1-1-2009

A genetic algorithm based partition generation and optimization method for finite element problems

Lubomir Jirasek
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>

Part of the [Aerospace Engineering Commons](#)

Recommended Citation

Jirasek, Lubomir, "A genetic algorithm based partition generation and optimization method for finite element problems" (2009).
Theses and dissertations. Paper 607.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

A GENETIC ALGORITHM BASED PARTITION GENERATION AND OPTIMIZATION
METHOD FOR FINITE ELEMENT PROBLEMS

by

Lubomir Jirasek,
B. Eng., Ryerson University, 2007

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in the Program of
Aerospace Engineering

Toronto, Ontario, Canada, 2009

© Lubomir Jirasek 2009

Author's Declaration Page

I hereby declare that I am the sole author of this Research Paper.

I authorize Ryerson University to lend this Research Paper to other institutions or individuals for the purpose of scholarly research.

Signature

Date

I further authorize Ryerson University to reproduce this Research Paper by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

Date

Abstract

A Genetic Algorithm Based Partition Generation and Optimization Method for Finite Element Problems

Master of Applied Science in Aerospace Engineering, 2009

Lubomir Jirasek

Department of Aerospace Engineering

Ryerson University

A two-step partitioning algorithm for FE meshes is proposed in this work for the purposes of time savings. A direct method based on the concept of ‘separateness’ was applied first, followed by a partition optimization process that used a Genetic Algorithm (GA). A total of 9 applications were evaluated to demonstrate the durability, versatility, and effectiveness of this partitioning algorithm with respect to interface node count and subdomain load balance. Beyond this, a wingbox optimization problem was performed on a single processor using a GA to demonstrate the possible time savings of this method. With a 30% decrease in compute time witnessed, it can be said with confidence that the proposed partitioning algorithm was a success.

Acknowledgements

The author of this paper would like to thank his supervisor Dr. Kamran Behdinin for all his time and help.

He would also like to extend his deepest gratitude to his family and friends, and especially L.K, for all their love and support.

Table of Contents

AUTHOR’S DECLARATION PAGE III

ABSTRACT V

ACKNOWLEDGEMENTS VII

TABLE OF CONTENTS IX

LIST OF TABLES..... XIII

LIST OF FIGURES..... XV

LIST OF APPENDICES..... XVII

NOMENCLATURE XIX

ACRONYMS XXI

1. INTRODUCTION 1

1.1 BRIEF HISTORY..... 2

1.2 OBJECTIVES 4

1.3 CONTRIBUTION 4

1.4 SCOPE..... 5

2. LITERATURE REVIEW 7

2.1 OPTIMIZATION TECHNIQUES 7

2.1.1 Evolutionary Algorithms..... 8

2.1.1.1 Fundamental Aspects of Evolutionary Algorithms 9

2.1.1.2 Genetic Algorithm (GA) 12

2.1.1.3 Genetic Programming (GP)..... 13

2.1.1.4 Evolutionary Strategies (ES) 15

2.1.1.5 Evolutionary Programming (EP)..... 16

2.1.2 Other Stochastic Heuristics 17

2.1.2.1 Simulated Annealing (SA) 17

2.1.2.2 Particle Swarm Approach (PSA)..... 19

2.1.2.3 Stochastic Evolution (SE) 21

2.1.2.4 Tabu Search (TS) 22

2.1.2.5 Artificial Neural Networks (ANN)..... 23

2.2 DIRECT PARTITION ALGORITHMS 24

2.2.1 Greedy Method 25

2.2.2 Minimum Bandwidth Method 27

2.2.3 Inertial Method..... 28

2.2.4 Recursive Bisection Method 29

2.2.5 Separateness Method..... 29

2.3 ITERATIVE/OPTIMIZATION PARTITIONING ALGORITHMS 30

2.3.1 Finite Element Tearing and Interconnecting (FETI) Methods 31

2.3.2 Recursive Bisection Methods..... 32

2.3.3 Two-Step Methods 33

2.3.4 Other Iterative/Optimization Based Partitioning Methods..... 35

2.4 WINGBOX STRUCTURAL OPTIMIZATION 36

2.4.1 Multidisciplinary Optimization (MDO)..... 37

2.4.2	Purely Structural Wingbox Optimization	38
3.	BACKGROUND INFORMATION	41
3.1	SEPARATENESS DIRECT PARTITIONING.....	41
3.2	PARTITION OPTIMIZATION	44
4.	CURRENT WORK	47
4.1	DIRECT METHOD	47
4.1.1	Convert Finite Element Mesh to Graph.....	47
4.1.2	Determine Subdomain Initialization Elements	48
4.1.3	Sequentially Expand Subdomains.....	49
4.2	OPTIMIZATION METHOD.....	50
4.2.1	Optimization Initialization.....	51
4.2.2	Partition Optimization.....	51
4.2.2.1	Balancing Item	52
4.2.2.2	Balancing Method	52
4.2.2.3	Fitness Penalty	53
4.2.2.4	Termination Criteria.....	53
4.2.3	Update Solution.....	53
4.3	GOVERNING PROCESS.....	54
4.4	IMPLEMENTATION.....	56
5.	WINGBOX OPTIMIZATION PROBLEM SETUP	61
5.1	OPTIMIZATION SETUP	62
5.2	FINITE ELEMENT MODEL	63
5.3	BOUNDARY CONDITIONS	65
6.	VALIDATION/CALIBRATION.....	67
6.1	MATLAB GA TOOLBOX VALIDATION	67
6.1.1	10 Bar Truss	67
6.1.2	25 Bar Truss	69
6.2	GA PARAMETER CALIBRATION	72
6.3	DIRECT PARTITION CALIBRATION	73
6.3.1	Expansion Element Choice.....	74
6.3.2	Graph Choice	74
6.4	PARTITION OPTIMIZATION CALIBRATION	75
6.4.1	Balancing Item.....	75
6.4.2	Balancing Method	76
6.5	GOVERNING PROCESS CALIBRATION.....	77
7.	RESULTS.....	83
7.1	PARTITIONING ALGORITHM – GENERAL APPLICATION	83
7.2	WINGBOX STRUCTURAL OPTIMIZATION	95
8.	DISCUSSION.....	99
8.1	PARTITION ALGORITHM – GENERAL APPLICATION	99
8.2	WINGBOX STRUCTURAL OPTIMIZATION.....	104
9.	CONCLUSION	109
10.	SUGGESTED FUTURE WORK	113

REFERENCES 115

APPENDIX 123

APPENDIX A - MATLAB CODE SAMPLE..... 123

APPENDIX B - ANSYS..... 124

 Appendix B.1 - ANSYS Parametric Design Language (APDL) and Batch Code..... 124

 Appendix B.2 - Substructuring Analysis 126

 Appendix B.3 - Frontal Solvers..... 130

APPENDIX C - WINGBOX LOAD CALCULATIONS 131

APPENDIX D - PSEUDO-CODE FOR RGB METHODS 134

List of Tables

TABLE 1 – MATERIAL PROPERTIES FOR 10 BAR TRUSS.....	68
TABLE 2 – PREVIOUS 10 BAR TRUSS OPTIMIZATION RESULTS.....	68
TABLE 3 – CURRENT 10 BAR TRUSS OPTIMIZATION RESULTS	69
TABLE 4 – DESIGN VARIABLE GROUPING	70
TABLE 5 – LOADS APPLIED TO 25 BAR TRUSS OPTIMIZATION PROBLEM	71
TABLE 6 – PAST 25 BAR TRUSS OPTIMIZATION RESULTS	71
TABLE 7 – CURRENT 25 BAR TRUSS OPTIMIZATION RESULTS	71
TABLE 8 – GA CALIBRATION RESULTS	73
TABLE 9 – BALANCING ITEM CALIBRATION RESULTS.....	76
TABLE 10 – BALANCING METHOD CALIBRATION RESULTS	77
TABLE 11 – MESH 2 PARTITION RESULTS SUMMARY FOR CURRENT WORK	84
TABLE 12 – MESH 2 PARTITION RESULTS SUMMARY FOR PAST WORK	84
TABLE 13 – MESH 3 PARTITION RESULTS SUMMARY.....	85
TABLE 14 – MESH 4 PARTITION RESULTS SUMMARY.....	86
TABLE 15 – MESH 5 PARTITION RESULTS SUMMARY.....	93
TABLE 16 – MESH 6 PARTITION RESULTS SUMMARY FOR CURRENT WORK	93
TABLE 17 – MESH 6 PARTITION RESULTS SUMMARY FOR PAST WORK	93
TABLE 18 – MESH 7 PARTITION RESULTS SUMMARY.....	94
TABLE 19 – MESH 8 PARTITION RESULTS SUMMARY.....	94
TABLE 20 – MESH 9 PARTITION RESULTS SUMMARY.....	94
TABLE 21 – BEST SOLUTION OF GA WINGBOX STRUCTURAL OPTIMIZATION.....	96
TABLE 22 – PRE-RUN TIME ESTIMATIONS	96
TABLE 23 – MEASURED TIME	97
TABLE 24 – TIME VALIDATION	97

List of Figures

FIGURE 1 – SINGLE-POINT CROSSOVER.....	10
FIGURE 2 – MULTI-POINT CROSSOVER.....	11
FIGURE 3 – GA CHROMOSOME.....	12
FIGURE 4 – PROGRAM TREE.....	14
FIGURE 5 – CROSSOVER REPRODUCTION	14
FIGURE 6 – EVOLUTIONARY STRATEGY PROCESS	16
FIGURE 7 – EVOLUTIONARY PROGRAMMING PROCESS.....	17
FIGURE 8 – SIMULATED ANNEALING FLOW CHART	18
FIGURE 9 – STOCHASTIC EVOLUTION PROCESS.....	22
FIGURE 10 – TABU SEARCH PROCESS.....	23
FIGURE 11 – NEURON PROCESSING STRUCTURE	23
FIGURE 12 – NEURAL NETWORK ARCHITECTURE	24
FIGURE 13 – GRAPH TYPES	25
FIGURE 14 – GREEDY ALGORITHM EXPANSION PROCESS	26
FIGURE 15 – MODIFIED GREEDY ALGORITHM EXPANSION PROCESS	27
FIGURE 16 – RECURSIVE CO-ORDINATE BISECTION PROCESS	29
FIGURE 17 – RECURSIVE GRAPH BISECTION PROCESS	32
FIGURE 18 – RECURSIVE SPECTRAL BISECTION PROCESS	33
FIGURE 19 – ELEMENT PERTURBATION.....	34
FIGURE 20 – SIMPLE GRAPH.....	42
FIGURE 21 – ELEMENT CLIQUE REPRESENTATIVE GRAPH GENERATION PROCESS.....	43
FIGURE 22 – SUBDOMAIN INITIALIZATION ELEMENTS	43
FIGURE 23 – INITIALIZATION ELEMENTS.....	48
FIGURE 24 – SUBDOMAIN EXPANSION.....	50
FIGURE 25 – PARTITION OPTIMIZATION SEARCH SPACE	51
FIGURE 26 – GOVERNING PROCESS FLOW CHART.....	55
FIGURE 27 – FLOW CHART OF CODE	57
FIGURE 28 – CUTAWAY VIEW OF P180 WING STRUCTURE	61
FIGURE 29 – WINGBOX DIMENSIONS	63
FIGURE 30 – WINGBOX MESH VALIDATION.....	64
FIGURE 31 – FE MODEL OF WINGBOX WITH SKIN PANELS MISSING.....	64
FIGURE 32 – FE MODEL CONSTRAINTS.....	65
FIGURE 33 – ELLIPTICAL LIFT DISTRIBUTION	66
FIGURE 34 – COMPLETE WINGBOX BOUNDARY CONDITIONS	66
FIGURE 35 – 10 BAR TRUSS PROBLEM	68
FIGURE 36 – 25 BAR TRUSS PROBLEM	70
FIGURE 37 – GA CALIBRATION MODEL	72
FIGURE 38 – EXPANSION RESULTS FOR DIAGONAL DUAL AND DUAL GRAPHS	74
FIGURE 39 – BALANCING ITEM CALIBRATION PARTITIONS.....	76
FIGURE 40 – BALANCING METHOD CALIBRATION PARTITIONS.....	77
FIGURE 41 – CALIBRATION WINGBOX DIMENSIONS.....	78
FIGURE 42 – TRANSPARENT VIEW OF CALIBRATION WINGBOX	78
FIGURE 43 – BOUNDARY CONDITIONS APPLIED TO CALIBRATION WINGBOX	79
FIGURE 44 – FE EVALUATION TIME CALIBRATION DATA.....	80
FIGURE 45 – DIRECT PARTITION EVALUATION TIME CALIBRATION DATA.....	80
FIGURE 46 – FE EVALUATION TIME TREND DATA.....	81

FIGURE 47 – DIRECT PARTITION EVALUATION TIME TREND DATA	82
FIGURE 48 – MESH 1 PARTITION RESULTS	83
FIGURE 49 – MESH 2 PARTITION RESULTS	84
FIGURE 50 – MESH 3 PARTITION RESULTS	85
FIGURE 51 – MESH 4 PARTITION RESULTS	86
FIGURE 52 – MESH 5 PARTITION RESULTS	88
FIGURE 53 – MESH 6 PARTITION RESULTS	89
FIGURE 54 – MESH 7 PARTITION RESULTS	90
FIGURE 55 – MESH 8 PARTITION RESULTS	91
FIGURE 56 – MESH 9 PARTITION RESULTS	92
FIGURE 57 – GA FITNESS PLOT	95
FIGURE 58 – DIRECT PARTITION RESULTS FOR WINGBOX PROBLEM	97
FIGURE 59 – PARTITION OPTIMIZATION RESULTS FOR WINGBOX PROBLEM	97
FIGURE 60 – WINGBOX STRESS CONSTRAINT VALIDATION	98
FIGURE 61 – WINGBOX DISPLACEMENT CONSTRAINT VALIDATION	98
FIGURE 62 – ANSYS GUI.....	125
FIGURE 63 – ANSYS LOG FILE.....	126
FIGURE 64 – ANSYS SUBSTRUCTURING ANALYSIS FLOW CHART	127
FIGURE 65 – SIMPLE FE EXAMPLE	127
FIGURE 66 – GEN PASS	128
FIGURE 67 – USE PASS	129
FIGURE 68 – EXP PASS	129
FIGURE 69 – FRONTAL SOLVER FLOW CHART	131
FIGURE 70 – LOADS APPLIED TO WINGBOX PROBLEM	133
FIGURE 71 – ENGINE LOADS APPLIED TO WINGBOX	134

List of Appendices

APPENDIX A – MATLAB CODE SAMPLE.....123

APPENDIX B – ANSYS.....124

 Appendix B.1 – ANSYS Parametric Design Language (APDL) and Batch Mode.....124

 Appendix B.2 – Substructuring Analysis.....126

 Appendix B.3 – Frontal Solvers.....130

APPENDIX C – WINGBOX LOAD CALCULATION.....131

APPENDIX D – PSEUDO-CODE FOR RGB METHODS.....134

Nomenclature

f_i	Fitness of i^{th} individual
f_k	Fitness of k^{th} individual
Ψ	GA Population Size
ε	Tournament Size
y_n	Decimal number
k	Counting variable
h	Number of binary numbers
B	Binary number (0 or 1)
x_n	Decimal number
x^u	Upper limit decimal number
x^l	Lower limit decimal number
dx	Desired decimal resolution
$F(x)$	Corrected fitness function
$F_{\max}$	Largest expected fitness value
$f(x)$	Fitness function
R	Penalty Variable
m	Number of constraints
J	Constraint
$\phi(x)$	Penalty function
$g_J(x)$	Constraint function
$f'(x)$	Transformed fitness function
p	Simulated annealing probability
δf	Change in fitness function
T	Simulated annealing temperature value
$x_{i,d}$	Swarm particle position
$v_{i,d}$	Swarm particle velocity
ω	Inertial factor
c_1	Cognitive coefficient
r_{1d}	Random value
c_2	Social coefficient
r_{2d}	Random value
$pbest_{i,d}$	Particle's best value
$lbest_{i,d}$	Group's best value
t	Time
u	Direction vector
I	Inertia
x_i	x-coordinate of i^{th} item
x_{cg}	x-coordinate of center of gravity
y_i	y-coordinate of i^{th} item
y_{cg}	y-coordinate of center of gravity
z_i	z-coordinate of i^{th} item
z_{cg}	z-coordinate of center of gravity
N_n	Number of items
λ_2	Second Eigenvalue

x_2	Fiedler vector
S	Separateness value
r_i	Subdomain expansion element (graph node)
n_i, n_o, v, w	Graph nodes
n_p	Graph node of expansion element
q	Number of subdomains
G'	Dual graph
G''	Diagonal dual graph
V'	Dual graph vertices (nodes)
V''	Diagonal dual graph vertices (nodes)
E'	Dual graph edges (members)
E''	Diagonal dual graph edges (members)
N	Total number of elements or degrees of freedom
j	Next subdomain in generation process
α	Fitness function penalty parameter
N_k	Number of items in k^{th} subdomain
$N_{\text{best}, k}$	Ideal number of items in k^{th} subdomain
$W_{\text{RMS}, k}$	RMS wavefront of k^{th} subdomain
t_s	Standard (non-partitioned) FE evaluation time
t_d	Direct partition time
t_p	Partitioned FE evaluation time
$t_{s, \text{tot}}$	Total standard (non-partitioned) FE evaluation time
$t_{p, \text{tot}}$	Total partitioned FE evaluation time
n	Load Factor
σ_{max}	Maximum von Mises stress in model
$\sigma_{\text{max}, \text{allow}}$	Maximum allowable von Mises stress in model
d_{max}	Maximum vector sum displacement in model
$d_{\text{max}, \text{allow}}$	Maximum allowable vector sum displacement in model
UX	Displacement in x-direction
UY	Displacement in y-direction
UZ	Displacement in z-direction
RX	Rotation about x-axis
RY	Rotation about y-axis
RZ	Rotation about z-axis
$F_t(e)$	Time estimation for governing process as a function of element count
e	Element count
L	Lift force
L'	Lift distribution
W	Aircraft weight
ρ_{∞}	Free stream density
V_{∞}	Free stream velocity
Γ_o	Circulation at wing root
b	Wingspan
y_w	Span co-ordinate
α_i	Induced angle of attack
D_i'	Drag distribution

Acronyms

AC	Ant Colony
ACGA	Ant Colony-Genetic Algorithm
ANN	Artificial Neural Networks
APDL	ANSYS Parametric Design Language
ASA	Adaptive Simulated Annealing
CFD	Computational Fluid Dynamics
EA	Evolutionary Algorithm
EP	Evolutionary Programming
ES	Evolutionary Strategies
FAR	Federal Aviation Regulations
FE	Finite Element
FEM	Finite Element Method
FETI	Finite Element Tearing and Interconnecting
GA	Genetic Algorithm
GP	Genetic Programming
LBF	Load Balance Factor
MDO	Multi-Disciplinary Optimization
PCG	Preconditioned Conjugate Gradient
RAM	Random Access Memory
SA	Simulated Annealing
SE	Stochastic Evolution
SRT	Spanning Root Tree
SSR	Stochastic Sampling with Replacement
RCB	Recursive Co-ordinate Bisection
RGB	Recursive Graph Bisection
RMH	Random Matching Heuristic
RSB	Recursive Spectral Bisection
TS	Tabu Search
VFSA	Very Fast Simulated Annealing

1. Introduction

The use of Finite Element (FE) Methods in the aerospace industry has most certainly increased over the last few decades. With more powerful processors available, today's engineers can easily analyze problems that were not practical only a decade ago. Even with these significant advances in compute power, there is still a need for higher capability. As is the case with FE software, optimization heuristics have also gained popularity in recent years. Applications such as topological wing rib optimization [1] and multi-disciplinary optimization [2] have most definitely become more common. In these types of structural problems, FE models can easily have around 250,000 degrees of freedom [2]. Evaluating one such problem may be reasonable as far as computational effort is concerned, but when applied to an optimization case, compute time can drastically increase. In these situations, models may have to be re-evaluated many times, especially if non-traditional search methods are applied. Often, this is the case since traditional gradient based optimization methods have a tendency of getting stuck in a sub-optimal solution. Thus, if any sort of time savings can be achieved in FE calculations, it is of great benefit.

To overcome this issue, partitioning algorithms have been proposed from many different individuals. The aim here is to partition a FE mesh such that it can be run in parallel on several processors or in series on one processor. If more than one processor is available, a parallel architecture can significantly decrease the compute load since the work can be distributed amongst them. Even in the case of a single processor, running partition calculations in series can give time savings, though not as great as a parallel case. That said, for a partitioning algorithm to be effective, it should have several key characteristics [3]:

- a) Be general in use so that it can be easily applied to various applications
- b) Generate balanced subdomains so that compute load is evenly distributed amongst available processors in a parallel environment and total evaluation time is kept to a minimum in a series application

- c) Reduce the number of interface nodes in order to minimize synchronization costs in a parallel setup and minimize the time required for the 'USE' step in a series evaluation
- d) Produce subdomains with proper aspect ratios as convergence issues may arise in applications containing iterative solvers

By incorporating all of the above items, one can be quite confident that the partitioning algorithm can be applied to a wide variety of problems and still be effective in carrying out its task.

