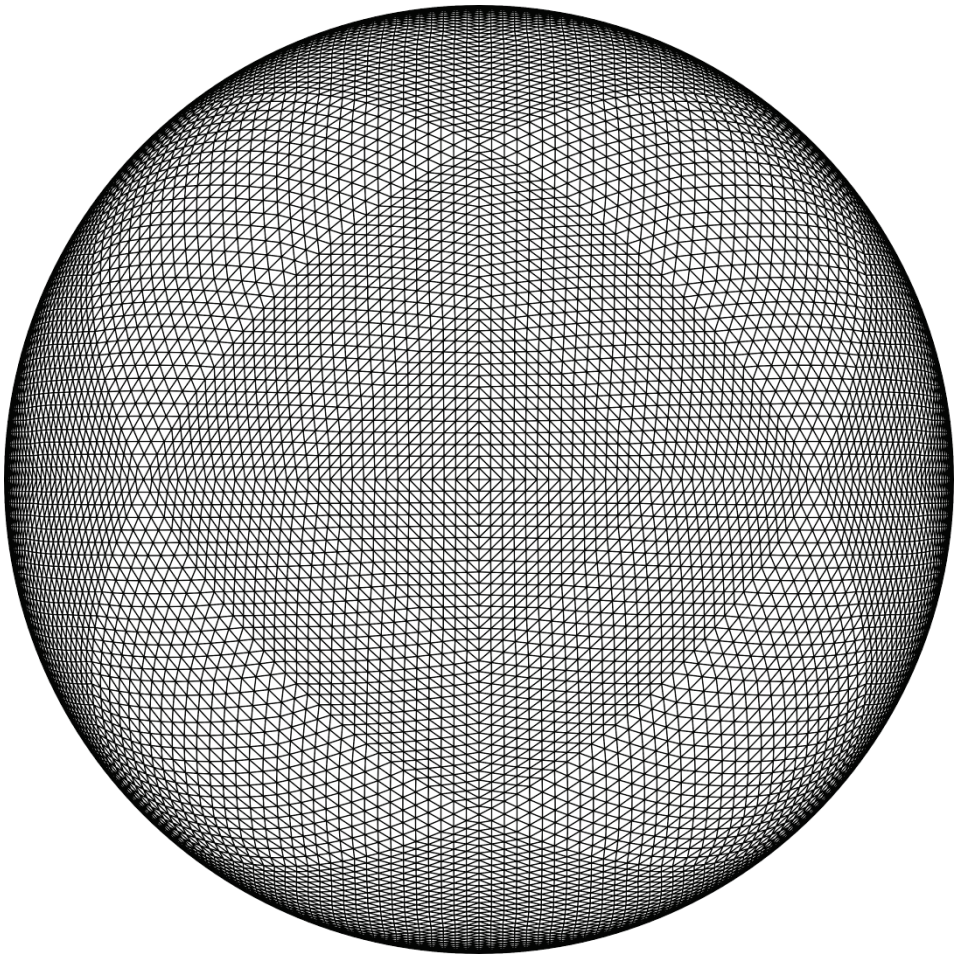

**Figure B.11. Full graph of a sphere.**

When the graph is plotted without coordinate information, it is evident that not much irregularity is present; this indicates the regular pattern of the mesh elements as showing in Figure B.12.