1.1 Brief History

There are many ways to classify the numerous partitioning methods available today. One way is to put them under Graph based or Engineering based methods. Graph based methods refer to algorithms that rely on graph theory to partition a FE mesh. Essentially, 'elements' and 'nodes' are converted to 'vertices' and 'members' to allow for easier manipulation of the mesh. On the other hand, Engineering based methods are algorithms that do not require graph theory to function since they are algebraic [3].

Conversely, another way of categorizing partitioning algorithms is to put them under direct or iterative/optimization headings, which will be the case here. Direct methods refer to partitioning codes that are not very computationally intensive, but do not always achieve high quality partitions. One of the first methods to fall into this category was that of the Greedy Method, first proposed by Farhat [5] in the late 1980's and named after its expansion process which looks like 'bites'. Subsequently, Al-Nasra et al. [6] further developed this algorithm several years later. Another method that falls under the direct umbrella is that of bandwidth based partitioning. These types of algorithms were introduced in the latter part of the 1980's by Malone [7] and are purely algebraic. Inertial based methods discussed by Farhat and Lesoinne [8] in the early 90's on the other hand come in many forms and rely on inertial aspects of FE meshes to generate proper cut directions. Once these directions are defined, the FE mesh is essentially torn into the desired number of subdomains. Moving on, recursive bisection methods are also capable of generating partitions fairly quickly. These were discussed by Simon [9] in

1991 and essentially split the mesh recursively based on various criteria. The last direct methods to mention here are ones based on the idea of ‘separateness’ introduced in the latter part of the 1990’s. These methods are based on studies by Kaveh and Roosta [10, 11] and work by defining initialization elements from which the subdomains are expanded. The initialization elements are chosen on a basis of ‘separateness’ in order to generate the best quality partition.

In general, iterative/optimization based partitioning methods are more computationally intensive but generate higher quality partitions. One of the first widely used algorithms in this category was that of the Finite Element Tearing and Interconnecting (FETI) method, first introduced by Farhat [12] in 1990. This method relies on Lagrange multipliers to carry out domain decomposition in a two step, iterative process. Many different variants of this method have been developed since then. Another type of common partitioning algorithm that fits under the optimization group is that of recursive bisection based methods. Some of the first published work in this field was carried out by Pothen et al. [13] on a graph based process that uses a Kernighan-Lin optimization algorithm and eigenvectors to find an optimal bisection path. More recent work by Kaveh et al. in the last decade has focused on similar partition methods, but using neural [14] and genetic [15] optimization heuristics instead. The last major sub-group under the iterative/optimization category to briefly discuss here is that of two-step methods. Vanderstraeten et al. [4] were the first to propose such a process during the mid 1990’s. Essentially, an initial partition was carried out using a direct partition algorithm, after which it was improved upon by an optimization process that perturbed elements between adjacent subdomains. The work used three stochastic heuristics, which included simulated annealing, tabu search, as well as stochastic evolution. Similar work followed soon after by Bouhmala et al. [16] that used simulated annealing and Kernighan-Lin as the optimization heuristic. Several other algorithms exist that fit under the iterative/optimization heading and will be discussed in the Literature Review (Section 2).

1.2 Objectives

Having now attained some feeling for what is out there as far as FE partitioning algorithms are concerned, this paper will focus on a modified two-step method based on work done by Vanderstraeten et al. [4]. Instead of using a recursive bisection method for the direct portion though, a ‘separateness’ based algorithm [3] will be applied here. Furthermore, a genetic algorithm will be applied to the optimization phase of the code in place of simulated annealing, tabu search, and stochastic evolution used originally. The intent here is to investigate whether a partitioning algorithm can be made such that it can be durable and applied generally to a wide variety of applications while still achieving quality partitions.

Beyond the above objective, another goal will be to apply the partitioning algorithm to a more specific problem. An assumption will be made that the code will be applied exclusively to wingbox structural optimization problems using a GA on a single processor. Thus, a governing process to define when to use the partitioning process and with how many partitions will be made based on this main assumption. Also, it will be assumed that a frontal FE solver will be used to evaluate the possible solutions during the optimization process. As will be shown later, this last assumption will greatly affect the method in which the subdomains will be balanced during the partition optimization process. All code was written using Matlab R2007a, with the genetic algorithm being a modified version of the GA Toolbox. Finite Element aspects of this work on the other hand were handled by incorporating ANSYS V11 and calling it up remotely as required.

1.3 Contribution

This work will make three main contributions to the Finite Element community,

- A modified ‘separateness’ based direct partition algorithm will be proposed that functions off of an element centroid graph. The algorithm will use alternative means to choose subdomain initialization elements and expand the subdomains compared to Kaveh et al. [3].

- A modified partition optimization method will be proposed based on work carried out initially by Vanderstraeten et al. [4]. The algorithm will be iteratively applied by way of genetic algorithm and will feature a slightly modified objective function and subdomain load balancing method.
- A partition evaluation process will be outlined that is run in series on a single processor using the substructuring feature in ANSYS. It will be shown that when applied to GA optimization routines, time savings can be achieved when compared to standard evaluation methods.

1.4 Scope

Given the objectives mentioned above, the scope of this paper will go as follows. Section 2 will give a thorough literature review dealing with non-deterministic optimization methods, direct partitioning methods, iterative/optimization partitioning methods, as well as wingbox optimization problems. The reason for including a summary of non-deterministic heuristics is due to the fact that these are commonly used in partition optimization algorithms and having a basic understanding can be beneficial for the remainder of the paper. Section 3 will give more in-depth background information to past works that share very close ties with the current work. This will include the ‘separateness’ direct partition method [3] and two-step partition optimization method [4]. Section 4 will go on to discuss the current partitioning algorithm. Subsections here will include the Direct Method, Optimization Method, Governing Process, and Implementation. Section 5 will go on to describe the wingbox structural optimization problem undertaken. Aspects related to the optimization setup, FE model, and boundary conditions will be reviewed to give a good understanding. Validation and Calibration will be the subject of Section 6. Aspects related to the Matlab GA Toolbox validation, genetic algorithm parameter calibration, direct partition parameter calibration, optimization parameter calibration, and governing process calibration will be discussed. Results of the general partition algorithm will be summarized in Section 7, along with the findings of the wingbox structural optimization problem. Finally, Sections 8, 9, and 10 will go over the Discussion, Conclusion, and Suggested Future work, respectively.

2. Literature Review

The current section is meant to provide the reader a broad overview of FE partitioning methods as well as wingbox optimization approaches to give some idea as to how the current method fits into the big picture. Since a GA was used in this work, this portion of the paper will first look at giving the reader a basic understanding of all the common stochastic optimization techniques, as they are the most common type in FE partitioning applications. More focus will of course be placed on the GA as it was applied in the current work. Next, a review of the more common direct partitioning methods will then be undertaken to show how the separateness based method compares to its counterparts. This will then be followed by a review of iterative/optimization based partitioning methods to give insight as to how the two-step method compares to alternative methods. Rounding out this section, an overview of past wingbox optimization problems will be conducted to give some perspective as to where the application problem in this work falls in the grand scheme of things.

2.1 Optimization Techniques

Generally speaking, there are two main types of optimization techniques available, that of traditional methods and stochastic heuristics. Traditional methods [18] in the field of optimization have encountered difficulties in the past as the search space for multivariable problems can have several local optimums which may lead to problems. Many of these traditional methods are based on mathematical gradient procedures that function by calculating the first or higher order derivatives. This may not always be possible as objective functions in optimization problems are frequently complex differential equations. Furthermore, these methods have a hard time determining the actual global optimum since local optimum regions can be present in the search space. Stochastic heuristics provide an alternative method to overcome these issues. Optimization methods under this category rely on random events to some extent in order to close in on a global optimum [18]. Due to their non-deterministic nature, discrete problems can be more easily evaluated here. It is because of this robust nature that many of these methods are applied to FE partitioning problems [4, 14, 15]. The remainder of Section 2.1 will focus on Evolutionary Algorithms (EAs) as well as other stochastic heuristics so

that the reader has a basic understanding when the iterative/optimization partitioning methods are reviewed in Section 2.3. Since the GA was used in the current work though, this optimization heuristic will be focused on the most.

2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a group of methods that function on the principles of Darwinian evolution. That said, these processes evolve over a span of time based on the simple notion of survival of the fittest. In the end, the fittest individual in the group is regarded as the optimal solution. Since individuals are represented by strings or vectors, search spaces that are complex, multi-modal, non-differential, and discontinuous can be handled [19].

A traditional EA operates by first randomly initializing a population. Each member of the population can be considered a possible solution containing one or more parameter(s). The population is then evaluated against fitness criteria, which is also known as the objective function. Fit individuals are randomly selected for reproduction, creating a generation of offspring that can be considered fitter than their parents. After a number of subsequent iterations, an optimal individual (solution) can be identified. The fact that EAs contain within them aspects of random operation puts these methods in to a class known as stochastics.

Four traditional EA methods will be discussed in the subsequent sections. These will include genetic algorithms (GAs), genetic programming (GP), evolutionary strategies (ES), and evolutionary programming (EP). Most of the focus will be centered on GAs as they are applied in the current work. Subsequently, a discussion regarding simulated annealing (SA), the particle swarm approach (PSA), stochastic evolution (SE), tabu search (TS), and artificial neural networks (ANN) will also be carried out as these are related to EAs. First though, a summary of fundamental EA aspects will be carried out.

2.1.1.1 Fundamental Aspects of Evolutionary Algorithms

As mentioned above, EAs operate by evolving a population of solutions and guiding or evolving it toward an optimum. Thus, when it comes to operations like selection and reproduction, there exists some commonality between the four traditional methods. The current section will go over these aspects in some detail.

Individuals, which are represented by vectors or strings in EAs, must be evaluated somehow to determine if they are a poor or good solution. The level of how well a given individual satisfies a solution is called 'fitness'. This value is determined by evaluating the individual against a criterion that can determine its performance. When it comes to optimization with EAs, the objective function of the problem being assessed is used to achieve this. Essentially, the parameters of the individual are substituted into this governing equation and the output value is what defines one's fitness [20].

Since only the fittest individuals in a population are chosen to be reproduced in evolutionary methods, there must be a selection process in place that can achieve this. A total of three common selection schemes exist, which go as follows.

The first, probability based selection, is one of the simplest selection schemes available and is based on assigning probabilities to each individual in the population. By calculating the fitness of each individual (f_i) and the total fitness of ψ population ($\sum_{\psi} f_k$), the probability can be calculated as $f_i / \sum_{\psi} f_k$. This allows for individuals of lower fitness to be weeded out so that only the favorable members of the population are left for reproduction [21].

Moving on, the basic Roulette Wheel selection process is a stochastic or, in particular, stochastic sampling with replacement (SSR) procedure. This simply means that the sample size and assigned probabilities remain the same throughout the selection process, which goes as follows. A range is setup from zero to the sum of either all individual fitness values or fitness probabilities. Next, a random number is selected in this range and the individual whose range occupies the selected number is chosen and added to the mating pool. This process repeats until

there is a desired amount of individuals in the mating pool. The logic behind this form of selection is that fitter individuals will occupy a wider range thus having higher chances of getting selected [20].

Finally, tournament selection is another popular method used to identify the proper individuals for reproduction in EA methods. Here, a random number of (ϵ) individuals is chosen from the population at random and a tournament is initialized. Each individual in group (ϵ) goes up against another and the one with the higher fitness wins. This continues until there is only one individual left, at which point the individual is then added to the mating pool and the process is repeated. It should be noted that the larger (ϵ) is, the higher the fitness will be in the individual that wins the tournament [22].

When talking about reproduction in EAs, genetic operators are what govern this process. Depending on what particular EA we are talking about, either crossover, mutation, or both are used to create a new generation of individuals.

Even though there are several different types of recombination genetic operators, the most common types are single and multi-point crossover. These methods are based on two parent individuals cooperating with one another to produce two new offspring individuals. Shown in Figure 1 below is a simplified representation of single-point crossover [20].

Figure 1 – Single-Point Crossover

Here, the parents on the left can be represented by strings or vectors, depending on the EA method. What happens here is a random point along the individual is chosen and the parameters after this point are swapped or ‘crossed over’ when the offspring are produced, which are shown on the right in Figure 1. Multi-point crossovers on the other hand are shown below in Figure 2 [20].

Figure 2 – Multi-Point Crossover

In this case, several points are randomly chosen along the parent strings. Information is then exchanged between the selected regions when the offspring are produced. The end result of this type of operation can be seen on the right side of Figure 2. It should be noted that the probability of crossover is usually a parameter that is controlled in the algorithm by the user [20].

Just like the recombination operators mentioned above, there exist a number of different mutation procedures. In its most basic form, a mutation operation takes an individual and at a randomly selected location changes the value of the parameter. The value is chosen by random means and does not get calculated in any way. A reason for doing this is that the search space can be effectively increased to look for solutions that would be otherwise undiscovered. This kind of operation highlights the global search nature of EAs. As in the case of crossover, mutation probability is usually a user defined parameter, but is much smaller [20].

Elitism is another mechanism that is common to EAs. As populations evolve from generation to generation, it is not always guaranteed that the fittest individual is carried on since these methods are stochastic (i.e. based on random occurrences). Thus, elitism guarantees that the best individual (solution) is never lost and the evolutionary process can continue moving forward [23].

Lastly, termination criteria for EAs are usually difficult to determine as convergence may take a long time once the region of the global optima has been found. For this reason, EAs are terminated in most circumstances after a set amount of iterations (generations). Putting this control in the hands of the user means that judgment must be applied when choosing a proper end condition [20].

2.1.1.2 Genetic Algorithm (GA)

Genetic Algorithms were first presented systematically by Holland [24]. He based the principles of the algorithm to work on Darwin's law of survival of the fittest. Once created, it did not take long to realize that the method could be applied to solving optimization problems that were non-linear as well as having discontinuous variables.

The fundamental process requires first that a population of individuals be randomly generated. These individuals, also called chromosomes, can be made up of several parameters. Each parameter is represented by a binary string of numbers (i.e. 0's and 1's). An example of a GA chromosome is shown below in Figure 3 [25].

Figure 3 – GA Chromosome

From above, the total chromosome length is twenty and is made up of four parameters, each five digits long. Binary numbers can be converted to decimals (y_n) by using the formula shown below [25].

$$y_n = \sum_{k=1}^h 2^k B_k \quad \text{Eq.1}$$

Here, $k = 1, 2, \dots, h$ and B_k can be either 0 or 1. If upper (x^u) and lower (x^l) limits are applied to the binary number with (h) digits on the other hand, the decimal (x_n) can be computed by [25]:

$$x_n = x^l + \frac{x^u - x^l}{2^{h-1}} \sum_{k=0}^{h-1} 2^k B_k \quad \text{Eq.2}$$

If high precision is needed in an interval then more binary numbers (h) are required. For a desired resolution of dx , (h) can be found by [25]:

$$2^h = \frac{x^u - x^l}{dx} + 1 \quad \text{Eq.3}$$

Typically, GAs are solved by maximizing an unconstrained objective function. If minimization and/or constraints are required in a problem, two transformations must be applied. In the equation shown below, both have been applied [25].

$$F(x) = F_{max} - (f(x) + R \sum_{j=1}^m \Phi(g_j(x))) = F_{max} - f'(x) \quad \text{Eq.4}$$

By maximizing $F(x)$, the actual objective function $f(x)$ gets minimized, which is what is desired here. In order to make this work, F_{max} must be chosen such that it is larger than any possible value of $f(x)$. As far as the constraints are concerned, these are applied via penalty function Φ and are multiplied by the penalty variable R . Finally, function $g_j(x)$ represents the J^{th} constraint out of a total of m . Naturally, (x) is the design solution or individual being assessed.

Application of genetic operators, like crossover and mutation, can be controlled by probabilities defined by the user. This way the population can evolve with a high degree of control. Termination of the algorithm is also set by the user, which lets the GA evolve over a set amount of generations. A more detailed discussion on genetic operators and termination criterion can be found above in Section 2.1.1.1 [25]. For a more general overview of genetic algorithms please refer to Chipperfield et al. [20].

2.1.1.3 Genetic Programming (GP)

Genetic Programming is an optimization technique developed in the 1980's by John Koza [26]. The purpose behind GP is to create a method that can generate programs automatically via an evolutionary process. This is done by translating the code into a tree structure (program tree) so that it can be manipulated with ease. An example of this tree setup is shown below in Figure 4 [27].

Figure 4 – Program Tree

Since we are dealing with programs here, fitness of an individual (program tree) is not determined by evaluating an equation. Instead, the program tree is run under varying conditions and its behaviour is observed.

Each tree is composed of nodes and pointers, as seen above. Nodes that do not have any more pointers below them are called 'leaf nodes' or 'terminals'. These are basic functions that do not require any form of input data to carry out their intended purpose. Nodes which have pointers below them are termed 'operators'. There are three main types of operators: i) Arithmetic (+,-), ii) Boolean (AND, OR), and iii) Problem Specific (Average, Min, Max). Each operator has a number of successive nodes, also called an 'arity'. Combining together operators and terminals results in what is known as an 'area'.

That said, the GP at this point functions similar to a GA in that genetic operators are used to converge onto a solution. Individuals or program trees are randomly initialized and the fittest are selected for reproduction. This is usually done by a tournament selection process. Reproduction takes place by crossover, as shown below in Figure 5 [27].

Figure 5 – Crossover Reproduction

Crossover is carried out by essentially picking a random node on two program trees and swapping the areas as shown. Mutation operators are also introduced by choosing random nodes in individuals and generating random sub-trees. The aim here is to increase the search space so that important regions are not missed. Once the genetic operations have been completed, the process is repeated for a number of predefined iterations (generations).

In order to acquire results efficiently, GPs have large populations and are run with fewer generations. Also, the nature of GPs makes them rapidly increase in size or ‘bloat’. As a result, a significant amount of computation time is expected to determine an optimal solution [28].

2.1.1.4 Evolutionary Strategies (ES)

Evolutionary Strategies were developed by Rechenberg [29] and Schwefel [30] for the purpose of parameter optimization. In this method, individuals are represented by real numbers called object variables. These can be considered the actual parameters of the problem. Beyond this, individuals also carry items called strategy parameters. The purpose of the latter is to simply control the behaviour of mutation, which is the dominant operator in ESs. Strategy parameters carry out this control by using the variances and covariances of the object variables.

Like any other EA, individuals in this method are randomly initialized. Once this has been accomplished, fitness values are calculated for each individual via objective function and parents are randomly selected for recombination. At this phase, two parents come together to produce one new offspring. The object variables of the offspring are selected by simply taking all the object variables from one of the parents. New parents are randomly selected at this point and the averaged strategy parameters between the two parents are passed on to the offspring. Though other recombination operators exist, these methods have proven to be the most effective. Next comes the mutation operator, which as stated above is considered more important than the recombination performed earlier. First, changes are made to the strategy parameters of the offspring. Once this is complete, mutation of the object variables takes place and is controlled by distributions assigned to the now modified strategy parameters. At this point, the old population can be removed or kept with the new in subsequent iterations. The former is usually

done since the self-adaptive nature of ESs, which comes from the mutation of the strategy parameters, can be compromised when old members of the population linger. This cycle continues until a set amount of iterations have taken place [19]. Shown below in Figure 6 [19] is a summary of the procedure described above.

1. A current population of μ individuals is randomly initialized.
2. Fitness scores are assigned to each of the μ individuals.
3. λ new offspring are generated by recombination from the current population.
4. The λ new offspring are mutated.
5. Fitness scores are assigned to the λ new offspring.
6. A new population of μ individuals is selected, using either (μ, λ) or $(\mu + \lambda)$ selection.
7. The new population becomes the current population.
8. If the termination conditions are satisfied exit, otherwise go to step 3.

Figure 6 – Evolutionary Strategy Process

2.1.1.5 Evolutionary Programming (EP)

Evolutionary Programming, which was initially created by L. J. Fogal et al. [31], shares strong similarities to ESs. Here, each individual is also composed out of a string of real numbers. Evolution though, relies solely on a mutation operator and excludes recombination of individuals all together. Mutation is controlled here by way of a Gaussian probability distribution which is defined by a linear transformation of the parent individual's fitness value. The transformation is an item that the user must define in order to make this type of optimization method work. To avoid this, a derivative of EP called meta-EP can be applied that combines the problem variables with the distribution variances, thus making the process self-adapted.

At the start, the population is randomly initialized as in any other EA method. Each individual is then evaluated against the objective function so that fitness values can be determined. Mutation takes place next to create a new population, which is then followed by another fitness assessment. Selecting offspring for mutation is accomplished by tournament selection (refer to Section 2.1.1.1). Fitter individuals can thus be carried on by simply increasing the tournament size since a larger group will most likely generate a winner with a higher fitness. This form of selection also ensures that the best individual goes on to form the next population. At this point,

the process is repeated for a set number of iterations so that determination of a proper solution takes place [19]. Shown in Figure 7 [19] below is a summary of the method described above.

1. A current population of μ individuals is randomly initialized.
2. Fitness scores are assigned to each of the μ individuals.
3. The mutation operator is applied to each of the μ individuals in the current population to produce μ offspring.
4. Fitness scores are assigned to the μ offspring.
5. A new population of size μ is created from the μ parents and the μ offspring using tournament selection.
6. If the termination conditions are satisfied exit, otherwise go to step 3.

Figure 7 – Evolutionary Programming Process

2.1.2 Other Stochastic Heuristics

The following five methods, though they may not have the traditional structure of an Evolutionary Algorithm (e.g. recombination, mutation), can still be classified as related methods since they adapt and evolve throughout the optimization process. Simulated Annealing (SA), Particle Swarm Approach (PSA), Stochastic Evolution (SE), Tabu Search (TS), and Artificial Neural Networks (ANN) all fit in this category and are included below. It should be noted that all are stochastic methods that rely at least to some extent on random operators.

2.1.2.1 Simulated Annealing (SA)

Simulated Annealing is a global search approach developed in the mid 1980's that depends on random operations to find an optimum. The fundamental aspects are based on the process in which metals slowly cool from liquid to solid state. Here, as the temperature decreases, the internal energy of the particles making up the metal material also decreases. Eventually, if the cooling process was properly controlled, the particles will achieve a very stable structure that corresponds to the lowest possible internal energy. SA tries to copy this natural process and apply it to optimization problems [32].

Since the actual process can take a significant amount of time when the optimal outcome is required, Very Fast Simulated Annealing (VFSA) and Adapted Simulated Annealing (ASA) have been developed so that they can be applied in a practical way.

The method's favorable global search performance stems from the fact that it relies on random operations as well as allowing moves that increase the objective function. The probability of these moves can be calculated with Equation 5 [32].

$$p = \exp\left(\frac{-\delta f}{T}\right) \quad \text{Eq.5}$$

From above, (δf) is an increase in the objective function while (T) is a parameter unrelated to the objective function that represents the temperature which was explained above. In order to properly start and control the cooling process, an 'annealing schedule' must be present. Without this, the optimizations process has no framework to function off of. A simple flow chart of a SA procedure is shown below in Figure 8 [32].

Figure 8 – Simulated Annealing Flow Chart

SA does have some strengths that should be noted. For one, it is a very robust technique. More specifically, it can handle non-linear objective functions with constrained and discontinuous variables. Furthermore, it is well suited for global searches and does not get stuck in regions of local optimum. Thus, SA can handle complex search spaces. Lastly, SA is a very versatile method that can be applied to a wide variety of applications. In other words, it does not rely on a particular setup or model.

That said, SA also has some unworthy characteristics. First, SA works well in theory, but when the time comes to apply the method in a practical way, problems can arise. This is due to the fact that many choices have to be made to develop a working code. Next, the nature of SA means that there is an obvious trade-off between getting an answer in a respectable amount of time and getting an answer that is reliable. Thus, a compromise must be found between the two. Lastly, precision plays a large role in the accuracy of the final answer. As a result, a high degree of care must be taken when deciding the types of parameters to be used [32].

2.1.2.2 Particle Swarm Approach (PSA)

The Particle Swarm Approach, like SA, is a method this is related to EAs in that it adapts to a changing environment and evolves throughout the duration of the search. Initially developed by Kennedy and Eberhart [33], PSAs are based on swarming found in nature such as a flock of birds or a school of fish. In theory, these swarms are intended to move around and swarm the solution. This gives the method an obvious advantage of being able to find global optimums rather than getting stuck in local regions of complex, non-convex search spaces.

The way in which PSAs are implemented depends on the movement of particles. Each particle can be considered a possible solution to the problem. Instead of creating and eliminating particles (individuals) as seen in GAs, the population of PSAs stays constant. Instead, these particles are left to move around the search space so that an optimal solution can be found. As time goes on, it is hoped that the particles will move from regions of lower fitness and eventually swarm over a region of high global fitness. Just like most of the other EAs mentioned up to this

point, fitness is evaluated via an objective function. The duration of this method is a user defined parameter and depends on the problem being assessed.

During each iteration, a particle's position and velocity must be updated. The equation governing the updating of displacement takes the form of [34]:

$$x_{i,d}(t + 1) = x_{i,d}(t) + v_{i,d}(t + 1) \quad \text{Eq.6}$$

On the other hand, the equation used to update a particle's velocity is accomplished by [34]:

$$v_{i,d}(t + 1) = \omega v_{i,d}(t) + c_1 r_{1d}(t)[pbest_{i,d}(t) - x_{i,d}(t)] + c_2 r_{2d}(t)[lbest_{i,d}(t) - x_{i,d}(t)] \quad \text{Eq.7}$$

The equation of position is quite easy to understand as it is simply the previous position plus the change of position during the last interval, giving the new position. Velocity on the other hand is a bit more complex and requires some explanation. The first term found in the velocity equation is simply the current velocity multiplied by the inertial factor ' ω '. This parameter can be varied to achieve a different coverage of the search space. For instance, if $\omega > 1$, the particle is encouraged to look for other solutions outside of what it has already found. The second term represents the particle's best solution to date. Hence, it is this part of the equation that affects the velocity of the particle based on local experience. The third term alters the velocity based on the entire group's best solution to date. It should be evident at this point that each particle's position in the search space is affected by a complex model that takes into account important and necessary aspects.

In order to make it possible for particles to communicate with each other, a neighborhood must be set up. This not only defines the number or proximity of interactions, but also the pattern (social network structure) in which the interactions take place. Just to list a few patterns, these can include a star, ring, wheel, or pyramid setup.

When it comes to dealing with constrained problems, three options exist with PSAs. One is that the constraints can be ignored all together, which may result in a solution that is less than

adequate. The second deals with a penalty function, similar to the one mentioned in Section 2.1.1.2. This would alter the objective function and deter the search from unfeasible spaces. Lastly, feasibility preserving can be used to initialize the solution inside the feasible space. This might not always converge onto a solution inside the constraint limits, but at least it gives the PSA a good chance of staying within the specified bounds [34].

2.1.2.3 Stochastic Evolution (SE)

Stochastic Evolution is another method closely related to EAs. First proposed by Saab and Rao [35] in 1989, this method has good applicability to bisection problems [36]. The algorithm works by first creating an initial set of solutions by random means or known values from past work that generated a good solution. Characteristics of each member of the population are evaluated as to their sustainability in the current environment by various functions and operators. Each characteristic must prove its sustainability under the existing environment in order to be retained in the next generation, otherwise it is replaced by one that can achieve a better fitness. During each generation, trials are carried out that evaluate the various characteristics in the species. All fitness decreasing characteristics are carried over to the next generation automatically, but fitness increasing characteristics are accepted through stochastic means. This latter feature ensures that the algorithm does not get stuck in a local optimum. When a fitness increase is found, the number of trials is increased to give an incentive for the algorithm to search in that direction. This process continues on until no decrease in fitness is seen over a time period or until a specified number of iterations have been reached [36]. A simplified overview of the process is shown below in Figure 9 [37].

Algorithm SE:

```

 $S = S_0;$  /* initial state */
 $S_{\text{Best}} = S;$  /* save initial state */
 $p = p_0;$  /* initialize control parameter */
 $\rho = 0;$  /* initialize counter */
REPEAT
   $C_{\text{pre}} = \text{COST}(S);$ 
   $S = \text{PERTURB}_{\text{SE}}(S, p);$ 
   $C_{\text{cur}} = \text{COST}(S);$ 
  UPDATE ( $p, C_{\text{pre}}, C_{\text{cur}}$ );
  IF ( $\text{COST}(S) < \text{COST}(S_{\text{Best}})$ ) THEN
     $S_{\text{Best}} = S;$  /* save best state */
     $\rho = \rho - R;$  /* decrement counter by  $R$  */
  ELSE
     $\rho = \rho + 1;$  /* increment counter */
  ENDIF;
UNTIL  $\rho > R;$  /* stopping criterion */
RETURN ( $S_{\text{Best}}$ ); /* report best state */

```

Figure 9 – Stochastic Evolution Process

2.1.2.4 Tabu Search (TS)

Tabu search is a combinatorial based heuristic attributed to Glover [39], [40]. The fundamental principles of the algorithm are very easy to understand, but get considerably more difficult. Essentially, the method travels iteratively through a search space to find the best possible solution. The moves of algorithms in the search space are guided by both the short term (tabu list) and long term memory. Moves which increase the fitness functions are kept in a short term memory by marking them as ‘tabu’ and including them in a ‘tabu list’. These moves are avoided as long as they are present within the list. The length of this list is called the tabu tenure. As new moves are added to the tabu list, the old entries are cycled down and eventually removed so that they can be revisited again later. As for the long term memory, this is present to both intensify and diversify the search and acts as a more global decision making criteria. An overview of a simple tabu search process is shown below in Figure 10 [39]. For a detailed description of the tabu search algorithm, please refer to Glover [39], [40].

SIMPLE TABU SEARCH

1. Select an initial $x \in X$ and let $x^* := x$. Set the iteration counter $k = 0$ and begin with T empty.
2. If $S(x) - T$ is empty, go to Step 4. Otherwise, set $k := k + 1$ and select $s_k \in S(x) - T$ such that $s_k(x) = \text{OPTIMUM}(s(x): s \in S(x) - T)$.
3. Let $x := s_k(x)$. If $c(x) < c(x^*)$, where x^* denotes the best solution currently found, let $x^* := x$.
4. If a chosen number of iterations has elapsed either in total or since x^* was last improved, or if $S(x) - T = \emptyset$ upon reaching this step directly from Step 2, stop. Otherwise, update T (as subsequently identified) and return to Step 2.

Figure 10 – Tabu Search Process

2.1.2.5 Artificial Neural Networks (ANN)

Artificial neural networks (ANN) are a class of algorithms that mimic the biological nervous system. This method has been in existence for a long time as it was first introduced by McCulloch and Pitts [41] in 1943. In its simplest form, ANNs are made up of many neurons working in parallel that are all receiving inputs. These inputs are processed and transformed via weight functions to give a desired output. Figure 11 below shows the processing structure for each neuron [42].

Figure 11 – Neuron Processing Structure

In order to achieve the desired output, each neuron must be adjusted or trained so that a proper weight function can be defined. Not until after this setup process has been carried out can an artificial neural network produce the desired output.

Several different forms of architectures are possible when it comes to arranging the neurons. In its most basic form though, there exist a total of three layers that include input, hidden, and output as shown in Figure 12 [43].

Figure 12 – Neural Network Architecture

The architecture shown in Figure 12 constitutes a feed-forward direction since the outputs of each neural network is the input of the next layer. This kind of setup may be suitable for certain applications, but scenarios that have more dynamic characteristics may require a feedback connection to achieve the necessary output. These kinds of neural architecture are referred to as recurrent networks [43]. For a more thorough description of artificial neural networks, refer to Abraham [43].

2.2 Direct Partition Algorithms

Direct methods are algorithms that can partition an FE mesh fairly quickly without the use of iterative or optimization subroutines. Partition quality is not expected to be the best here, but at the least reasonable. As stated above in Section 1.1, a common method of categorizing direct partitioning algorithms is to place them under graph or engineering based methods. Graph based methods are ones which rely on graph theory to carry out partitions in FE models. Essentially what happens is FE elements are converted to a format more suitable for manipulation and

partitioning. This format requires each FE element to be converted to a vertex (node), which are connected together by members. Vertices can be new locations (such as element centroid) or coincide with a node location of the existing FE mesh. Each member has a total of two vertices, one at each end. The manner in which members are arranged depends on the type of element adjacency criteria. A dual graph has members join vertices only when the represented elements share a common edge. On the other hand, a diagonal dual graph contains members when the elements share a common node. These two types of graphs are shown below in Figure 13 for a simple problem with four node shell elements [4]. If the reader desires a more thorough description of graph theory, please refer to Kaveh [44].

Figure 13 – Graph Types

At the other end of the spectrum, engineering based partitioning methods function on different principles when compared to graph based methods. Here, the FE mesh is manipulated as is through algebraic means. There is no need to convert the mesh into a graph.

It must be made clear that categorization of partitioning methods under graph and engineering is not exclusive to direct partitioning algorithms and also applies to iterative/optimization based methods. The following subsections will give the reader a basic understanding of the available direct partition methods and show how the method used in the current work (separateness) compares to others.

2.2.1 Greedy Method

The first time this engineering based partitioning algorithm was proposed was in 1988 by Farhat [5]. In this work Farhat outlined the fundamental principles of what is today known as the greedy algorithm. This type of algorithm uses FE mesh connectivity information to carry out a

partition. First an initial element is chosen such that it has fewest adjacent elements. After this, the algorithm marks adjacent elements of a particular subdomain. Once the elements for one subdomain have been marked, i.e. the number of marked elements is $(\text{total elements})/(\text{number of subdomains})$, it marks elements for the next subdomain and so on. This process of marking resembles a biting action, hence the name 'greedy' algorithm. Figure 14 shown below depicts the expansion process as it starts from the initialization element [5].

Figure 14 – Greedy Algorithm Expansion Process

Several years after its first introduction Al-Nasra et al. [6] released a modified version of the greedy algorithm to take into account geometric properties of FE models. Essentially, this modified greedy method decreased the weight of expansion for elements in the direction of longer geometry to discourage expansion there. In other words, the expansion is not uniform for all conditions and has a bias in the direction of shorter geometry to produce subdomains with shorter interfaces. This modified expansion process is shown in Figure 15 [6].

2.2.3 Inertial Method

Farhat et al. [8] briefly discussed the applicability of inertial methods to FE mesh partitioning problems in 1993. These engineering partitioning methods rely on the definition of a direction u based on principle inertial directions of a FE mesh. The principle inertial directions refer to the eigenvectors I_1 , I_2 , and I_3 that correspond to the eigenvalues of the inertial matrix shown below [8]:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \quad \text{Eq.8}$$

Where,

$$\begin{aligned} I_{xx} &= \sum_{i=1}^{i=N_n} (y_i - y_{cg})^2 + (z_i - z_{cg})^2 \\ I_{yy} &= \sum_{i=1}^{i=N_n} (x_i - x_{cg})^2 + (z_i - z_{cg})^2 \\ I_{zz} &= \sum_{i=1}^{i=N_n} (x_i - x_{cg})^2 + (y_i - y_{cg})^2 \\ I_{xy} &= I_{yx} = - \sum_{i=1}^{i=N_n} (x_i - x_{cg})(y_i - y_{cg}) \\ I_{xz} &= I_{zx} = - \sum_{i=1}^{i=N_n} (x_i - x_{cg})(z_i - z_{cg}) \\ I_{yz} &= I_{zy} = - \sum_{i=1}^{i=N_n} (y_i - y_{cg})(z_i - z_{cg}) \end{aligned}$$

Once u is calculated, all other nodes are orthogonally projected onto it. The nodes are then sorted along u and collected to generate the required subdomains. An alternative form to this method is to recursively carry out this process, which includes projection, sorting, and collection.

Implementation of these types of algorithms often results in very slender subdomains, albeit with less interface nodes compared to results from the minimum bandwidth method [8].

2.2.4 Recursive Bisection Method

The first major introduction of recursive bisection methods with respect to mesh partitioning was done by Simon [9] in 1991. In his work he outlined 3 variations of the graph based recursive bisection algorithm that differed by the manner in which the bisection was defined. Only the first method (recursive co-ordinate bisection) based on vertex co-ordinates can be considered a truly direct method since the other two (recursive graph bisection and recursive spectral bisection) rely on iterative procedures to define the best bisection path. Thus, it should be obvious that these types of algorithms apply to both direct and iterative/optimization categories.

The recursive co-ordinate bisection method is a very simple process that first defines a principle direction for bisection (x , y in 2D and x , y , z in 3D) based on which co-ordinate has the longest component of geometry. Once the direction is found, all vertices in the graph are sorted with respect to it, i.e. one side is assigned to one subdomain while the other side is assigned to another [9]. A summary of this process is shown below in Figure 16 [9].

-
(1)	Determine longest expansion of domain (x -, y -, or z -direction)
(2)	Sort vertices according to coordinate in selected direction
(3)	Assign half of the vertices to each subdomain
(4)	Repeat recursively (divide and conquer)
-

Figure 16 – Recursive Co-ordinate Bisection Process

2.2.5 Separateness Method

Separateness based direct partitioning methods can be attributed to Kaveh and Roosta. These algorithms were first introduced in the late 1990's in both an engineering [10] and graph [3] format. Even though differences exist between the two methods, they rely on the same principle.

The first step in separateness based partitioning algorithms is to define subdomain initialization elements. This aspect is carried out by locating an element with minimal degree (number of adjacent elements). Next, the furthest most possible element in the FE model is found. If only two subdomains are to be constructed, the initialization phase is ended. On the other hand, if

three or more subdomains are to be constructed, initialization elements are chosen such that maximum distance exists between them. This is the reason why these methods are called ‘separateness’ based [3, 10].

Once all initialization elements are identified, expansion takes place sequentially. Each subdomain is given a weight, which can be the number of elements in the subdomain, degrees of freedom in the subdomain, or some other value. A subdomain with minimum weight is then identified and an unmarked element adjacent to it is marked based on minimum separateness, number of adjacent elements in the current subdomain, and number of adjacent elements in other subdomains. Separateness here is a summation of distances calculated during the initialization phase between the element in question and all other initialization elements. Once an element has been chosen, subdomain weights are updated and the expansion process is repeated until all elements in the model have been marked [3, 10].

This algorithm is more versatile than the recursive bisection method since it is capable of producing any number of subdomains, not just 2^q . Furthermore, by using this theory of ‘separateness’, subdomains are more likely to retain acceptable aspect ratios due to there being relatively equal amounts of unmarked elements around each initialization element.

2.3 Iterative/Optimization Partitioning Algorithms

Iterative/optimization based partitioning algorithms are different from the direct type. Generally speaking, these take more compute power to generate subdomains in a FE mesh. Partition quality on the other hand is usually higher in terms of interface node count and load imbalance. In some cases, iterative/optimization methods use direct methods as an initial step [4, 16] to generate an improved solution. An aspect that both methods share in common though is that they can be graph or engineering based. The subsections to follow summarize three common groups of iterative/optimization partitioning methods, which include finite element interconnecting and tearing (FETI), recursive bisection, and two-step types. Furthermore, a quick overview will be given at the end of this section regarding algorithms which do not fit under the three main groups but are still classified as iterative/optimization based. The intent

here is to not only provide background information on the method applied here (two-step), but also to give some idea as to how it compares to iterative/optimization based partition methods.

2.3.1 Finite Element Tearing and Interconnecting (FETI) Methods

The roots of the finite element tearing and interconnecting (FETI) partition algorithm can be traced back to Farhat et al. [12, 49, 50] in the early 1990's. From its inception, FETI methods have been created for the purposes of solving non-linear FE problems in parallel and sequential computer architectures for applications including static, dynamic, structural, and acoustic problems [53]. An important feature of the method is that it is numerically scalable, that is, the convergence rate is asymptotically independent of the number of elements within the problem [52].

The FETI algorithm is an iterative domain decomposition method based on Lagrange multipliers, which are applied to the subdomain interface regions to enforce displacement continuity. During iteration, equilibrium is achieved in each subdomain, but the continuity constraints are only achieved when global convergence is found. Each iteration encompasses a two-step preconditioned conjugate gradient (PCG) algorithm. The first (precondition) step deals with solving subdomain problems with Dirichlet boundary conditions. Once this has been accomplished, related subdomain problems containing Neumann boundary conditions are solved by means of direct (FEM solver) methods. The second step mentioned allows for the computation of residuals, which are necessary for global convergence [48]. This two-step process is also known as the dual Schur complement method due to the fact that upon initialization, a dual Schur complement operator is constructed [51].

Many derivatives of this method have been added to literature since its initial release. Modified versions of the FETI algorithm include applications for second-order elasticity/fourth-order plate and shell problems [50, 54, 55, 56]. On top of this, FETI methods have also been applied to frictionless contact problems [58]. Recent work [53] has focused on making improvements to the continuity constraints.

2.3.2 Recursive Bisection Methods

One of the first major publications on recursive bisection methods was in 1991 by Simon [9]. In this work, he outlined three variants of this algorithm which included recursive coordinate bisection, recursive graph bisection, and recursive spectral bisection. The only fundamental difference between these three methods comes down to the manner in which the split or bisection is defined. As discussed in Section 2.2.4, recursive coordinate bisection is the simplest form of this method and falls under the direct partitioning group. On the other hand, the other two versions fall under the iterative/optimization heading and are briefly described below [9].

As is obvious by the name, recursive graph bisection is a graph based partitioning method. Thus, the first step in executing this algorithm is to convert the existing FE mesh to a graph (vertices and members). Once this has been carried out, the diameter of the graph is found, which is defined as the longest path between two extremal vertices. A path in this work refers to the shortest number of members between the two vertices in question. With the two vertices defined, a level structure is constructed that is rooted from one of the vertices. All that this means is the graph distances (# of members) to all vertices are found. The last step is then executed, which assigns half of the closest vertices to the initialization vertex, thus generating the bisection. The only somewhat difficult step in the recursive graph bisection method is to define the best diameter of the graph and generate the level structure. Fortunately, established iterative methods like the reverse Cuthill-McKee are capable of carrying this out [9]. If more subdomains are required at this point, the algorithm is recursively applied. A brief overview of the steps discussed here is shown below in Figure 17 [9].