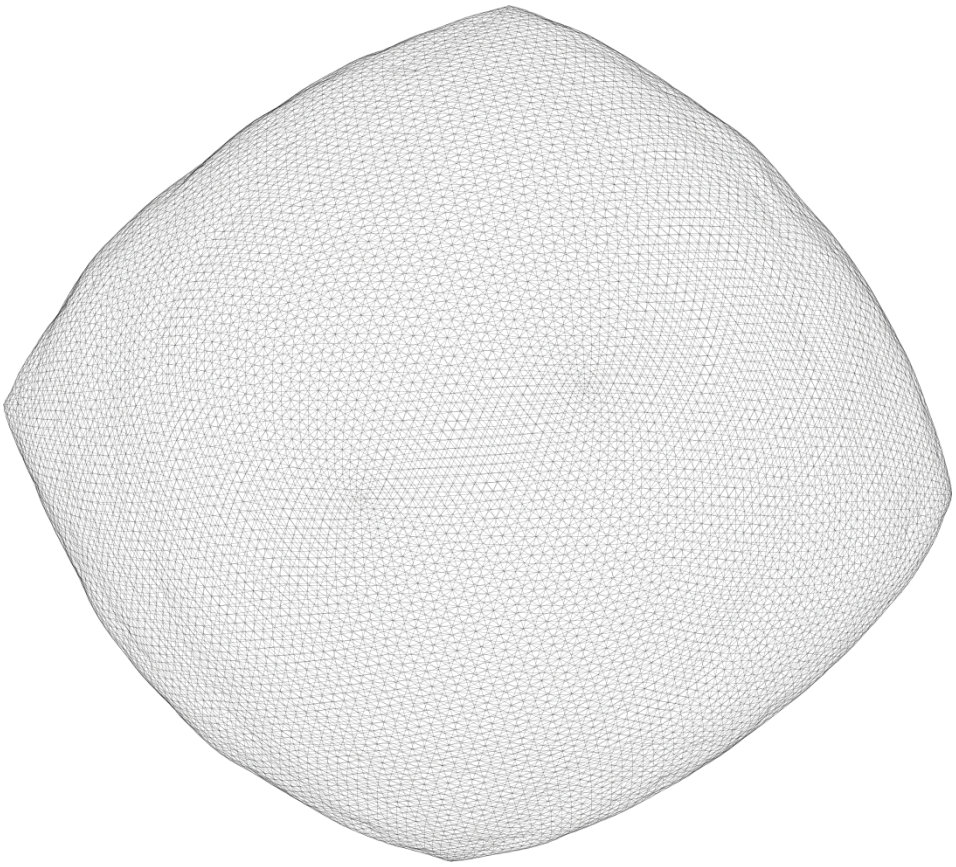

**Figure B.12. Full graph of a sphere without coordinate information.**

## B.2. Synthetic Graphs

A set of synthetic graphs was created to study the behavior of the algorithm with greater detail. They include the different geometries present in the real life graphs, but in a regular, and predictive, pattern. In addition, some random irregularity is introduced in order to analyze the performance of the partitioning process. The next paragraphs describe the different synthetic graphs and their structure.

### B.2.1. Regular

Regular graphs are designed to establish how the geometry affects the partition. For this purpose, 2D graphs, and their respective 3D version, are part of the set. Three different geometries are used: one quadrangular and two triangular geometries.

#### Square

This is a graph with a quadrangular geometry. Its size is measured by the number of vertices per side; being always a regular square of $n \times n$ vertices. Figure B.13 depicts a square graph with 5 vertices per side ($5 \times 5$).

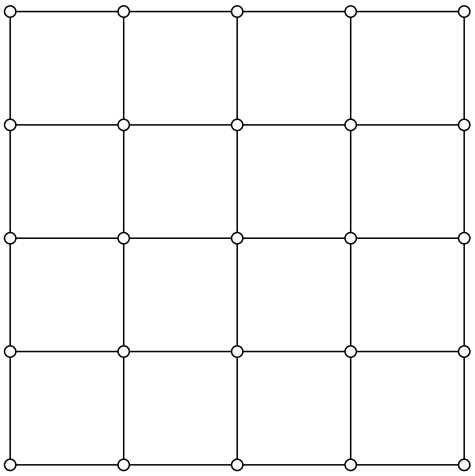

**Figure B.13. Square graph with 5 vertices per side.**

The 3D version of the square graph is generated by adding a third dimension with $n$ layers (copies) of the original square graph and connecting them together. The result of this process is a graph with a size of $n \times n \times n$ vertices.

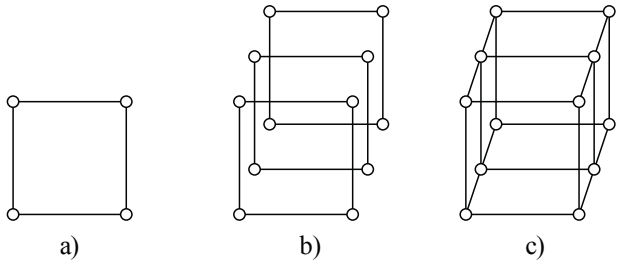


**Figure B.14. Creation of a 3D square graph with 2 vertices per side and 3 levels.**

Figure B.14 shows the process used to create a 3D version of a square graph. It is a simplification of the process to better understand how the graph is created. In this case, the resulting graph has a size of $2 \times 2 \times 3$, and not $2 \times 2 \times 2$ as previously mentioned. The process starts with the original graph as shown in a); subsequent copies of the graph are generated in b), in this case 3 copies in total. Finally, all layers are connected with edges, creating the cubes, by linking all vertices.

## Triangular-Square

This is a graph with a triangular geometry. Its size is measured by the number of vertices per side; being always a regular square of $n \times n$ vertices. Figure B.15 depicts a triangular square graph with 5 vertices per side ($5 \times 5$).

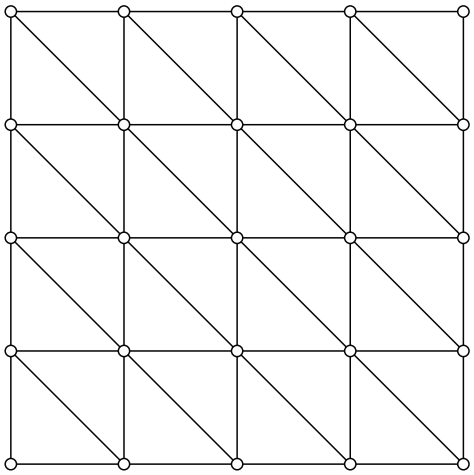

**Figure B.15. Triangular square graph with 5 vertices per side.**

The 3D version of the triangular square graph is generated by adding a third dimension with $n$ layers (copies) of the original triangular square graph and connecting them together. The result of this process is a graph with a size of $n \times n \times n$ vertices.

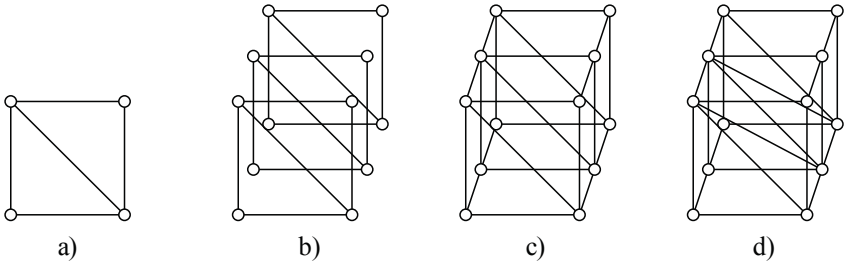


**Figure B.16. Creation of a 3D triangular square graph with 2 vertices per side and 3 levels.**

Figure B.16 shows the process used to create a 3D version of a triangular square graph. It is a simplification of the process to better understand how the graph is created. In this case, the resulting graph has a size of $2 \times 2 \times 3$, and not $2 \times 2 \times 2$ as previously mentioned. The process starts with the original graph as shown in a); subsequent copies of the graph are generated in b), in this case 3 copies in total. Next, all layers are connected with edges by linking all vertices; similar to the 3D square graph. Finally, a new edge is added to every cube linking the vertices having already 4 incident edges.

### Dense Triangular-Square

This is a graph with a triangular geometry. Its size is measured by the number of vertices per side; being always a regular square of $n \times n$ vertices. However, the total number of vertices is not given by its size, as described next. Figure B.17 depicts a dense triangular square graph with 5 vertices per side ($5 \times 5$). Besides the regular square graph it is based on, $(n-1) \times (n-1)$ vertices are added and linked to the original $n \times n$ vertices. This creates a dense graph with triangular geometry.

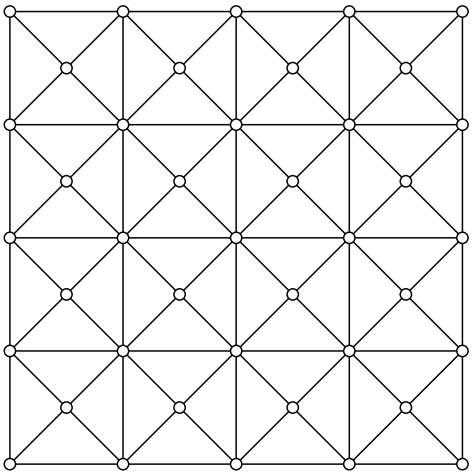

**Figure B.17. Dense triangular square graph with 5 vertices per side.**

The 3D version of the dense triangular square graph is generated by adding a third dimension with $n$ layers (copies) of the original dense triangular square graph and connecting them together. The result of this process is a graph with a size of $n \times n \times n$.

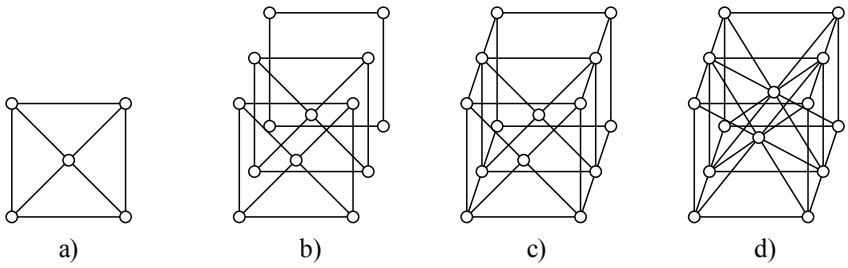


**Figure B.18. Creation of a 3D dense triangular square graph with 2 vertices per side and 3 levels.**

Figure B.18 shows the process used to create a 3D version of a dense triangular square graph. It is a simplification of the process to better understand how the graph is created. In this case, the resulting graph has a size of $2 \times 2 \times 3$, and not $2 \times 2 \times 2$ as previously mentioned. The process starts with the original graph as shown in a); subsequent copies of the graph are generated in b), in this case 3 copies in total. Note that the last copy is a square graph, not a dense triangular graph. Next, all layers are connected with edges by linking the vertices; similar to the 3D square graph. Finally, the vertex in the center of every square is linked to the vertices of the square in the next level.