-
- (1) Use of SPARSPAK RCM algorithm to compute a level structure
 - (2) Sort vertices according to the RCM level structure
 - (3) Assign half of the vertices to each subdomain
 - (4) Repeat recursively (divide and conquer)
-

Figure 17 – Recursive Graph Bisection Process

Similar to the recursive graph bisection, the recursive spectral bisection method is based on graph theory which means the first step in this process is to convert the FE mesh to vertices and

members. The similarity ends here though, as a modified version of the Lanczos algorithm is applied to the Laplacian matrix of the graph. What is of desire here is the second eigenvector (also known as the Fiedler vector, x_2) due to the directional information it gives for all vertices in the graph via weights. Thus, this modified Lanczos algorithm iteratively finds only the second eigenvalue (λ_2) of the Laplacian matrix of the graph to save compute effort. By sorting the vertices according to the calculated weights, an efficient bisection path can be determined to generate two new subdomains. If more subdomains are required, this method is applied recursively of course. A summary of the steps mentioned here is shown in Figure 18 [9].

-
- (1) **Compute Fiedler vector for graph using the Lanczos algorithm**
 - (2) **Sort vertices according to size of entries in Fiedler vector**
 - (3) **Assign half of the vertices to each subdomain**
 - (4) **Repeat recursively (divide and conquer)**
-

Figure 18 – Recursive Spectral Bisection Process

In the past decade, several new partitioning methods have been proposed featuring genetic algorithms to find optimal bisection [15] and trisection [57] paths based on a minimization criterion that deals with the Laplacian matrix. Other methods have incorporated artificial neural networks to find an optimal solution to the bisection problem [59].

2.3.3 Two-Step Methods

The most basic form of the two-step method was developed by Vanderstraeten et al. [4], [60] in the mid 1990's. Based on a graph approach, the algorithm goes as follows.

First, the FE mesh is converted to a graph via vertices and members. The graph structure can either be a dual or diagonal dual composition, though it has been found that the latter produces better quality partitions [4]. Once a graph is generated, a direct partitioning method is applied to give a quick partition. As an alternative, an iterative/optimization based method can also be applied here if it can be carried out with low compute effort. The goal here is to get a partition quickly, not to get the best possible partition quality. In the case of [4] a modified recursive

graph bisection method was applied, but a greedy [5] or separateness [3] based method can be used just as easily.

Once an initial partition is generated, it is used as an initial solution for a non-deterministic heuristic method such as simulated annealing, stochastic evolution, or tabu search [4]. Elements on the subdomain interfaces are perturbed between their original and neighbouring subdomains in an effort to achieve a smaller node interface count while increasing the subdomain load balance. A visual representation of this perturbation is shown below in Figure 19 [4].

Figure 19 – Element Perturbation

The reason for limiting the search space to elements adjacent to the subdomain interfaces is to keep the complexity of the optimization problem low while obtaining improvements on the initial partition. Otherwise, the problem becomes not only too large to solve, but non-convergent as well.

Two separate versions of the algorithm mentioned here were introduced by Vanderstraeten et al. [4]. One was based on optimization of topologically general decomposition while the other was for optimization of topologically one-dimensional decompositions. The latter is more applicable to problems with narrow/slender geometry [4].

About the same time Vanderstraeten et al. [4, 60] introduced the above algorithm, several similar methods were proposed. Bouhmala et al. [16] for one, focused on a method which used a greedy algorithm for the direct phase, and compared three algorithms for the optimization phase that included Kernighan-Lin, simulated annealing, and a random matching heuristic (RMH). The

search space for this work was treated similarly to [4] by grouping elements along the subdomain interfaces into pairs and perturbing them [16].

Farhat et al. [61] also introduced a two-step method based on a mesh partitioning program for parallel processing called TOP/DOMDEC. Various types of direct methods could be used here which included greedy, bandwidth, inertial, and recursive bisection algorithms. Optimization heuristics on the other hand included simulated annealing, stochastic evolution, and tabu search. A total of eight aspects of the partition in this work could be chosen for optimization [61].

2.3.4 Other Iterative/Optimization Based Partitioning Methods

Several other iterative/optimization partitioning algorithms exist that do not fit into the previous three sections. Some examples of such methods are briefly discussed below.

One of the earliest applications of simulated annealing to graph partitioning problems was by Johnson et al. [62] in the late 1980's. This study compared the graph partition optimization potential of the simulated annealing method to the Kernighan-Lin method. Implementation of simulated annealing in this case was for the most part straight forward. A cost function was developed which took into account the interface size and imbalance of the graph. Each vertex of the graph was set to a variable and initialization was carried out randomly [62].

Ghosal et al. [63] showed that an artificial neural network could be successfully implemented on its own for graph partitioning problems. A competitive learning approach was used to calibrate the weight matrix of the network and achieve the desired partitioning results. Inputs for the network were the elements themselves, outputs consisted of the assigned subdomain, and the input pattern was defined in such a way to reflect the element adjacency [63].

Walshaw et al. [64] approached the graph partitioning problem in a different way than most of the common approaches. In this work, an initial partition was carried out by constructing a graph and then coarsening it until only 'q' vertices are left (q is the number of desired subdomains). Once the initial mesh has been constructed, the graph is then expanded back to its original state

while keeping the subdomains intact. At this point, the Kernighan-Lin optimization algorithm is iteratively applied with a cost function based on subdomain aspect ratios. The optimization process continues for a user defined number of iterations before it is terminated [64].

Recently, Kaveh et al. [65] released a study regarding an optimization based partitioning algorithm called p-median methodology. In this algorithm, an FE mesh is converted to graph and coarsening is applied by way of a heavy-edge matching algorithm. Once the coarse graph has been attained, an optimization heuristic is applied to create what are called p-medians. This particular study used two optimization methods, that of a standard ant colony (AC) method and hybrid ant colony-genetic algorithm (ACGA) method. The later method is primarily based on ant colony optimization and uses the genetic algorithm to speed up its performance. With the p-medians defined, the vertices on the medians are defined as the interface and the subdomains are expanded to complete the partition [65].

Another very recent partitioning method can be attributed to Kaveh et al. [66]. This work looked at combining graph theory with differential equations to generate partitions faster than previous methods. Essentially, the eigenvalue problem common to many graph based partitioning methods was converted to an ordinary differential equation problem. The bounds of the initial solution point of the differential equation were determined by implementing the Wittrick-Williams algorithm. By using this alternate method to compute eigenvalues, many well established and advanced numerical methods can be applied to solve partitioning problems with greater ease [66].

2.4 Wingbox Structural Optimization

This section is meant to give the reader some background information regarding wingbox structural optimization. By providing this information, it is hoped that a better understanding of the optimization problem in the current work will be attained. The past studies discussed here fall into two main categories. The first category to be discussed is that of multidisciplinary optimization. Here, more than one engineering discipline is used to evaluate the model during optimization. Most often, wingbox problems include computational fluid dynamics (CFD) to

calculate the aerodynamic loads and finite element methods (FEM) to determine the displacements/stresses in the structure. Non-multidisciplinary methods on the other hand contain only one engineering discipline, which in this case is purely structural. Loads are determined here by means of approximations or assumptions, not extensive calculation such as CFD. This latter optimization method was applied in the current work.

2.4.1 Multidisciplinary Optimization (MDO)

A brief overview of four recent MDO studies will be carried out below to give the reader an idea of this type of optimization problem.

The first study to mention is that of Klimmek et al. [67], which was introduced in 2002. In this work, both the topological geometry and member thicknesses were considered design variables. Initialization of the problem began with a series of steps, starting with planform area definition, followed by airfoil cross-sections at specific spans of the wing. Once this was done, the front spar was located at a constant chord percentage. The rear spar location was determined by control surface size requirements. Rounding out the initialization process was a definition of rib and additional spar (if it is a multi-spar wing) locations. Once an initial setup was defined, the optimization process was carried out using existing tools for aero-elasticity, structural analysis, and design optimization, which included ZAERO, LAGRANGE, and iSIGHT respectively [67].

Also in 2002, Schuhmacher et al. [68] released a MDO study centered on wingbox sizing for the new Fairchild Dornier regional jet aircraft (at the time). This was a very extensive effort as a detailed model of the entire fuselage and wing was generated that included thousands of design variables ranging from strength, to aero-elasticity, to manufacturing. Applied to this were a series of loading conditions spanning from gusts to static limit loads. Many scenarios were looked at such as buckling, flutter, fatigue, and ultimate strength. To reduce complexity, the same FE model was used for aero-elastic and static problems. A series of secondary programs had to be developed to manage all the data generated during the optimization process. These sub-programs were based around MSC Software GmbH since the optimization process was conducted with MSC Nastran SOL 200 [68].

In 2004, Venter et al. [69] proposed a MDO method that utilized a particle swarm optimization heuristic. This work used both discrete as well as continuous design variables in the form of wing planform area and thicknesses of structural members (i.e. skin, spars, ribs, etc.). The processes began with an initialization of variables and an update of the FE structural model. Next came the aerodynamic analysis for pressure calculations, which was handled by a custom made approximation method. The pressure distribution was then transferred over to the FE model and evaluated using software called GENESIS. After the stresses were assessed, the aerodynamic drag was then found along with the aircraft range. Finally, the particle swarm optimization algorithm was applied to assess the solution and change design variables as required [69].

The last and most recent MDO application to mention is that of Wunderlich [70]. Here, the wing optimization problem was applied specifically to a transonic mission. Design variables included sizing aspects such as sweep angle, aspect ratio, thickness and twist angle of wing. During each optimization iteration, an aerodynamic grid was generated by a software called Centaur. The aerodynamic problem itself was handled by a transonic flow solver known as DLR TAU-Code. Structural grids on the other hand were generated using PARA_MAM, also from DLR and run in ANSYS to assess fatigue and limit strengths. To transfer over the aerodynamic data to the structural model, interpolation software called MESH2MESH was utilized. Once aerodynamic and structural data was obtained, the optimization algorithm (Subplex) modified the design variables as required to close in on the global optimum. This process was repeated until a predetermined termination criteria was met [70].

2.4.2 Purely Structural Wingbox Optimization

This section will give some insight regarding purely structural wingbox optimization methods. Here, aerodynamic loads are defined via simple approximations or assumptions, that is, only structural aspects of the problem are evaluated during the optimization process. A total of four recent works will be overviewed and go as follows.

McIlhagga et al. [72] carried out a study in 1996 regarding a variety of search algorithms applied to a wingbox optimization problem. The design scenario used was a static condition at a load factor of 2. Variables for the problem included the number of equi-distant ribs (all the same thickness) as well as the thickness of the top skin panels. The objective was to minimize the weight of the wingbox as much as possible while ensuring that the top skin panels avoid buckling due to the compressive load. To carry out the optimization process, several deterministic and stochastic search algorithms were applied, which included random search, brute force search, gradient descent, simulated annealing, and a distributed genetic algorithm among a few others [72].

The same year, Ewing et al. [73] released a paper regarding the application of a genetic algorithm and gradient search method to a wingbox optimization problem containing a rectangular cross-section. Design variables in this study included skin thicknesses, spar cap cross-sectional areas, stringer areas, and stringer spacing. A total of three scenarios were used to demonstrate the potential of the GA, including a single load case/fixed rib spacing, multiple load case/fixed rib spacing, and single load case/variable rib spacing. Constraining the problem was carried out by way of bending and buckling analysis during the evaluation of each possible solution [73].

More recently in 2004, Yao Shen et al. [74] conducted an optimization study on the wingbox for an F-16 follow-on design. The wing structure included 2 spars and 4 stringers with the design variables defined as the location of each of the six structural members. Given a constant cross-sectional area, the volume and thus the weight of each member varied according to location in the swept wing. Applied aerodynamic loads were constant throughout the optimization process and based on approximated chord-wise values and an elliptical distribution span-wise. To constrain the problem, both bending and shear aspects were assessed and applied via constraint function. As for the actual optimization method, an unconventional comparison technique was used by comparing the optimal solution to a known ‘bad’ arrangement [74].

The final wing structure study, which was also released in 2004, was that of Krog et al. [71] from Airbus UK. Here, a topological optimization was carried out on ribs for the Airbus A380

commercial aircraft. First a global analysis was carried out on the entire wingbox at critical load scenarios to define loads in the region of a specific rib. Another load analysis was then done in the local region to obtain load information for critical scenarios. A detailed topological FE model of the rib was then constructed which included both the interface and local loads. The optimization analysis was then carried out with commercial codes (Altair) by moving/removing material within the boundaries of topological mesh in order to obtain the lowest possible weight [71].

3. Background Information

With a good overview of stochastic optimization techniques and partitioning algorithms now acquired, an in-depth description will be provided in this section on methods that the current paper is primarily based on. More specifically, Section 3.1 will discuss the direct partitioning method by Kaveh et al. [3] and Section 3.2 will go over the two-step optimization partition method by Vanderstraeten et al. [4].

3.1 Separateness Direct Partitioning

The overview of separateness based partitioning methods described in Section 2.2.5 showed the basic processes used in this technique. In this section, more elaboration will be made regarding the graph based version proposed by Kaveh et al. [3].

Before any discussion can be made, some initial concepts must be understood. The graphs used in this method are made up of members and nodes (also known as vertices). Adjacent nodes are connected to one another by members based on certain adjacency criteria. Distance in graph theory refers to the number of members in the shortest path between two nodes (path being a sequence of members that do not cross). A spanning root tree (SRT_{n_0}) is a series of paths rooted from a specific node such that each node of the graph has the shortest possible path to the root node (n_0). SRT's expand through the entire graph and contain all nodes. In the context of a SRT, contours are nodes which have equal distance to the root node. Finally, separateness $S(n_0, n_i)$ between node n_i and node n_0 can be defined as the number of nodes selected in the creation of contours of SRT_{n_0} before node n_i is reached. In other words, as SRT's are generated contour by contour, the total number of selected nodes before n_i is reached is the separateness $S(n_0, n_i)$ [3]. To illustrate this concept, Kaveh et al. describe a simple example shown below in Figure 20 [3]. It should be noted that $S(n_1, n_2) = 4$ while $S(n_2, n_1) = 3$.

Figure 20 – Simple Graph

Being a method based on graph theory, the first step required in this direct partition algorithm is to convert the FE mesh to a graph. This task is carried out here in multiple steps by using various graph representations of the FE mesh to achieve the final desired layout, i.e. an element clique representative graph. To start off, all adjacent nodes in the FE mesh are connected by members. Adjacency here is defined as nodes which share a common element edge. This gives what is known as the skeleton graph. An arbitrary node is then chosen and a spanning root tree (SRTv) is rooted from it to obtain node w on the last contour with minimum degree (# of adjacent nodes). Another SRT is then carried out, rooted from w to obtain node n_o on the last contour with minimum degree. An SR subtree is then executed from n_o contour by contour until a node from each element in the FE mesh is selected. To obtain the representative graph, all non-selected nodes are deleted. The last step in the graph generating process is then to connect all nodes that represent elements in the FE mesh which share a common FE node [3]. The various graphs mentioned here can be seen in Figure 21 [3]. The individual diagrams represent: a) the FE mesh, b) the element clique graph, c) the skeleton graph, d) the representative graph, and e) the element clique representative graph

Figure 21 – Element Clique Representative Graph Generation Process

Once the element clique representative graph has been constructed, the element initialization process may begin. The intent here is to of course choose nodes that have the maximum number of elements between one another to allow for proper expansion of the subdomains, as shown in Figure 22 below [3].

Figure 22 – Subdomain Initialization Elements

The first step in this process is to choose an arbitrary node in the graph and form a SRT from it to find node r_1 with minimum degree on the last contour. Once this is established, SRT_{r_1} is performed and the separateness from r_1 to each node n_i $S(r_1, n_i)$ is determined along with the node r_2 on the last contour with minimum degree. The same process is carried out from node r_2 . If only two subdomains are to be generated, the initialization process ends here. Otherwise, node r_3 is determined in the graph with the maximum total separateness, i.e. summed separateness rooted from r_1 and r_2 . After r_3 has been found, the separateness $S(r_3, n_i)$ is determined and the total

separateness is once again updated so that r_4 can be selected. This process continues until the number of initialization nodes is equal to the number of desired subdomains (q).

With the subdomain initialization nodes (elements) defined, the expansion process may now begin. First, each subdomain is assigned a weight. This may be the number of elements, total degrees of freedom, or some other parameters. Next, a specific subdomain k is identified that has minimum weight. An unmarked node n_p is then chosen that is connected to a marked node from subdomain k that has maximum value for the least total separateness, minus the number of nodes adjacent to n_p from other subdomains, plus the number of nodes adjacent to n_p from subdomain k . Once n_p has been selected, the subdomain weight is updated and the process is repeated until all elements in the FE mesh have been assigned a subdomain via marked nodes in the element clique repetitive graph [3].

3.2 Partition Optimization

This section is meant to give the reader a more detailed description of the two-step partition optimization method proposed by Vanderstraeten et al. [4], beyond what was said in Section 2.3.3.

Since the two-step method is based on a direct partition followed by a partition optimization, a direct partition method must first be defined. In Vanderstraeten et al. [4], a modified recursive graph bisection (RGB) method was used for this portion of the partitioning process. Actually, two versions of this method were used which included one for topologically general decomposition and one for topologically 1-D decomposition. Both methods though, require construction of a graph G from the initial FE mesh containing N elements. Even though a dual ($G' = V', E'$) as well as diagonal dual ($G'' = V'', E''$) graph can be used here, Vanderstraeten et al. [4] found that the diagonal dual graph (G'') achieved better performance (V and E refer to sets of vertices and edges, respectively). A pseudo-code for one bisection of G into G_1 and G_2 using a topologically general decomposition version of the RGB algorithm is shown in Appendix D.

In the pseudo-code, distance refers to the number of edges (members) in the shortest path between two vertices. Diameter on the other hand is a pair of vertices (n_1, n_2) that have maximum distance in graph G .

A slightly different version of the RGB is used for topological 1-D partitioning. Here, subdomains are constructed sequentially. For instance, assume that at some point in the partition process subdomains 1 through j have been created. To create subdomain $j + 1$, all unselected vertices beside subdomain j are assigned to subdomain $j + 1$ to ensure 1-D topology. Next, a modified RGB is applied such that approximately N/q elements are assigned to subdomain $j + 1$. This is carried out by constructing a diameter with vertex n_1 located on the interface of subdomain j and $j + 1$. A pseudo-code of this topologically 1-D partitioning process is provided in Appendix D.

Once the direct partition has been carried out, the partition optimization can begin. This phase of the process is carried out by defining the search space as the elements adjacent to the subdomain interfaces. Reasoning behind only choosing these elements was based on forming a problem that could be practically solved. Otherwise, the solution might take too long to obtain or even end up being non-convergent. During the optimization process, elements are perturbed between their original and adjacent subdomains as was shown above in Figure 19 [4] (Section 2.3.3). To allow for evaluation of each possible solution, a cost function was proposed for a minimization problem that took into account both the interface size as well as load imbalance of the subdomains [4] as shown below.

$$Cost = \alpha(\text{Interface Size}) + (1 - \alpha)(\text{Load Imbalance}) \quad \text{Eq.9}$$

The value α is known as the penalty parameter and can be varied from 0 to 1 to achieve a different weight between the importance of interface size to load imbalance. In the first term, ‘interface size’ refers to the number of edges ‘E’ whose vertices belong to more than one subdomain. As for load imbalance, this term is expanded below in Equation 10 [4].

$$Load\ Imbalance = \sum_{k=1}^q [N_k - N_{best,k}]^2 \quad \text{Eq.10}$$

As can be seen from Equation 10, load imbalance is quantified via a summation function for each subdomain. Essentially, the difference between the ideal number of elements ($N_{best,k}$) is compared to the actual number of elements (N_k) in each subdomain. If any differences exist then they are recorded and tallied. Determining $N_{best,k}$ depends on the type of balancing method used. For an equicut method, $N_{best,k}$ is simply N/q . On the other hand, if a frontal solver is assumed, a more complicated set of criteria must be met. In this case, $N_{best,k}$ is found such that $N_{best,k}[W_{RMS,k}]^2$ is independent of k and all $N_{best,k}$ values sum up to N [4].

Carrying out the actual optimization phase of the process are three non-deterministic heuristics which include simulated annealing, stochastic evolution, and tabu search. The optimization structure is based on twin nested loops. The inner loop perturbs the feasible solutions, which are selected based on criteria specific to each heuristic. The outer loop on the other hand controls the perturbations, also in such a way that is unique to each heuristic. For simulated annealing, both perturbation and acceptance of solutions are stochastic steps. Stochastic evolution on the other hand has perturbation that is deterministic with acceptance that is stochastic. Finally, the opposite is true for tabu search.

To measure the load imbalance of subdomains, a parameter known as the load balancing factor (LBF) is defined as follows [4].

$$LBF = 1 - \max_k \left| \frac{Load_k - Load_{av}}{Load_{av}} \right| \quad \text{Eq.11}$$

Essentially, this parameter is a measure of the ratio between the maximum load imbalance to the average load. For a perfectly balanced partition, $LBF = 1$. In cases where very large imbalances exist, the LBF might even be negative. As for the definition of 'Load', this again depends on the type of balancing method used. For an equicut approach, the load is simply the number of elements in the k^{th} subdomain, i.e. $Load_k = N_k$. When a frontal balancing method is used, this definition is changed to $Load_k = N_k[W_{RMS,k}]^2$ [4].

4. Current Work

This section will outline in detail the proposed two-step partitioning algorithm. Section 4.1 will discuss a modified separateness direct partition method based on Kaveh et al. [3]. Section 4.2 will then go on to describe a new partition optimization method based on Vanderstraeten et al. [4] that uses a genetic algorithm as the optimization heuristic. An overview of the governing process that controls the partitioning algorithm will be given in Section 4.3 based on the assumption that it will be used in a wingbox structural optimization problem along with a frontal solver. Finally, Section 4.4 will show the specifics as to how each portion of the algorithm was implemented.

4.1 Direct Method

The direct partitioning method applied in this work was one based on the concept of ‘separateness’ [3]. To show the general applicability of the method, it was decided that the algorithm must be able to produce any number of desired partitions in FE meshes that contain 2 node spar, 3 or 4 node shell, and 8 node solid elements. The following three sub-sections will outline the main steps required in executing the proposed direct partitioning method.

4.1.1 Convert Finite Element Mesh to Graph

Being a graph based method, the FE mesh is imported and converted to a graph with vertices located at the element centroids. Members are then created that join each of the vertices together, i.e. each member has two vertices (one at each end). The manner in which the members are joined depends on the element adjacency criterion used. Two options exist for this aspect of graph generation which includes a dual and diagonal dual graph, as discussed in Section 2.2. A visual description of a dual and diagonal dual centroid graph is shown in Figure 13 for a simple FE mesh with 4-node shell elements. This process differs slightly from Kaveh et al. [3] since a centroid graph is generated instead of an element clique representative graph.

4.1.2 Determine Subdomain Initialization Elements

Once the centroid graph has been constructed, a vertex r_1 corresponding to an element is selected with the lowest degree (number of adjacent elements). Next, a spanning root tree (as defined in Section 3.1) is carried out that is rooted from r_1 (SRT_{r1}) to find a vertex r_2 on the last contour with minimum degree. Once SRT_{r1} has been carried out, another is initiated from the vertex r_2 (SRT_{r2}) and vertex r_3 is found. If only two subdomains are being generated, the initialization step ends here with elements corresponding to vertices r_2 and r_3 being the initialization elements for subdomain 1 and 2 respectively. Otherwise, another vertex r_4 is chosen such that it has the maximum distance (as defined in Section 3.1) to the nearest vertex already selected. This is carried out by calculating the distance from each vertex in the model to the vertices already selected during each SRT generation. Vertex r_4 is then assigned to subdomain 3, and this last step is repeated until the total number of selected vertices is equal to the number of desired subdomains.

The intent of this initialization process is to find vertices with maximum ‘separateness’ so that proper expansion can take place. By having the startup vertices (elements) as far as possible from one another, it is more likely that the generated subdomains will maintain proper aspect ratios and avoid being stretched out in any way [3]. Figure 23 below shows an arbitrary FE shell mesh with three initialization elements chosen.

Figure 23 – Initialization Elements

Kaveh et al. [3] approached this stage of the direct partition process slightly differently. The first two initialization elements were found the same way, but the subsequent ones were not. Here,

separateness was calculated at each element for each SRT. The third expansion element was then found by calculating the maximum superimposed separateness as defined in Section 3.1. Another SRT was then carried out from the third initialization element with separateness recalculated and added to the existing values for each element (vertex).

4.1.3 Sequentially Expand Subdomains

The final step in the direct method proposed here is to expand the subdomains from the initialization elements defined in the previous step. Shown below are the processes that accomplish this task:

- a) A subdomain k is chosen such that it has minimal weight. Weight can be defined as the number of elements, degrees of freedom, or any other criteria attained by a subdomain in the expansion process. At the beginning, usually all weights are the same so the numerically lowest subdomain is selected.
- b) Once subdomain k has been chosen, the algorithm narrows down the possible expansion element to the lowest contour relative to the initialization element that still contains unmarked elements. The nature of the contours is of course dependent on the type of graph (dual or diagonal dual) being used.
- c) Once a contour has been selected, an expansion element is chosen such that its vertex has the shortest co-ordinate distance to the vertex corresponding to the initialize element for subdomain k . The selected element must not have more adjacent elements from subdomains other than k . If this is the case, the next closest element is chosen. By including this logic, the chances of broken and chaotic subdomain interfaces are reduced significantly when the expanding subdomains begin to interact with one another.
- d) Once an element has been marked for expansion, subdomain lists and weights are updated. The process returns back to step a) and is repeated until all elements in the model have been assigned to a particular subdomain.

Staying consistent with Figure 23, the expanded partition takes the shape of Figure 24 shown below.

Figure 24 – Subdomain Expansion

The expansion process described here has the same fundamental characteristics as Kaveh et al. [3], but is carried out in a slightly different way. The same subdomain weighting is used to pick which subdomain (k) is to be expanded, but the selection of the actual expansion vertex (element) is done by alternative means. Instead of expanding contour by contour as in the current method, an unmarked vertex adjacent to a marked vertex in subdomain k is chosen such that it has maximum value for separateness, minus the number of adjacent vertices belonging to other subdomains, plus the number of adjacent vertices in subdomain k .

4.2 Optimization Method

Optimization of the initial partition is carried out here based on work done by Vanderstraeten et al. [4]. A fundamental difference exists with the optimization technique used though. In the original work, stochastic evolution, tabu search, and simulated annealing were applied to achieve a better partition. In the current work a genetic algorithm is applied iteratively. The following three subsections describe each part of the partition optimization process in greater detail.

4.2.1 Optimization Initialization

It is important for the search space of the optimization problem to be properly defined. If too many variables are chosen, the problem can be too large to solve practically and still achieve time savings. In this work, all elements adjacent to the subdomain interface are set as variables in the optimization problem as per Vanderstraeten et al. [4]. Figure 25 identifies the elements included in the search space, which are filled in white.

Figure 25 – Partition Optimization Search Space

4.2.2 Partition Optimization

Once a search space has been defined, the optimization problem may commence. Based on Vanderstraeten et al. [4], the elements identified for optimization are perturbed between their original and adjacent subdomains in order to achieve a better partition. That is, the variables in the GA problem are the subdomain numbers. The fitness function used in this process is a modified version of Equation 9 that uses the FE interface node count in the first term instead of the edge (member) count from the graph. This updated fitness function is shown below via Equation 12.

$$Fitness = \alpha(\text{Interface Node Count}) + (1 - \alpha)(\text{Load Imbalance}) \quad \text{Eq.12}$$

The default value for the penalty parameter α used for this work was 0.95, as suggested by Vanderstraeten et al. [4]. As far as the second term was concerned, no modifications were made to 'Load Imbalance' as defined in Equation 10. When evaluating this part of the fitness function,

there are two main options that have to be defined. Both of these options are discussed in the following sections.

4.2.2.1 Balancing Item

In certain applications a FE mesh may have only one type of element, meaning the compute cost of each element is the same. For these situations, it is possible for N_k and $N_{best,k}$ to represent the number of elements and the ideal number of elements in the k^{th} subdomain respectively, as was done by Vanderstraeten et al. [4]. Many applications will not have one element type though. In this kind of situation, the element degrees of freedom will have to be applied to account for computational difference between the various elements. Thus, N_k and $N_{best,k}$ must represent the degrees of freedom and the ideal degrees of freedom required in the k^{th} subdomain. When dealing with the load balancing factor (LBF) discussed in Section 3.2, one must be careful to define $Load_k$ properly based on the balancing item used, i.e. the proper value for N_k must be chosen. Unlike Vanderstraeten et al. [4], both element and degree of freedom balancing items were looked at in the current work.

4.2.2.2 Balancing Method

In Equation 10, N_k can be easily found for a given solution through inspection. On the other hand, $N_{best,k}$ must be calculated based on one of two possible methods outlined below.

The equicut method is the easiest way to balance the subdomains during the optimization process. Here, $N_{best,k}$ is the same for all subdomains and is equal to N/q where N is the total number of elements or sum of subdomain degrees of freedom (depending on what balancing item is used). This method should be applied for general applications where it is not known what FE solver will be used to evaluate the problem.

It can be shown that the compute load for a frontal solver is proportional to $N_k[W_{RMS,k}]^2$ where $W_{RMS,k}$ refers to the root mean square wavefront of the k^{th} subdomain [75] (for more information on frontal solvers, please refer to Appendix B.3). Wavefront is very dependent on the order in which the elements are numbered. In ANSYS, an automatic element reordering method is

applied [76] based on work carried out by Hoit et al. [77]. As an alternative, a geometric reordering algorithm [76] can also be applied to reorder the elements of a FE model. In this work, both methods are applied and the lowest $W_{RMS,k}$ value is chosen. Once this has been found, $N_{best,k}$ is determined such that $N_{best,k}[W_{RMS,k}]^2$ is independent of k and $\sum_{k=1}^q N_{best,k} = N$ [4]. The attained $N_{best,k}$ values are then substituted into Equation 10 and ultimately Equation 12 to allow for calculation of the fitness values in the GA optimization process.

4.2.2.3 Fitness Penalty

Since elements on both sides of the subdomain interfaces were used in the partition optimization process, it was necessary to ensure that subdomains were not discontinuous in any way (i.e. split into 2 or more parts). This aspect of the algorithm was handled in the fitness function module of the GA. Every time a new solution was evaluated with respect to fitness, a SRT was carried out from each subdomain initialization element. Reasoning behind this was to see if all elements in the subdomain were marked. If a discontinuity was present, several elements in the subdomain in question were left unmarked due to the discontinuity. The fitness function was then multiplied by 100 times the unmarked element count to prevent the GA from accepting this type of poor partition. This method was not carried out in Vanderstraeten et al. [4] and is unique to the current work.

4.2.2.4 Termination Criteria

Termination of the optimization process is based on a stall generation approach. If after 50 generations the fitness value of the best individual in the GA does not improve, then the optimization process ends. An ultimate limit of 1000 generations is applied to limit the process to a finite number, with no time limit given. The intent here is to allow for the GA to find the optimal partition without prematurely ending the optimization process.

4.2.3 Update Solution

Once the optimal partition has been found, the best solution is then set as the current partition and the process is repeated (back to Section 4.2.1). This outer or iterative loop is continued until

an end criterion, dependent on the balancing method, has been achieved. For the equicut method, there must be a stall of three iterations before the entire optimization process is ended. Three stall iterations are used since this method has a very convergent fitness value due to the fact that $N_{\text{best},k}$ values remain constant during the entire process.

The wavefront balancing method on the other hand is terminated after the first time the fitness value is seen to increase. Here, $N_{\text{best},k}$ values are re-calculated each time the GA optimization is applied. Thus, convergence with this method is harder to achieve. Outer iterations are terminated after an increase in the fitness is witnessed to ensure that the best fitness value does not start to diverge as further iterations are carried out.

The process mentioned above was developed in this work and is not based on Vanderstraeten et al. [4].

4.3 Governing Process

With the partitioning algorithm defined, a governing process unique to the current work must be put in place that controls when to use the algorithm, how many subdomains to generate, and how long to apply the partition optimization phase. The last point is required since in some applications the benefits of partitioning may be small when compared to evaluating the problem by standard means, thus the partition optimization process cannot run for a long enough time to achieve the optimal partition. The aim is to create a code that automatically determines the most computationally efficient optimization evaluation setup given some particular inputs, which include FE model, boundary conditions, FE solver, optimization constraints, GA population, and GA generations.

In order to define this governing process though, some important decisions must be made regarding the application. That said, it was assumed that the problem would be a static wingbox structural optimization carried out with a genetic algorithm for a defined population size and generation count. Furthermore, it would be assumed that the partitions would only be run in series on a single processor and not in parallel on several.

The manner in which this governing process would be conducted would be one based on reference data for a generic case to approximate certain time intervals. This required reference data would include:

- a) The time required to evaluate a standard FE wingbox problem (t_s) as a function of element count
- b) The time required to perform a direct partition (t_d) as a function of element count for a varying number of subdomains (q)
- c) The time required to evaluate a partitioned FE wingbox problem (t_p) as a function of element count for a varying number of subdomain counts (q)

Once this reference data is acquired, the governing process may be calibrated so that any type of static wingbox optimization problem can be carried out with the best possible time savings. The governing process used in this work is shown below as a flow chart in Figure 26.

Figure 26 – Governing Process Flow Chart

From above, the first step is to estimate the total FE evaluation time required for a non-partitioned model during the structural optimization ($t_{s,tot}$). This is done via the reference data as a function of the number of elements in the current model, the GA population, and generation count. Next, the direct partition time (t_d) and total FE evaluation time required for a partitioned model during the structural optimization is determined ($t_{p,tot}$), also as a function of the element count, the GA population, and generation count. It should be noted that $t_{p,tot}$ is based on only a direct partition and does not include benefits of partition optimization. Furthermore, the number of subdomains (q) is determined by looking at which number achieved the lowest t_p in the reference data. With the critical times obtained, it is possible to determine whether or not there are any benefits in partitioning the FE model for the given element count, GA population size, and GA generation count. This aspect is found by subtracting the $t_{p,tot}$ from $t_{s,tot}$.

If the remaining time is still more than t_d , then the partition process is initiated. Otherwise, a standard GA optimization is carried out with no partitioning. Once the partitioning process is started, the direct partition is carried out immediately. Subsequently, the partition optimization is executed until it reaches its termination criteria or until it reaches a time limit equal to $0.1(t_{s,tot} - t_{p,tot} - t_d)$, i.e. 10% of the approximated time savings. This ensures that the partition optimization process does not take large time out of the possible time savings. On the other hand for very large models or large GA populations/generations, this criterion allows the partition optimization phase to reach its own termination criteria and achieve an optimal partition. The last step is then to carry out the structural optimization with the partitioned FE model.

4.4 Implementation

Two commercial tools were used to apply the algorithm described in the sections above, which included Matlab R2007a and ANSYS V11. Matlab was used for the actual programming of the code, with the genetic algorithm portion handled by a built in module (GA Toolbox). FE aspects of the algorithm on the other hand were conducted through ANSYS, which was called up by the Matlab program whenever required. The overall structure of this algorithm is shown in Figure 27 along with the program used to carry out each process.

Figure 27 – Flow Chart of Code

Starting at the top, the governing module is handled by Matlab. The FE mesh is imported here along with the desired size of the GA population and generation count for the structural optimization problem. If it is found that partitioning is not beneficial, the standard optimization module (lower left side of Figure 27) is called up which uses Matlab's GA Toolbox with custom mutation, creation, and fitness functions. If it is deemed that partitioning is of benefit, the partition module on the right side of Figure 27 is executed. A copy of the Matlab code used in this module can be found in Appendix A.

The first thing the partition module does is import all required information, including the number of subdomains (q), the node list, and element list. Next, the direct partition module is called up to obtain a direct partition. This module returns a matrix containing the elements assigned to each subdomain. Moving down the right side of Figure 27, the partition optimization loop is entered. At this point, the code calculates $N_{\text{best},k}$ values required for the partition optimization process. This calculation is trivial if the equicut balancing method is used. If, on the other hand, the wavefront balancing method is used, ANSYS must be called upon to calculate the minimum wavefront of each subdomain. In order to do this, the Matlab code creates an input file in ANSYS Parametric Design Language (APDL) that is run in ANSYS batch mode. For a more detailed explanation of APDL, please refer to Appendix B.1. All measures were taken to avoid repetition of unnecessary commands to minimize the time required for ANSYS to calculate the wavefront value. As stated in Section 4.2.2.2, two ANSYS functions are used to determine minimum wavefront and the lowest value is chosen. ANSYS exports the results into a text file that is then read by the partition module in Matlab. At this point the search space is identified, and the partition optimization problem is run by way of a modified version of the GA Toolbox. Custom mutation, creation, and fitness functions are of course used here. Once the GA has finished, the partition is updated based on the results of the optimization process and the algorithm loops back to carry out another optimization iteration if the termination criteria or time limit have not been reached. If the termination conditions or time limit have been reached, the structural optimization module is called up (lower right part of Figure 27). Before the partition optimization module is ended though, the optimal partition and wavefront reorder method are both exported to text files.

Once in the partitioned structural optimization module, the Matlab code imports all necessary information which includes the optimal partition, wavefront reorder method, node list, element list, real constant list, material model, analysis type, constraints, nodal forces, pressure forces on elements, and inertial loads. Based on this information, the input files needed for structural evaluation are generated for ANSYS in APDL. Since the partitioning process uses the substructuring function in ANSYS to run the partitions in series, q 'generation' input files, one 'use' input file, and q 'expansion' files were needed (a total of $2q + 1$ files). An overview of the ANSYS substructuring function used here can be found in Appendix B.2. With the input files generated, the GA Toolbox is executed for a defined population and generation count. Extreme care was taken to minimize the complexity of the code. An important feature was to only edit the design variables in the input files instead of creating a new input file every time. Once the input files are updated, ANSYS is called upon in the fitness module to determine stresses and displacements, which are summarized in a text file. The fitness module then imports the maximum displacement and stress to determine whether or not they have been breeched. Fitness is adjusted accordingly via penalty function (discussed further in Section 5.1) if it is found that the constraints are violated. Final results of the structural optimization are printed in the Matlab command prompt as well as being exported to a file. The process mentioned in this paragraph is carried out for the non-partitioned structural optimization module as well, other than only requiring a single input file since no partitioning is needed there.

5. Wingbox Optimization Problem Setup

A wingbox weight optimization problem was performed to evaluate the effectiveness of the proposed partitioning algorithm. The primary objective here was to demonstrate that the partitioning algorithm could achieve noticeable time savings compared to a standard GA structural optimization process. A secondary objective was to attain realistic results from the structural optimization problem itself. The reference aircraft used to model the wing structure in this analysis was the Piaggio P-180 Avanti II, which was applied to a steady-state, limit load scenario. A load factor of $n = 3.253$ was applied here, as defined by FAR 23.337. The method used to calculate load factor can be found in Appendix C. Shown in Figure 28 is a cutaway of the wing structure of the Piaggio P-180 Avanti II [78].

Figure 28 – Cutaway View of P180 Wing Structure

5.1 Optimization Setup

In all, a total of 29 design variables were used in this structural optimization problem, which included the thicknesses of 19 ribs, 2 spars, 6 strings, and 2 skins. Discrete values were used for all ribs consistent with standard industry thicknesses [79]. As for the spars, stringers and skin sections, continuous thicknesses from 0.000 – 0.050 m were defined since these members were machined in the P180 reference aircraft [80]. To constrain this optimization problem, stress and displacement limits were set to 2.3e8 Pa and 0.6 m respectively. The stress limit was based on the yield limit of Aluminum Alloy 2024-T3 [81] with a safety factor of 1.5 as per FAR 23.303. Since aluminum is a ductile material, the maximum equivalent tensile stress (von Mises stress) was used and compared to the yield limit [82]. To enforce the constraints, a penalty factor was applied to the fitness function values proportional to the sum of the stress and displacement violation. The penalty factor and fitness function are shown below in Equation 13 and Equation 14 respectively.

$$Penalty\ Factor = \left(\frac{\sigma_{max}}{\sigma_{max,allow}} + \frac{d_{max}}{d_{max,allow}} \right) + 1 \quad Eq.13$$

$$Fitness = (Penalty\ Factor)(Total\ Wingbox\ Weight) \quad Eq.14$$

In Equation 13, σ_{max} , $\sigma_{max,allow}$, d_{max} , and $d_{max,allow}$ refer to the maximum von Mises stress (in model), maximum allowable stress (2024-T3 yield stress), maximum vector sum displacement (in model), and maximum allowable vector sum displacement respectively. The penalty factor is only applied if either or both constraints have been violated; otherwise it defaults to a value of 1. As for the GA parameters, a population of 20 individuals was used and evaluated for 125 generations. The population is based on calibration carried out for the partition optimization phase, which is discussed in Section 6.2.

5.2 Finite Element Model

Even though the FE model was based on the Piaggio P180, some simplifications were made. These included equal spacing of the ribs, approximating the wingbox cross-section as rectangular, and omitting the third (secondary) spar located behind the rear spar from the wing root to nacelle. Reasoning behind the last item was due to the high number of elements needed for a refined model with this kind of geometry. With an element count exceeding 30,000, the structural optimization would take far too long, even with the use of partitioning.

Dimensions of the wingbox were acquired via P180 wing dimensions as well as scaling orthogonal views [80]. It was assumed that the stringer height was 0.020 m based on cutaway views [78]. Main dimensions of the wingbox model can be found in Figure 29 below.

Figure 29 – Wingbox Dimensions

The finite element model used in this analysis was constructed out of 24,007 4-node shell elements (SHELL63 in ANSYS) and 23,764 nodes. Element edge length for the wingbox was 0.020 m, but was reduced to as low as 0.000005 m at the corners of the root to capture the high stress gradients in those regions. This size was confirmed with a mesh validation that is summarized in Figure 30. From below, it can be seen that at an element edge length of 0.000005 m the maximum von Mises stress had more or less stabilized.

Figure 30 – Wingbox Mesh Validation

Material properties were based on Aluminum Alloy 2024-T3, which gave a modulus of elasticity, Poisson's ratio, and density of 7.24×10^{10} Pa, 0.33, and 2770 kg/m^3 respectively [81]. A picture of the FE model used here can be seen in Figure 31 with the upper and lower skin omitted to allow visual access to internal structural members.

Figure 31 – FE Model of Wingbox with Skin Panels Missing

5.3 Boundary Conditions

As stated above, this wingbox optimization problem was based on a steady-state limit load scenario. First off, all root nodes of the wingbox were constrained in all 6 degrees of freedom (UX, UY, UZ, RX, RY, RZ), as shown in Figure 32.