## B.2.2. Irregular

The regular synthetic graphs are used as base to build the irregular graphs. Edges are randomly removed from the regular graph, with uniform probability, to create an irregular version of it. Different probabilities are used to see the effect of the irregularity in the partition results.

### Square

This is a graph with an irregular quadrangular geometry. Its size is measured by the number of vertices per side; being always a square of $n \times n$ vertices. Figure B.19 depicts an irregular square graph with 5 vertices per side ($5 \times 5$); all edges are visited and removed with a probability of 25% for this example.

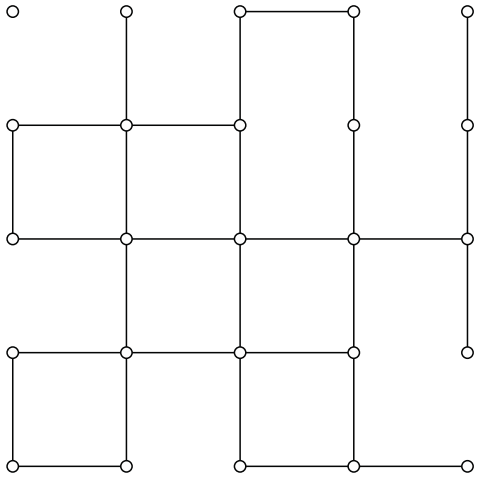

**Figure B.19. Square graph with 5 vertices per side. Edges were removed with a probability of 25%.**

The 3D version of the irregular square graph follows the same process. It starts from a regular 3D square graph, and then all edges are visited and removed with a given probability. The resulting irregular 3D graph is used for the evaluation of the partitioning algorithm.

### Triangular-Square

This is a graph with an irregular triangular geometry. Its size is measured by the number of vertices per side; being always a square of $n \times n$ vertices. Figure B.20 depicts an irregular triangular square graph with 5 vertices per side ($5 \times 5$); all edges are visited and removed with a probability of 25% for this example.

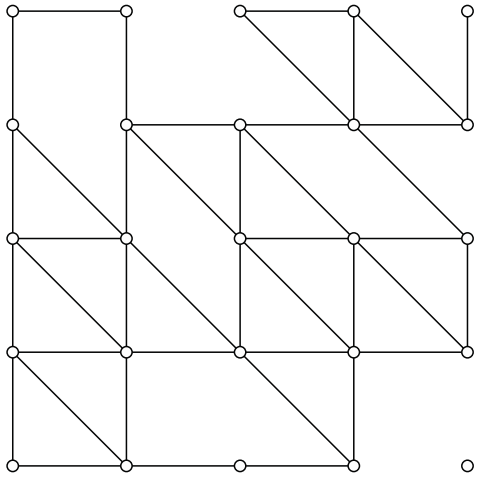

**Figure B.20. Triangular square graph with 5 vertices per side. Edges were removed with a probability of 25%.**

The 3D version of the irregular triangular square graph follows the same process. It starts from a regular 3D triangular square graph, and then all edges are visited and removed with a given probability. The resulting irregular 3D graph is used for the evaluation of the partitioning algorithm.

### Dense Triangular-Square

This is a graph with an irregular triangular geometry. Its size is measured by the number of vertices per side; being always a square of $n \times n$ vertices. However, the total number of vertices is not given by its

size, similar to the regular version. Figure B.21 depicts an irregular dense triangular square graph with 5 vertices per side (5 × 5); all edges are visited and removed with a probability of 25% for this example.

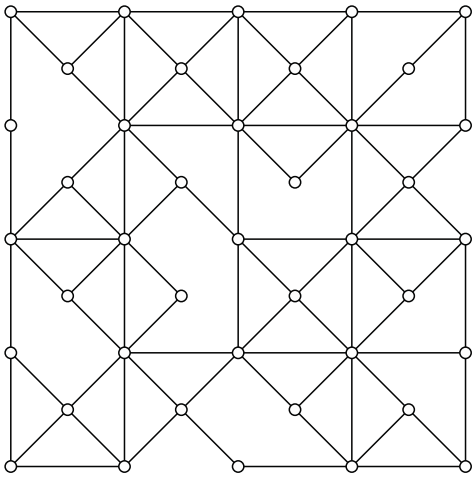

**Figure B.21. Dense triangular square graph with 5 vertices per side. Edges were removed with a probability of 25%.**

The 3D version of the irregular dense triangular square graph follows the same process. It starts from a regular 3D dense triangular square graph, and then all edges are visited and removed with a given probability. The resulting irregular 3D graph is used for the evaluation of the partitioning algorithm.