Figure 32 – FE Model Constraints

An elliptical lift distribution was assumed [83] and applied at the $\frac{1}{4}$ chord location on the upper wing surface along 305 nodes which corresponded to $n = 3.253$. At each of these node locations, a very small induced drag component was also included based on the calculated lift vectors [83]. The PT6A-66B engine weight [84] was evenly distributed at 3 hard points on the upper wing surface that coincided with rib locations as seen in the cutaway view [78]. A fuel weight corresponding to 462.3 L [78] was applied as a uniform pressure along the lower skin. Lastly, the effects of structure weight due to gravity were also included in the analysis. Figure 33 and Figure 34 shown below depict the elliptical lift distribution and the complete boundary conditions applied to the FE wingbox model. For a thorough explanation of the loads calculated here, please refer to Appendix C.

Figure 33 – Elliptical Lift Distribution

Figure 34 – Complete Wingbox Boundary Conditions

6. Validation/Calibration

With the partition algorithm now defined, many aspects and parameters must be validated or calibrated before the code is applied to an example. Sections 6.1 to 6.5 will explain in detail the validation of the Matlab GA Toolbox, calibration of main GA parameters, calibration of the direct partition method, calibration of the partition optimization method, and calibration of reference data for the governing process, respectively.

6.1 Matlab GA Toolbox Validation

Before the Matlab GA Toolbox could be used in any way, it had to be confirmed that the algorithm could achieve accurate results when compared to other Genetic Algorithms. Thus, a validation process was carried out based on two very common models in structural optimization, that of the 10 bar truss and 25 bar truss problem [87]. Since inconsistencies with respect to the design variables existed amongst all the studies which carried out these two problems in the past, both discrete and continuous variables were applied in this work at varying numbers of GA population. The goal here was to vary the cross-sectional area of the bars to achieve the lightest possible weight, given certain displacement and stress constraints. Good correlation was witnessed between the GA Toolbox and past works.

6.1.1 10 Bar Truss

The 10 bar truss problem (shown in Figure 35) is one of the easiest applications one can apply to validate an optimization algorithm. This six node structure is constrained at two nodes in x and y, with 100,000 lbs loads acting in the negative y direction from another two nodes. Material properties used for this model can be found in Table 1. To constrain the problem, a maximum displacement and stress of 2.0 in and 25 ksi was applied respectively. As for the input variables, discrete values were used according to Lewis [87] along with a continuous variable set from 0.1 to 33.5 in. Table 2 and Table 3 summarize past results and current results, respectively.

Figure 35 – 10 Bar Truss Problem

Table 1 – Material Properties for 10 Bar Truss

Modulus of Elasticity [psi]	1×10^7
Density [lb/in ³]	0.1

Table 2 – Previous 10 Bar Truss Optimization Results

Case	Gellatly [89]	Venkayya [90]	Schmit [91]	Qian [92]	Belegundu [93]	Hajela & Berke [94]	Ali [88]	Camp [85]	Camp [85]	Lewis [87]	Lewis [87]
Population	NA	NA	NA	NA	NA	NA	NA	20	40	20	40
Generations	NA	NA	NA	NA	NA	NA	NA	NA	NA	50	50
A_1 [in]	22.21	23.416	23.76	23.545	25.7	30.774	22.48	30	26.5	22	22
A_2 [in]	15.6	14.904	14.56	14.96	0.1	0.112	14.94	15.5	13.5	14.2	13.9
A_3 [in]	0.24	0.53	0.53	0.297	25.11	17.4	0.42	1.8	1.62	1.8	1.99
A_4 [in]	0.1	0.128	0.1	0.1	19.39	11.425	0.14	1.8	1.62	2.13	1.62
A_5 [in]	31.35	30.416	30.67	30.902	0.1	0.108	30.75	30	30	30	30
A_6 [in]	0.1	0.101	0.1	0.1	0.1	0.487	0.12	1.99	1.62	1.62	1.62
A_7 [in]	22.06	21.084	21.07	21.275	15.4	5.593	21.1	19.9	22.9	26.5	19.9
A_8 [in]	8.35	8.578	8.578	7.611	20.32	22.953	7.62	11.7	7.22	13.5	11.5
A_9 [in]	0.14	0.101	0.1	0.1	20.74	20.886	0.13	1.62	1.62	1.62	1.62
A_{10} [in]	20.03	21.077	20.96	21.156	1.14	0.1	20.74	19.9	22	18.8	26.5
Weight [lb]	5112	5084.9	5076.85	5069.4	5472	4692.49	5054.98	5607	5556.9	5659.1	5623.3
Constraint Violation	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes

Table 3 – Current 10 Bar Truss Optimization Results

Parameters	Discrete					Continuous				
Case	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA
Population	5	10	20	40	80	5	10	20	40	80
Generations	3000	3000	2000	2000	1000	3000	3000	2000	2000	1000
$A_1 [in]$	22.90	19.90	15.50	19.90	18.80	23.63770	22.68563	28.05833	29.58048	25.07170
$A_2 [in]$	16.00	18.80	18.80	11.50	15.50	23.97289	14.70371	14.50366	16.05419	14.54323
$A_3 [in]$	1.62	3.09	1.62	2.13	1.62	0.47936	0.10000	0.15304	0.55489	0.10169
$A_4 [in]$	1.62	4.80	1.62	1.62	1.62	0.10001	0.10000	0.10000	0.10000	0.10000
$A_5 [in]$	26.50	19.90	30.00	26.50	30.00	30.54361	29.76535	30.13274	28.62203	30.74995
$A_6 [in]$	1.62	1.62	2.88	1.62	1.62	0.84425	0.10000	0.10000	0.10000	0.10000
$A_7 [in]$	30.00	22.90	33.50	22.90	33.50	21.37417	23.37486	18.97573	20.03587	19.97384
$A_8 [in]$	11.50	26.50	3.84	15.50	7.97	5.46394	8.84300	7.87644	6.90416	8.33367
$A_9 [in]$	1.80	11.50	1.62	2.38	1.62	0.10006	0.10001	0.10000	0.10000	0.10006
$A_{10} [in]$	18.80	22.00	30.00	30.00	19.90	22.03487	20.06195	21.91686	20.55647	21.45875
Weight [lb]	5690.98	6672.54	6045.99	5881.25	5696.69	5358.10400	5095.11400	5117.72500	5123.63500	5082.77500
FEM Time [s]	86.073	170.969	228.004	456.926	457.998	85.900	174.018	234.124	470.247	474.208
Total Time [s]	204.828	296.172	310.437	540.016	493.985	235.766	327.688	329.547	570.594	524.297
Constraint Violation	No	No	No	No	No	No	No	No	No	No

From the two tables above, several items can be seen. For one, many of the past results were seen violating the stress or displacement limits. This check was carried out by plugging each solution into ANSYS. The current results for a discrete setup gave values for weight a little higher than the reference data, but it must be pointed out that no constraints were violated. As for the continuous input model, very good correlation with the past works was witnessed, also with no constraint violations.

6.1.2 25 Bar Truss

Another very common optimization test problem is that of the 25 bar truss, shown in Figure 36. This model used the same material properties as the 10 bar truss problem discussed in Section 6.1.1, but with a different layout, design variables, and constraints. First off, even though there were 25 bar members in the model, there were only 8 variables. The reason for this was that the cross-sectional areas were grouped together in the setup shown in Table 4 to increase the number of members while using a search space similar to that of the 10 bar truss problem. Similar to the 10 bar truss problem, both discrete and continuous inputs were exploited. The discrete range was kept consistent with Lewis [87] while the continuous values ranged from 0.1 to 3.8 in. Boundary conditions included nodes 7, 8, 9, and 10 being constrained in x, y, and z components,

with the loading setup defined below in Table 5. Optimization constraints on the other hand were limited to 0.35 in and 40 ksi for the displacement and stress respectively. Table 6 and Table 7 summarize past results and current results, respectively.

Figure 36 – 25 Bar Truss Problem

Table 4 – Design Variable Grouping

Group	Elements
1	1
2	2,3,4,5
3	6,7,8,9
4	10,11
5	12,13
6	14,15,16,17
7	18,19,20,21
8	22,23,24,25

Table 5 – Loads Applied to 25 Bar Truss Optimization Problem

Joint	Co-ordinates		
	X [psi]	Y [psi]	Z [psi]
2	1000	-10000	-10000
3	0	-100000	-10000
3	500	0	0
6	600	0	0

Table 6 – Past 25 Bar Truss Optimization Results

Case	Coello [95]	Adeli& Park [96, 97]	Ali [88]	Rajeev [86]	Rajeev [86]	Lewis [87]	Lewis [87]
Population	NA	NA	NA	20	40	20	40
Generations	NA	NA	NA	NA	NA	50	50
A_1 [in] (1)	NA	0.6	0.1006	0.20	0.10	0.10	0.10
A_2 [in] (2-5)	NA	1.4	0.1029	1.80	1.80	1.30	0.20
A_3 [in] (6-9)	NA	2.8	3.4103	2.30	2.30	2.50	3.40
A_4 [in] (10,11)	NA	0.5	0.1086	0.20	0.20	0.30	0.10
A_5 [in] (12,13)	NA	0.6	1.8284	0.10	0.10	0.30	1.80
A_6 [in] (14-17)	NA	0.5	0.74813	0.80	0.80	0.90	0.90
A_7 [in] (18-21)	NA	1.5	0.1	1.80	1.80	0.90	0.60
A_8 [in] (22-25)	NA	3.0	3.7166	3.00	3.00	2.80	3.60
Weight [lb]	493.94	543.95	459.82	546.76	546.01	517.70	478.57
Constraint Violation	NA	Yes	Yes	No	No	Yes	No

Table 7 – Current 25 Bar Truss Optimization Results

Parameters	Discrete					Continuous				
	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA	Matlab GA
Case										
Population	5	10	20	40	80	5	10	20	40	80
Generations	2000	1000	500	500	250	2000	1000	500	500	250
A_1 [in] (1)	0.2	0.1	0.2	0.1	0.1	0.10138	0.14081	0.11028	0.10139	0.12209
A_2 [in] (2-5)	1.8	2.3	2.3	0.6	1.8	1.56724	0.54480	0.53878	0.24696	0.33160
A_3 [in] (6-9)	3.6	2.3	1.9	3.6	3.0	2.73524	3.30498	3.24963	3.61035	3.41763
A_4 [in] (10,11)	0.2	0.3	0.1	0.1	0.1	0.10564	0.11816	0.10491	0.10000	0.10081
A_5 [in] (12,13)	0.1	0.1	0.7	0.8	1.5	1.39022	0.91324	1.57363	1.74930	1.91110
A_6 [in] (14-17)	0.9	0.6	1.0	0.6	0.8	0.76047	0.73523	0.86832	0.79613	0.85675
A_7 [in] (18-21)	1.7	1.5	1.1	0.9	0.5	0.12224	0.71420	0.25678	0.19599	0.17977
A_8 [in] (22-25)	2.3	3.2	3.2	3.6	3.2	3.79821	3.72267	3.77443	3.79255	3.78805
Weight [lb]	564.9	548.1	537.7	500.2	511.9	488.58700	489.91300	475.97500	469.97900	471.73600
FEM Time [s]	240.204	244.746	242.089	484.237	479.713	250.625	248.137	248.883	495.452	499.316
Total Time [s]	314.312	276.406	258.046	501.046	488.000	336.484	285.313	267.578	516.047	510.922
Constraint Violation	No	No	No	No	No	No	No	No	No	No

Looking at the results in Table 7, it can be seen that both the discrete and continuous models attained similar if not better results than most of the past works shown in Table 6. Once again, it was found that several of the previous results did in fact violate at least one of the constraints as

determined with an ANSYS validation. None of the solutions generated by the GA Toolbox violated any constraints, which only underlines its capability as an optimization heuristic.

6.2 GA Parameter Calibration

As stated before, the GA Toolbox from Matlab R2007a was used in this work and modified as required. The GA has quite a few parameters within it, so calibration was necessary to ensure it was running to the most of its potential in the partition optimization phase. The model used to perform this work was a simple rectangular box structure with both ends open, as shown in Figure 37.

Figure 37 – GA Calibration Model

In all, 1,300 4-node shell elements were used with 1,326 nodes. Three optimization iterations were carried out with 8 subdomains ($q = 8$) and the best fitness value of two runs was used. The most critical parameters that were assessed included the GA population and mutation rate. These were varied from 5 – 50 and 0.1 % - 3.0 %, respectively. Other parameters were also varied, but kept to their default values as no significant benefits were witnessed using alternate configurations. Evaluation of the various methods was based on fitness function value as well as compute time. Table 8 below shows a summary of the GA calibration, with the best configuration highlighted. For a description of each of the parameters listed in Table 8, please refer to [98].

Table 8 – GA Calibration Results

Parameter	Final Fitness	Total Time [sec]
GA Population = 5	668.35	77.44
GA Population = 10	635.75	92.29
GA Population = 15	606.05	110.98
GA Population = 20	567.55	128.33
GA Population = 25	546.25	146.28
GA Population = 30	541.60	164.25
GA Population = 35	535.30	181.13
GA Population = 40	540.70	201.91
GA Population = 45	540.35	220.95
GA Population = 50	506.00	240.61
Mutation Rate = 0.001	618.75	130.16
Mutation Rate = 0.005	524.60	129.67
Mutation Rate = 0.010	539.25	129.00
Mutation Rate = 0.015	545.30	127.53
Mutation Rate = 0.020	581.05	128.53
Mutation Rate = 0.025	575.75	129.08
Mutation Rate = 0.030	593.45	127.86
Fitness Scaling = Linear Shift	582.95	128.63
Fitness Scaling = Proportional	607.55	128.34
Fitness Scaling = Top	586.80	127.83
Fitness Scaling = Rank	595.30	128.36
Selection = Stochastic Uniform	604.90	129.22
Selection = Remainder	627.95	129.53
Selection = Uniform	685.65	125.84
Selection = Roulette	621.00	130.42
Selection = Tournament, 2	566.40	132.39
Selection = Tournament, 3	585.55	130.34
Selection = Tournament, 4	574.60	130.32
Elite Count = 1	563.35	129.41
Elite Count = 2	574.25	129.19
Crossover = Scattered	573.10	127.94
Crossover = Single Point	594.00	128.24
Crossover = Two Point	583.60	128.88

6.3 Direct Partition Calibration

For the direct partition process, the two main parameters that had a large impact on performance included expansion element choice (closest or furthest element to the initialization element) and the graph used (dual or diagonal dual). Both of these dealt with the expansion process of the subdomains, so the quality of the partition was greatly affected. It should be noted that selection of both parameters in this section was based on the desire for durability of the algorithm rather than achieving the best possible partition quality for specific applications.

6.3.1 Expansion Element Choice

This aspect essentially defined the expansion on a given contour of a subdomain during the expansion process. Depending on which method was used, the interfaces could have noticeable differences in quality. Simple models were used to assess a ‘furthest element’ option, but research of this aspect ended quite quickly as errors were encountered. It is believed that this type of expansion created a conflict with giving priority to elements that had many neighbors in other subdomains. Thus, it was decided that the ‘closest element’ option would be implemented for all subsequent work.

6.3.2 Graph Choice

In many applications the diagonal dual graph yielded very promising results. Aspect ratios of the generated subdomains were for the most part maintained and the interfaces were reasonable. On the other hand, simple rectangular meshes with more slender geometry showed a problem when using this sort of graph, as shown below in Figure 38 for a mesh with 26,000 shell elements.

Figure 38 – Expansion Results for Diagonal Dual and Dual Graphs

Looking at the upper mesh of Figure 38, it is clear that the subdomain on the far left had a poor interface with its neighbors. Due to this issue with expansion, it was decided that the dual graph would be applied to the direct partition algorithm, even if it may generate subdomains that are less balanced in certain applications. It was assumed that the inclusion of the partition optimization algorithm would make the last item mentioned less important as the subdomain imbalance could be improved upon anyways.

6.4 Partition Optimization Calibration

For the partition optimization process, both the balancing item and method were looked at to maximize the performance of the algorithm. A standard configuration included element balancing along with the equicut method and $\alpha = 0.95$. This setup worked well for a general mesh partition, but when more than one element type was introduced that had to be solved using the frontal method, alternative options had to be used. More specifically, degree of freedom balancing item and a wavefront balancing method had to be applied. The effectiveness of each configuration was judged by visual quality of the subdomains, interface node count, as well as load balance (LBF).

6.4.1 Balancing Item

The model used to calibrate the balancing item was a simple bridge containing a total of 2,680 elements (620 spar, 1,860 shell, and 200 solid) and 1,610 nodes. Calibration was carried out using 5 subdomains and an equicut balancing method. When the ‘degree of freedom’ option was applied initially, the generated interfaces were visibly broken. Thus, the penalty parameter in the fitness function (α) was varied in order to try and achieve the most balanced subdomains with the lowest unbroken interface count. The findings of this process are shown below in Table 9 with the best configuration highlighted. When looking at the data, the best overall compromise was evidently at $\alpha = 0.999$. A visual representation of the original ($\alpha = 0.95$) and best ($\alpha = 0.999$) configuration is depicted below in Figure 39.

Table 9 – Balancing Item Calibration Results

α	Interface Nodes	LBF	Broken Interface?
0.95	157	0.999	YES
0.99	139	0.996	YES
0.995	132	0.988	YES
0.999	126	0.976	NO
0.9999	128	0.927	NO

Figure 39 – Balancing Item Calibration Partitions

6.4.2 Balancing Method

A model of a panel with stiffeners [99], which contained a total of 4,752 shell elements and 4,795 nodes was used to calibrate the balancing method. The specific case used here had a partition of 4 subdomains, with the balancing item being elements since only one element type was present. Initially, when the wavefront balancing method was applied, the presence of a broken interface similar to the balancing item calibration was present. Once again, the fitness function penalty parameter ' α ' was varied from 0.95 to 0.9999. The findings of this calibration process are summarized in Table 10. Looking at the highlighted value, it should be evident that 0.995 was the best compromise between low interface node count and balanced subdomains,

while having a continuous subdomain interface. The standard configuration ($\alpha = 0.95$) and best ($\alpha = 0.995$) partition can be seen side by side in Figure 40.

Table 10 – Balancing Method Calibration Results

α	Interface Nodes	LBF	Broken Interface?
0.95	331	0.829	YES
0.99	247	0.899	YES
0.995	227	0.978	NO
0.999	203	0.789	NO
0.9999	204	0.117	NO

Figure 40 – Balancing Method Calibration Partitions

6.5 Governing Process Calibration

Since it was assumed that the partitioning algorithm would be applied to a wingbox optimization problem, a simple model composed purely of 4 node shell elements was developed in order to obtain calibration data. The model was perfectly rectangular and consisted of 2 spars, 10 ribs and 2 skin panels (lower and upper). Boundary conditions identical to the ones discussed in Section 5.3 were used except for the fact that a uniform lift/drag distribution was applied along a set of nodes on the upper surface. The material model too was identical to the wingbox optimization problem (Alloy 2024-T3). As for the real constants, a default value of 0.015 m was used for all thicknesses. It should be noted that the stress/displacement values were only desired for comparison to prove that both the partition and non-partition evaluations were getting the correct results, as was found to be the case. Figure 41, Figure 42, and Figure 43 show the model

dimensions, a transparent view, and a view of the model with applied boundary conditions, respectively.

Figure 41 – Calibration Wingbox Dimensions

Figure 42 – Transparent View of Calibration Wingbox

Figure 43 – Boundary Conditions Applied to Calibration Wingbox

The number of elements in the test model were varied and included 440, 1760, 3960, 7040, 11,000, 15,840, 21,560, 28,160, and 35,640 setups. For each of the nine configurations, standard FE evaluation time, direct partition time, and partitioned FE evaluation time for $q = 2, 3, 4, 5, 6, 7,$ and 8 were measured. It should be noted that the partitioned evaluation times were based on only the direct partition and not optimized partition. The thought process here was that these evaluation times represented minimum time savings since the partition optimization was expected to produce partitions with even less compute time required. Figure 44 and Figure 45 shown below describe the complete data that was attained in this calibration process for individual FE evaluation times and direct partitioning times, respectively. These runs were carried out on a 2.4 GHz Pentium 4 processor with 512 Mb of RAM.

Figure 44 – FE Evaluation Time Calibration Data

Figure 45 – Direct Partition Evaluation Time Calibration Data

From Figure 44, it should be observed that the lowest partitioned evaluation time after the non-partition/partition cross-over point was achieved with $q = 3$. This is the case from the cross-over point up to the largest model tested (35,640 elements). It is believed that the fluctuation of the non-partitioned FE evaluation curve is due to an issue related to ANSYS memory usage [100] and/or memory limitation due to the workstation setup (virtual memory). Furthermore, it is also believed that if partitioned FE evaluation trends were plotted at higher element counts, a similar fluctuation would be seen for each starting with $q = 2$. That is, the partitioned FE evaluations are not linear trends but polynomial just like the non-partitioned FE evaluation, except for the fact that the exponential region is delayed further with higher partition counts (q).

Second order polynomial trend functions were applied to the standard FE evaluation time, direct partition time, and partitioned FE evaluation time at $q = 3$ with the aid of Matlab. The resultant trends can be found in Figure 46 and Figure 47 below. Trend equations for the direct partition time, non-partitioned FE evaluation time, and partitioned FE evaluation time are marked as Equation 15, Equation 16, and Equation 17, respectively.

Figure 46 – FE Evaluation Time Trend Data

Figure 47 – Direct Partition Evaluation Time Trend Data

$$F_t(e) = 3.291(10^{-6})e^2 + 4.865(10^{-3})e - 7.387 \quad \text{Eq.15}$$

$$F_t(e) = 5.397(10^{-7})e^2 - 2.928(10^{-3})e + 8.004 \quad \text{Eq.16}$$

$$F_t(e) = 9.199(10^{-8})e^2 + 4.742(10^{-3})e + 3.786 \quad \text{Eq.17}$$

With this data acquired, the governing process depicted in Figure 26 (Section 4.3) could be implemented. The direct partition time, partitioned FE evaluation time, and non-partitioned FE evaluation time could all be calculated as a function of element count to give the desired time approximations.

7. Results

The following sections summarize the results achieved by applying the partitioning algorithm to several general examples as well as a wingbox optimization problem. A 2.4 GHz Pentium 4 workstation with 512 Mb of RAM was used to carry this work out.

7.1 Partitioning Algorithm – General Application

To describe the direct and optimization capability of the partitioning algorithm proposed in this work, a general setup was applied. This setup consisted of a dual graph, closest expansion element, an element balancing item, and equicut balancing method. Some of the models (2, 6, 8, 9) were chosen since they were used in pervious works ([3], [4], and [10]) even though actual partition data was available for only model 2 [3] and 6 [4]. It should be noted that black elements in the direct partition plots constitute expansion initialization elements.

A demonstration of the versatility of the direct partition module is provided via mesh 1, mesh 2, mesh 3, and mesh 4. Mesh 1 consists of a FE model made purely of 272 spar elements (LINK8 in ANSYS) with 81 nodes. Figure 48 shows the partitions achieved here.

Figure 48 – Mesh 1 Partition Results

Mesh 2 consists of a solid model made out of 1,500 solid elements (SOLID45 in ANSYS) and 2,076 nodes. Visual partitions are given in Figure 49, with the current work results summarized in Table 11. Since the data was available, results from Kaveh et al. [3] for this particular model are shown in Table 12. A 80486 DXII66 workstation was used by Kaveh et al. [3].

Figure 49 – Mesh 2 Partition Results

Table 11 – Mesh 2 Partition Results Summary for Current Work

q	Interface Nodes	Compute Time [sec]
2	258	35
5	290	47
8	476	54
10	519	55

Table 12 – Mesh 2 Partition Results Summary for Past Work

q	Interfac Nodes	Compute Time [sec]
2	239	233
5	298	236
8	565	336
10	618	344

Moving onto Mesh 3, this was constructed out of 894 3-node shell elements (SHELL63 in ANSYS) and contained 488 nodes. The mesh was not structured in this model to show the general applicability of the algorithm. A visual summary of the partitions is given in Figure 50 while Table 13 provides the numerical results.

Figure 50 – Mesh 3 Partition Results

Table 13 – Mesh 3 Partition Results Summary

q	Interface Nodes	LBF	Compute Time [sec]
2	32	0.985	4.719
3	51	0.881	5.047
4	47	0.858	5.297
5	56	0.756	5.468
6	71	0.618	5.687
7	86	0.567	5.563
8	92	0.754	5.296

The last direct partition example to show is that of mesh 4. This model had 25,600 4-node shell elements (SHELL63 in ANSYS) with 25,921 nodes. A summary of partitions is provided in Figure 51 and Table 14.

Figure 51 – Mesh 4 Partition Results

Table 14 – Mesh 4 Partition Results Summary

q	Interface Nodes	LBF	Compute Time [sec]
2	319	1.000	1791.578
3	320	0.709	1946.125
4	321	1.000	2103.796
5	519	0.789	2348.250
6	574	-0.442	2452.828
7	610	0.223	2636.875
8	641	0.164	2767.828

Moving on, meshes 5 – 9 were used to demonstrate the capability of the partition optimization module. Mesh 5 consisted of a simple 4-node shell element (SHELL63 in ANSYS) model that had 441 nodes. Figure 52 and Table 15 show visual and numerical results, respectively.

Mesh 6 used the same element as mesh 5, but in slightly more complex geometry. This model contained 460 elements with 517 nodes. Figure 53 and Table 16 show visual and numerical results, respectively. Since the data was available, partition results from Vanderstraeten et al. [4] are included in Table 17.

Mesh 7 used the same element as mesh 5, but incorporated geometry more complex than mesh 6. A total of 1,300 elements were used here with 1,326 nodes. The same model was used in the GA parameter calibration process described in Section 6.2. Figure 54 and Table 18 show visual and numerical results, respectively.

Mesh 8 used the same element as mesh 5, but incorporated geometry more complex than mesh 7. A total of 1,416 elements were used here with 1,649 nodes. Figure 55 and Table 19 show visual and numerical results, respectively.

Finally, mesh 9 used the same element as mesh 5 with an additional 3-node shell element. A total of 1,340 elements were used here with 1,042 nodes. Figure 56 and Table 20 show visual and numerical results, respectively.

Figure 52 – Mesh 5 Partition Results

Figure 53 – Mesh 6 Partition Results

Direct

Optimized

q = 2

q = 3

q = 4

q = 5

q = 6

q = 7

q = 8

Figure 54 – Mesh 7 Partition Results

Figure 55 – Mesh 8 Partition Results

Figure 56 – Mesh 9 Partition Results

Table 15 – Mesh 5 Partition Results Summary

q	Direct Partition			Optimized Partition				% Node Reduction	% LBF Increase
	Interface Nodes	LBF	Compute Time [sec]	Interface Nodes	LBF	Compute Time [sec]	Iterations		
2	39	0.995	1.69	36	1.000	52.64	4	7.69%	0.50%
3	40	0.750	1.59	38	0.990	93.05	7	5.00%	32.00%
4	41	1.000	1.73	41	1.000	56.20	4	0.00%	0.00%
5	62	0.938	1.79	61	0.988	75.03	5	1.61%	5.33%
6	69	0.765	1.84	68	0.950	203.98	13	1.45%	24.18%
7	79	0.805	1.89	78	0.968	281.16	17	1.27%	20.25%
8	86	0.820	1.92	81	0.920	311.45	18	5.81%	12.20%

Table 16 – Mesh 6 Partition Results Summary for Current Work

q	Direct Partition			Optimized Partition				% Node Reduction	% LBF Increase
	Interface Nodes	LBF	Compute Time [sec]	Interface Nodes	LBF	Compute Time [sec]	Iterations		
2	34	0.987	1.84	32	1.000	82.36	6	5.88%	1.32%
3	26	0.978	1.95	26	0.996	57.14	4	0.00%	1.84%
4	34	0.957	2.23	34	0.957	59.94	4	0.00%	0.00%
5	60	0.772	2.19	59	0.967	176.22	11	1.67%	25.26%
6	82	0.774	2.27	77	0.957	187.22	11	6.10%	23.64%
7	94	0.859	2.34	84	0.987	231.52	13	10.64%	14.90%
8	99	0.713	2.34	85	0.957	333.05	18	14.14%	34.22%

Table 17 – Mesh 6 Partition Results Summary for Past Work

q	Optimization Method	General Topology		1D Topology	
		% Interface Node Reduction	LBF	% Interface Node Reduction	LBF
2	Simulated Annealing	9	1.00	0	1.00
	Stochastic Evolution	8	1.00	9	1.00
	Tabu Search	9	0.99	9	1.00
4	Simulated Annealing	66	1.00	13	1.00
	Stochastic Evolution	26	0.98	11	0.99
	Tabu Search	61	0.99	2	0.98
8	Simulated Annealing	25	0.94	5	0.90
	Stochastic Evolution	10	0.94	0	0.91
	Tabu Search	26	0.94	6	0.94

Table 18 – Mesh 7 Partition Results Summary

q	Direct Partition			Optimized Partition				% Node Reduction	% LBF Increase
	Interface Nodes	LBF	Compute Time [sec]	Interface Nodes	LBF	Compute Time [sec]	Iterations		
2	26	1.000	9.05	26	1.000	140.41	4	0.00%	0.00%
3	76	0.978	10.02	76	0.998	164.20	5	0.00%	2.04%
4	131	0.788	10.17	144	1.000	392.34	12	-9.92%	26.90%
5	179	0.912	10.64	174	0.996	692.49	21	2.79%	9.21%
6	191	0.946	11.31	185	0.992	436.86	13	3.14%	4.86%
7	201	0.840	11.88	198	0.988	863.31	25	1.49%	17.62%
8	213	0.671	12.23	209	0.985	2142.05	60	1.88%	46.80%

Table 19 – Mesh 8 Partition Results Summary

q	Direct Partition			Optimized Partition				% Node Reduction	% LBF Increase
	Interface Nodes	LBF	Compute Time [sec]	Interface Nodes	LBF	Compute Time [sec]	Iterations		
2	24	0.941	8.52	28	1.000	230.940	5	-16.67%	6.27%
3	66	0.951	8.94	56	1.000	307.140	9	15.15%	5.15%
4	84	0.751	9.48	78	1.000	739.620	13	7.14%	33.16%
5	69	0.884	10.02	68	0.990	469.800	9	1.45%	11.99%
6	110	0.754	10.56	115	0.979	508.500	7	-4.55%	29.84%
7	128	0.779	11.04	128	0.996	624.600	11	0.00%	27.86%
8	139	0.689	11.46	124	0.983	594.000	7	10.79%	42.67%

Table 20 – Mesh 9 Partition Results Summary

q	Direct Partition			Optimized Partition				% Node Reduction	% LBF Increase
	Interface Nodes	LBF	Compute Time [sec]	Interface Nodes	LBF	Compute Time [sec]	Iterations		
2	40	1.000	8.940	40	1.000	77.100	4	0.00%	0.00%
3	31	0.754	8.100	36	0.990	530.400	10	-16.13%	31.30%
4	22	0.997	8.580	22	0.997	56.520	4	0.00%	0.00%
5	47	0.608	8.940	47	0.963	624.840	11	0.00%	58.39%
6	64	0.652	9.300	56	0.970	614.820	14	12.50%	48.77%
7	84	0.600	10.200	65	0.960	671.880	11	22.62%	60.00%
8	94	0.710	10.080	67	0.985	597.240	10	28.72%	38.73%

7.2 Wingbox Structural Optimization

The wingbox optimization problem discussed in Section 5 was carried out with a partitioning algorithm setup consisting of a dual graph, closest expansion element, an element balancing item, and wavefront balancing method. Figure 57 and Table 21 give results for the GA fitness plot over the 125 generations and the best solution, respectively. Table 22, Table 23, and Table 24 on the other hand give time summaries regarding the pre-run estimations, measured values, and validation results, respectively. The best fitness achieved in this optimization run was 298.3 kg with a maximum stress and displacement equal to 220.93 MPa and 0.056 m respectively. The direct partition (Figure 58) achieved an interface node count of 180 with LBF = -0.491. Partition optimization (Figure 59) increased node count to 225 but was able to dramatically increase LBF to 0.414. The final solution was validated via ANSYS to ensure that the minimum weight was calculated properly and that both stress and displacement constraints were not violated. Figure 60 and Figure 61 show the stress and displacement results respectively for the best solution.

Figure 57 – GA Fitness Plot

Table 21 – Best Solution of GA Wingbox Structural Optimization

Real Constant	Wing Structure	Thickness Value [m]
1	Rib	0.000254
2	Rib	0.002052
3	Rib	0.000404
4	Rib	0.000320
5	Rib	0.000912
6	Rib	0.000320
7	Rib	0.000287
8	Rib	0.000404
9	Rib	0.000404
10	Rib	0.000226
11	Rib	0.000254
12	Rib	0.002304
13	Rib	0.000361
14	Rib	0.000643
15	Rib	0.000287
16	Rib	0.000226
17	Rib	0.000203
18	Rib	0.000142
19	Rib	0.000813
20	Spar	0.006055
21	Spar	0.012506
22	Stringer	0.000427
23	Stringer	0.001770
24	Stringer	0.001915
25	Stringer	0.010658
26	Stringer	0.037426
27	Stringer	0.015078
28	Upper Skin	0.020886
29	Lower Skin	0.018503

Table 22 – Pre-Run Time Estimations

Phase	Time [hr]
Standard GA Evaluation	174.128
Partitioned GA Evaluation	119.447
Direct Partition	0.557
Partition Optimization	5.412
Total Partition Time	125.417
Time Savings	48.711

Table 23 – Measured Time

Phase	Time [hr]
Partitioned GA Evaluation	139.960
Direct Partition	0.493
Partition Optimization	5.452
Total Partition Time	147.335

Table 24 – Time Validation

Method	Single Evaluation Time [s]	Total Evaluation Time [hr]
Standard	297.703	208.392
Partitioned	199.343	139.540
Time Savings*	98.360	62.907

* Total evaluation time includes partition time costs

Figure 58 – Direct Partition Results for Wingbox Problem

Figure 59 – Partition Optimization Results for Wingbox Problem

Figure 60 – Wingbox Stress Constraint Validation

Figure 61 – Wingbox Displacement Constraint Validation

8. Discussion

This section will analyze the results shown in Section 7. First, Section 8.1 will focus on the outcome of the general application with respect to the direct partition module and partition optimization module. Section 8.2 will then go over results achieved in the wingbox optimization phase of this work.

8.1 Partition Algorithm – General Application

Starting with the direct partition results, mesh 1 and 2 were included in this work to showcase the algorithm's ability to handle more than one type of element. Figure 48 and Figure 49 demonstrated that the algorithm could partition spar as well as solid elements quite reasonably with no broken interfaces and subdomains that are intact. Since the data was available, a summary of interface node counts for $q = 2, 5, 8,$ and 10 are provided as attained by Kaveh et al. [3] in Table 12 for mesh 2. Results achieved in this work for the same application are shown in Table 11. Comparing the interface node counts, some items should be observed. For $q = 2$, Kaveh et al. [3] produced a slightly better partition with 239 nodes compared to 258 obtained currently. For the other three cases, better results were achieved with the current algorithm. At $q = 5$ the difference was only 8 nodes, but this gap was up to 101 nodes by $q = 10$. Considering that at $q = 10$, Kaveh et al. [3] achieved an interface node count of 619 nodes, this is a substantial improvement to say the least. Compute times for each case are also shown for both works. Realistically, not much can be drawn from these numbers since it is more a case of comparing apples to oranges due to the different platforms used to evaluate each set of runs.

Mesh 3 was included to showcase the direct partitioning algorithm's versatility with respect to unstructured meshes. In this case, 3-node shell elements were used in a simple 2D format. Looking at Figure 50, $q = 2, 3, 4,$ and 5 produced partitions with respectable quality, i.e. the subdomains were for the most part intact and no broken interfaces were witnessed. For $q = 6, 7,$ and 8 the results are not as positive. In these three cases, some of the subdomains contained thin geometry that protruded between or into adjacent subdomains. The most probable cause for the result is the coarse nature of the mesh used here. As the number of subdomains increase in

smaller mesh sizes, picking initialization elements with equal amounts of unmarked neighbours around them gets more and more difficult. When this happens, it leads some subdomains with smaller element counts during the expansion process to protrude into their neighbours in an effort to balance the subdomain loads (equicut balancing). The numerical results for this partition are shown in Table 13, which includes node interface counts, LBF, and compute time. Minimal and maximal node interface counts were found at $q = 2$ and $q = 8$ at 32 and 92 respectively. LBF on the other hand peaked out at $q = 2$ (LBF = 0.985) and bottomed out at $q = 7$ (LBF = 0.567). Considering the coarseness of the mesh, the minimum LBF was respectable for 7 subdomains. As for the compute time of each run, all were between 4 and 6 seconds due to the simplicity of the FE mesh.

Mesh 4 shown in Figure 51 contained the same 2-D geometry as Mesh 3. Instead of using 894 3-node shell elements though, 25,600 4-node shell elements were applied. From Figure 50, it is clear that the direct partition algorithm can achieve symmetric and defined interfaces in higher fidelity models. For all cases ($q = 2$ to $q = 8$), it should be observed that all generated partitions contained a line of symmetry. Node interface counts, LBF values, and compute times are summarized in Table 14 for this example. The smallest interface was observed to be 319 at $q = 2$. At the other end of the spectrum, $q = 8$ had 641 interface nodes. As far as load balance was concerned, $q = 2$ and $q = 4$ achieved a perfect partition, i.e. LBF = 1.0. This is obvious when looking at Figure 51 since the generated subdomains for the two cases are visibly of equal size. On the other hand, a minimum value of -0.442 at $q = 6$ was observed. This poor balance is most likely attributed to initialization element selection. Since at $q = 5$, the initialization elements are completely symmetric, any new addition results in a compromised situation. Even though a large number of unmarked elements exist around each initialization element, some have greater distances between one another. Thus, during expansion, some subdomains may engulf a particular neighbour and prevent it from expanding further. This was most likely the case for the middle subdomain on the right side of the mesh. Combine this with a large expansion space available for the center subdomain towards the left side of the mesh, the potential for an imbalance results. Though this condition is not favourable, the fact that the partition optimization module can balance the subdomains makes this a less critical issue. The lesson to be gained from this experience is that certain combinations of q and mesh geometry lead to more

favourable partitions. Even though it is not shown, it would be expected that at $q = 9$, a very balanced, if not perfectly balanced, condition would arise. The last item to point out for mesh 4 was that of the compute time required to perform these direct partitions. Since a higher element count was used here, this aspect ranged from 1791.578 seconds at $q = 2$ to 2767.828 seconds at $q = 8$. As expected, the higher the subdomain count, the more time required for partitioning.

Mesh 5 was essentially a simplified version of mesh 4, which contained only 400 4-node shell elements. This example included a direct partition phase, which was followed by a partition optimization phase for $q = 2$ to $q = 8$. Plots of direct and optimized partitions are provided for $q = 2$ to $q = 8$ in Figure 52. It should be observed that the optimized plots contain interfaces that are straighter than their direct counterparts due to the node reductions achieved. These reductions are numerically summarized in Table 15. Depending on the case, the partition optimization algorithm was able to achieve interface node reductions between 0 % and 7.69 %. No cases of node interface increase were witness for mesh 5. As expected, since $q = 4$ achieved a perfectly balanced partition, no improvements were found in both interface and balance. LBF on the other hand was increased anywhere from 0 % at $q = 4$ to 32 % at $q = 3$. After optimization, the lowest LBF value was a respectable 0.92 at $q = 8$. If a model with high fidelity was used, LBF for $q = 8$ would be expected to be higher since balancing small meshes gets increasingly difficult the greater the number of partitions. Looking at compute time, all direct partitions were under 2 seconds due to the small model size. Partition optimization time on the other hand was far greater and varied depending on the number of optimization iterations. A total of 18 iterations were conducted for $q = 8$, which corresponded to a total partition optimization time of 311.45 seconds.

Mesh 6 was included in this work since the same model was used by Vanderstraeten et al. [4]. This model was composed of 460 4-node shell elements arranged in a 2D format. Direct and optimized partition plots are provided in Figure 53. Here, one can see that the optimization phase straightened out some of the interfaces, but only slightly. Generally speaking, there is not a visually large difference between the direct and optimized results. To find more information, one must refer to Table 16 where the partition results are summarized in detail. Here it can be seen that the node interface counts varied from 26 ($q = 3$) to 99 ($q = 8$) for the direct partition

phase. Node interface count reductions ranged from 0 % up to 14.14%. LBF on the other hand was lowest for $q = 8$ during the direct partitioning process. Optimization increased LBF from 0 % to 34.22 % between all the cases. No improvements were witnessed for $q = 4$ while the largest improvements were attained at $q = 8$. Compute time for direct partitioning and partition optimization behaved similar to what was seen for mesh 5, probably since roughly the same number of elements was used with the same number of partitions. Direct partitions took at most 2.34 seconds with the partition optimization process ranging from 82.36 to 333.05 seconds. Again, this depends on the number of optimization iterations performed. Looking at the results achieved by Vanderstraeten et al. [4] in Table 17, only limited comparisons can be made due to the limited amount of information. As far as percent interface node count reductions are concerned, Vanderstraeten et al. attained far greater improvements than the current work. These results must be interpreted with caution though, since the actual interface node counts are not known. It can easily be the case that the direct partitioning algorithm in this work achieved lower interface counts, resulting in smaller improvements during the optimization stage. That said, it is difficult to compare this aspect of the partitions. LBF on the other hand is another story. For $q = 2$, both results matched with LBF equal to 1.000. Vanderstraeten et al. attained better balance for $q = 4$ at 1.000 compared to the current work's 0.957 value. Lastly, for $q = 8$ the current algorithm produce a better balance at LBF = 0.957 compared to Vanderstraeten et al. (0.94). Looking at these numbers, it can be observed that LBF was similar between both methods, thus suggesting that the genetic algorithm applied in this work was at least as capable as simulated annealing, stochastic evolution, and tabu search methods applied in Vanderstraeten et al. [4].

Mesh 7 was made slightly more complex than the previous examples by incorporating 3-D geometry. In simplest terms, the model used here was a rectangular box with two ends missing that was composed purely of 1,300 4-node shell elements. A plot of the direct and optimized partitions can be found in Figure 54. As was the case for mesh 5 and 6, the optimized subdomains feature interfaces that are straighter when compared to the direct partition outcomes. This is especially true for higher subdomain counts. As for the quantitative results, these are summarized in Table 18. Looking at the interface node count improvements, no progress was achieved in this department for $q = 2$ and $q = 3$. As is rarely the case, a 9.92 % interface node

increase was witnessed for $q = 4$. Remaining runs had improvements ranging from 1.49 % to 3.14 %. Looking at LBF now, $q = 2$ and $q = 3$ had little improvement, suggesting that the direct partitions were close to optimal. In Figure 54, one can observe that this was visually the case. Even though an increase in interface node count was present for $q = 4$, a considerable increase of 26% was achieved for LBF. The other runs managed improvements ranging from 4.86 % ($q = 6$) to 46.80 % ($q = 8$). Considering that the lowest final LBF value achieved here was 0.985, it should be noted that the partitioning algorithm functioned quite respectively for this particular application. As far as compute time was concerned, direct partitioning varied from 9.05 seconds at $q = 2$ to 12.23 seconds at $q = 8$. Partition optimization time depended on the number of optimization iterations performed as in the other cases. For $q = 2$, only 4 iterations were conducted which resulted in only 140.41 seconds of run time. On the other hand, $q = 8$ took 60 iterations to converge, producing a total optimization time of 2142.05 seconds.

Mesh 8 was included in this work due to the fact that it was also used by Kaveh et al. [3]. Even though no quantifiable results were given in that work, it gave an opportunity to test the direct and optimization partition algorithms on more complex 2-D geometry that contained 1,416 4-node shell elements. A visual summary of the results achieved here can be found in Figure 55 while numerical results are given in Table 19. Going straight to Table 19, one can observe that respectable results were obtained with the direct partition module. From $q = 2$ to $q = 8$, interface node count varied from 24 to 139 while LBF ranged from 0.951 to 0.689. For $q = 2$ and $q = 6$, the interface node counts actually increased during the optimization phase. Both cases however, achieved increases in LBF. This result is understandable in situations where you have a compromise between an interface node reduction and load balance improvement. For $q = 7$, no improvements for interface node count were observed, but significant LBF improvements were once again witnessed (27.86%). All other cases saw improvements in both aspects, with $q = 8$ attaining the biggest benefits (10.79% and 42.67% for node interface count and LBF improvements respectively). The optimization process significantly improved LBF since the lowest final value was 0.979. As far as compute time goes, direct partitions ranged from 8.52 to 11.46 seconds since this particular model was not that large. The longest optimization time was for $q = 4$, which corresponded to the highest number of optimization iterations (13).

The final mesh to discuss as far as the general application of the partitioning algorithm is concerned was that of mesh 9. This was another 2-D model, which was composed of 1,340 3-node and 4-node shell elements. Visual representations of all direct and optimized partitions can be observed in Figure 56. As was the case in mesh 4, the symmetric nature of the geometry and higher number of elements within the model resulted in certain partitions having symmetric patterns. In particular, $q = 2$, $q = 4$, $q = 6$, and $q = 8$ embodied these characteristics. Looking at the quantitative results in Table 20, the direct partition phase carried out its duties quite reasonably. Interface node counts amongst all cases ranged from 40 to 94 with the highest and lowest LBF observed to be 1.000 and 0.600 respectively. The highest LBF case existed for $q = 2$ and saw no improvements during the optimization phase. A similar situation was present for $q = 4$ with $\text{LBF} = 0.963$. All other cases witnessed improvements in both aspects other than $q = 3$, which traded an interface node increase for an improvement in load balance. This was expected since the nature of the mesh did not allow for a straight forward partition of the model into three equal parts. Final LBF values after optimization were 0.960 or higher, which was very positive. Compute time for the direct partition phase was in the area of previous models having similar element counts, ranging from 8.94 seconds ($q = 2$) to 10.08 seconds ($q = 8$). Minimum optimization time was witnessed for $q = 4$ at 56.52 seconds since no improvements could be found here, resulting in the algorithm stalling out fairly quickly. On the other end of the spectrum, $q = 5$ ran for 624.84 second and carried out a total of 11 optimization iterations.

8.2 Wingbox Structural Optimization

Looking at Figure 57 above, one can see the progression of the fitness function over the 125 generations carried out in the wingbox structural optimization. During the entire run, the fitness steadily decreased from 432.6 kg to 298.3 kg by the end. Looking at this final value, one might say that it is quite reasonable, but when comparing it to the empty aircraft weight of 3493 kg [80], it contributes up to 17% of the total aircraft weight (both wings). Given all the other structural parts of the aircraft, landing gear, engines, and interior, it seems as though this weight might be slightly high. Further inspection into Table 21 sheds more light on this issue. Since the ribs were constrained to standard sheet metal sizes, none of these were exceptionally oversized. This was not the case with the spars, stringers and skin members as they had a continuous input

range from 0.000 m – 0.050 m. The spars (real constants 20 and 21) were expected to have thicker values as they had to bare much of the vertical loading. Stringers and skins, however were not expected to be very thick, which was the case for real constants (25 – 29). It should be obvious that a stringer 0.0374 m thick and skin members at 0.0209 m thick are not realistic. Two main explanations exist for this result. One is due to the premature ending of the GA run, i.e. the optimal solution was not reached in this case. This does not mean that there is something wrong with the algorithm, but merely that it must be run longer. The fact that the fitness function was still decreasing towards the end of the run supports this notion. Proper calibration of the GA parameters for a structural optimization could also have had an effect. In this case, parameters attained from the partition optimization calibration process were used, which may have not given the best convergence performance here. The nature of the structural optimization problem was different from the partition optimization problem in that fewer variables were used, but a far larger amount of inputs were possible. Another aspect that could have played a role in achieving this sort of result could have been due to the way in which the FE model was constrained, i.e. by all degrees of freedom at the root. Loading a cantilever structure that is constrained this way causes very high stress to occur at the corners of the root, as should be evident by the mesh resolution required for this analysis. In reality, an aircraft wing under ultimate loading is not attached to a rigid airframe. Thus, it may have been more realistic to model a section of the fuselage in order to avoid this kind of artificially high stress concentration. This may explain why the skin members were found to be so thick. In the end, it is quite possible that both of the above aspects led to the attained result. As far as the optimization constraints go, the model was most definitely limited by stress and not displacement. With a maximum allowable stress of 230 MPa, a value of 221 MPa gave only 4% margin. Displacement on the other hand was nowhere near the 0.600 m limit at 0.056 m. This once again points to the FE constraints giving unrealistically high stresses at the corners of the wing root.

Looking first at the direct partition, a node interface count of 180 and LBF of -0.491 was achieved. After the partition optimization portion was carried out, the interface node count increased to 225, but this was accompanied with a significant increase of LBF to 0.414 and slightly broken interface quality, as shown in Figure 59.

It should be noted that the partition optimization algorithm was ended by time limitation and it did not finish its first optimization iteration. What must be understood is the optimization process is carried out by first balancing the subdomains, which is usually accompanied by an increase in the interface node count. It is not until the latter stages of the optimization process, once the subdomains are balanced as much as possible, that the node count is decreased and interface quality is improved. The increase in interface node count and poor interface quality is thus a result of the premature termination. For models that have to run structural optimization for longer periods of time, this time limit is less of an issue and better partitions are achieved by the optimization module. As for the considerably low initial LBF of -0.491, this was most likely due to the highly refined nature of the FE wingbox root elements. The high concentration of elements at the root led the direct method to choose a third (middle) partition expansion initialization element too close to the wing tip, which prevented uniform expansion of the second subdomain (wing tip) and caused the imbalance.

Even though the result of the structural optimization problem was not completely favorable, the main reason for carrying it out was to demonstrate the potential time savings of the partitioning algorithm when applied to a single processor. To start off, the time approximations based on the simplified wingbox calibration model are summarized in Table 22. These numbers show a total GA fitness evaluation time of 174.128 hours. At the same time, a partitioned setup yields a GA fitness evaluation time of 119.447 hours. Taking the partition costs into account, this partitioned setup time increases to 125.417 hours, still giving estimated time savings of 48.711 hours which is quite considerable. The measured time of each segment in the partitioned wingbox structural optimization problem is shown in Table 23. One can see here that the measured total GA fitness evaluation time for a partitioned setup was 139.960 hours, considerably higher than what was expected (119.447 hours). The direct partition time and partition optimization time though, were very similar to the approximated values. Total measured solution time came out to be 147.335 hours. The difference in partitioned evaluation time can be explained when looking at Table 24. Here, the time required to evaluate one standard and partitioned model is shown. These results are based on the best run out of 5 as some variations due to memory allocation were witnessed. Also shown in Table 24 are the calculated total GA evaluation time given the single evaluation time is known along with the number of GA individuals and generations carried out. It must be

pointed out that the calculated total GA fitness evaluation time for a partitioned setup from Table 24 is almost identical to the measured value in Table 23, which gives confidence that the former was correct. This leads one to believe that the approximated values in Table 22 were not approximating the correct time, and the actual standard GA fitness evaluation time was 208.392 hours instead of 174.128 hours. When this situation was investigated even further, it could be seen that the ratio of partitioned/standard GA fitness evaluation from Table 22 (0.686) was very similar to the value in Table 24 (0.670). This suggested that the approximated values were not necessarily incorrect, but off by a scaling factor when compared to the measured times since the ratios were nearly the same. The most likely reason behind this was that the compute load was not only a function of the number of elements, but also geometric aspects of the FE model. Since the actual model had stringers and far more ribs than the calibration wingbox model, geometric complexity increased the wavefront, which in turn increased compute time. With that said, the approximated results can still be used in the governing process since proper comparison can be made between the standard and partitioned models. Furthermore, the total standard evaluation time was 208.392 hours. When partition costs are taken into account, the 147.335 hour total partitioned setup time still achieved time savings of 62.907 hours, which is quite considerable at 30.2%.

9. Conclusion

This work outlined a new two-step partitioning algorithm based on Kaveh et al. [3] and Vanderstraeten et al. [4] via genetic algorithm. The genetic algorithm applied in this work was validated while at the same time important partitioning parameters were calibrated to achieve the best possible results. The problems used to assess the proposed partition algorithm were grouped into two main headings, that of general application and wingbox structural optimization. The former was decomposed even further into direct partitioning and partition optimization. Concluding remarks regarding these three aspects can be viewed below.

The direct partition algorithm, which was based on a 'separateness' concept by Kaveh et al. [3] demonstrated several key characteristics in this work. One was that the algorithm was very versatile when it came to working with multiple element types. Even though great effort was not put into this area of the algorithm, it clearly showed that 3 dimensional meshes containing spar and solid elements could be partitioned with relative ease, so long as the proper conditional statements were included within the algorithm. For mesh 2, which was purely composed of solid elements, the current algorithm was able to achieve better results for 3 out of the 4 runs when compared to Kaveh et al. [3]. Moving on, the direct partition method proposed here demonstrated its ability to handle meshes that are not structured. This aspect is important since many application in the real world do have unstructured elements due to complex geometry, or have high degrees of mesh variation as a result of refined local regions. Finally, the algorithm showed that for meshes with higher numbers of elements, structured and symmetric subdomains could be generated with clean and uniform interfaces if the geometry itself was symmetric. All this was achieved with respectable load balances for the most part, albeit some specific combinations of geometry, mesh size, and subdomain counts were seen to achieve unfavorable outcomes. Even so, cases such as this were more of an anomaly and did not constitute the majority of cases examined. That said, the generated direct partition algorithm was considered a success due partially to its performance in partitioning, but more so for its durable nature with respect to general applicability.

9. Conclusion

This work outlined a new two-step partitioning algorithm based on Kaveh et al. [3] and Vanderstraeten et al. [4] via genetic algorithm. The genetic algorithm applied in this work was validated while at the same time important partitioning parameters were calibrated to achieve the best possible results. The problems used to assess the proposed partition algorithm were grouped into two main headings, that of general application and wingbox structural optimization. The former was decomposed even further into direct partitioning and partition optimization. Concluding remarks regarding these three aspects can be viewed below.

The direct partition algorithm, which was based on a ‘separateness’ concept by Kaveh et al. [3] demonstrated several key characteristics in this work. One was that the algorithm was very versatile when it came to working with multiple element types. Even though great effort was not put into this area of the algorithm, it clearly showed that 3 dimensional meshes containing spar and solid elements could be partitioned with relative ease, so long as the proper conditional statements were included within the algorithm. For mesh 2, which was purely composed of solid elements, the current algorithm was able to achieve better results for 3 out of the 4 runs when compared to Kaveh et al. [3]. Moving on, the direct partition method proposed here demonstrated its ability to handle meshes that are not structured. This aspect is important since many application in the real world do have unstructured elements due to complex geometry, or have high degrees of mesh variation as a result of refined local regions. Finally, the algorithm showed that for meshes with higher numbers of elements, structured and symmetric subdomains could be generated with clean and uniform interfaces if the geometry itself was symmetric. All this was achieved with respectable load balances for the most part, albeit some specific combinations of geometry, mesh size, and subdomain counts were seen to achieve unfavorable outcomes. Even so, cases such as this were more of an anomaly and did not constitute the majority of cases examined. That said, the generated direct partition algorithm was considered a success due partially to its performance in partitioning, but more so for its durable nature with respect to general applicability.

As for the partition optimization method, promising and positive results were found. Out of a total 35 applications using 5 different shell models, only 4 cases were present that saw an increase in interface node count after the optimization phase was complete. These numbers must be placed in context though, as each of these cases saw an increase in the LBF value, suggesting that the partition optimization process found a better compromise between interface node count and balance. Another 9 runs saw no improvement in interface node count, 5 of which were a result of optimal conditions achieved by the direct partition module (the only cases that saw no improvement in LBF). The other 4 had balance improvements coupled with no change in interface node count. Putting the above information into perspective, it must be pointed out that 22 out of the 35 runs (63%) achieved benefits in both interface node count and load balance. This outcome justifies the use of the partition optimization algorithm and considers it a success in terms of general application. A comparison with Vanderstraeten et al. [4] for mesh 6 saw similar load balance results after optimization, suggesting that a genetic algorithm for the optimization phase was at least as effective as simulated annealing, stochastic evolution, or tabu search.

The final and most important remarks are left for the wingbox optimization application, which used a wavefront balancing method unlike all previous problems evaluated in this work. With a final weight of 298.3 kg, the results of the actual GA structural optimization process were not as positive as was expected since some of the member, particularly the upper and lower skin, were still too thick at the end (over 0.018 m). As discussed in Section 8.2, this outcome was very likely due to a combination of GA parameter setup, premature termination, and unrealistic boundary conditions. None of the above issues can be considered all that extensive, that is, achieving better optimization results for this problem setup should not be that difficult given that the proper time is taken to ensure that refinements are incorporated and the problem is re-evaluated. It is not believed that this issue is that critical since the main reason for carrying out this application in the first place was to demonstrate the time saving potential of the partition algorithm proposed in this work on a single processor platform. The two step partitioning method demonstrated that a direct partition could be found with relative ease and be significantly improved upon with respect to LBF (from -0.491 to 0.414) even if a time limit was enforced in this phase. When taking the 6 hours required for the total partitioning process into account, time

savings of 30 % over a standard evaluation time of 208 hours can be considered substantial to say the least. Based on this outcome, application of the partitioning algorithm proposed in this work is easily justified for lengthy optimization problems that are limited to a single processor. On that note, this two-step algorithm can be considered a success.

10. Suggested Future Work

An overview of possible future studies related to the two-step partitioning algorithm proposed in this work is outlined below.

- Since the direct partition method shown here was created with general application in mind, the next step would be to incorporate conditional statements that would avoid unfavorable conditions in specific applications. Modifications to the subdomain initialization or the subdomain expansion process may be carried out to investigate this aspect of the direct partitioning algorithm.
- In the current work, an assumption was made that the wingbox optimization problem was to be evaluated with a frontal solver and a balancing criteria was designed to meet this requirement. Similarly, an assumption could be made that another FE solver (e.g. bandwidth based) is to be used and a balancing criteria can be developed to suit this type of application.
- The algorithm can be modified to achieve time savings for optimization problems requiring FE evaluations that are non-static in nature (i.e. modal, harmonic, transient, contact, etc.). If non-linear problems were to be introduced, another method of evaluating the model other than ANSYS would have to be found since superelements (subdomains) in this software do not support non-linear analysis [101].
- Another possible application the partition algorithm could be applied to in the future is a situation where optimization of each subdomain is required instead of the entire model. This sort of work can be done in a series or parallel architecture, depending on the problem requirements or processor availability.

References

- [1] Krog, L., Tucker, A., Kemp, M., & Boyd, R., "Topology Optimization of Aircraft Wing Box Ribs", *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, pg.1-11, 2004.
- [2] Schuhmacher, G., Murra, I., Wang, L., Laxander, A., O'Leary, O.J., & Herold, M., "Multidisciplinary Design Optimization of a Regional Aircraft Wing Box", *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pg.1-10, 2002.
- [3] Kaveh, A., & Roosta, G.R., "An algorithm for partitioning of finite element meshes", *Advances in Engineering Software*, 30, pg.857-865, 1999.
- [4] Vanderstraeten, D., & Keunings, R., "Optimized Partitioning of Unstructured Finite Element Meshes", *International Journal for Numerical Methods in Engineering*, 38, pg.433-450, 1996.
- [5] Farhat, C., "A Simple and Efficient Automatic FEM Domain Decomposer", *Computer and Structures*, 28, pg.579-602, 1988.
- [6] Al-Nasra, & M., Nguyen, D.T., "An Algorithm for Domain Decomposition in Finite Element Analysis", *Computers and Structures*, 34(3), pg.277-289, 1991.
- [7] Malone, J.G., "Automated Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessor Computers", *Computer Methods in Applied Mechanics and Engineering*, 70, pg.27-58, 1988.
- [8] Farhat, C., & Lesoinne, M., "Automatic partitioning of unstructured meshes for parallel solution of problems in computational mechanics", *International Journal of Numerical Methods in Engineering*, 36, pg.745-764, 1993.
- [9] Simon, H.D., "Partitioning of Unstructured Problems for Parallel Processing", *Computing Systems in Engineering*, 2(2), pg.135-148, 1991.
- [10] Kaveh, A., & Roosta, G.R., "Domain Decomposition for Finite Element Analysis", *Communications in Numerical Methods in Engineering*, 13, pg.61-71, 1997.
- [11] Kaveh, A., & Roosta, G.R., "An algorithm for partitioning of finite element meshes", *Advances in Engineering Software*, 30, pg.857-865, 1999.
- [12] Farhat, C., "A Lagrange Multiplier Based Divide and Conquer Finite Element Algorithm", *Computing Systems in Engineering*, 2, pg.149-156, 1991.
- [13] Pothén, A., Simon, H.D., & Liu, K.P., "Partitioning Sparse Matrices with Eigenvectors of Graphs", NAS Systems Division, NASA, RNR-89-009, pg.1-30, 1989.

- [14] Kaveh, A., Bahreninejad, A., & Mostafaei, H., "A hybrid graph-neural method for domain decomposition", *Computers and Structures*, 70, pg.667-674, 1999.
- [15] Kaveh, A., & Rahimi Bondarabady, H.A., "A hybrid graph-genetic method for domain decomposition", *Finite Elements in Analysis and Design*, 39, pg.1237-1247, 2003.
- [16] Bouhmala, N., Knutti, P. A., & Naegeli, H. H., "A Distributable Algorithm for Optimizing Mesh Partitions", *Proceedings of the 1996 International Conference on Parallel and Distributable Systems*, pg.544-549, 1996.
- [17] ANSYS, Inc., "15.9.3 Frontal Solvers", Theory Reference, Release 11.0 Documentation for ANSYS, 2007.
- [18] Rao, S.S., "Optimization Engineering – Theory and Practice (Third Edition)", New York: Wiley & Sons Inc., 1996, pg. 376-408, 715, 808-810
- [19] Jones, G., "Genetic and Evolutionary Algorithms", in Encyclopedia of Computational Chemistry, Chichester: John Wiley & Sons, 1998, pg.1-3
- [20] Chipperfield, A., Fleming, P., Pohlheim, H., & Fonseca, C., "Genetic Algorithm Toolbox, Version 1.2, User's Guide", England: University of Sheffield, 1994
- [21] Kamat, M.P., "Structural Optimization: Status and Promise", United States of America: American Institute of Aeronautics and Astronautics, 1993, pg.614
- [22] Ivask, E., "Genetic Algorithms in Test Pattern Generation", Estonia: Tallin Technical University, 1998, pg.13
- [23] Chakraborty, B., & Chaudhuri, P., "On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis" *Austrian Journal of Statistics*, 23(1,2), pg.13-27, 2003.
- [24] Holland, J.H., "Adaptation in Natural and Artificial Systems", Ann Arbor: University of Michigan Press, 1975.
- [25] Rao, S.S., "Optimization Engineering – Theory and Practice (Third Edition)", New York: Wiley & Sons Inc., 1996, pg. 808-810
- [26] Koza, J.R., "Genetic Programming: On the programming of Computers by Means of Natural Evolution", MIT Press, 1992.
- [27] Streichert, F., "Introduction to Evolutionary Algorithms", *Frankfurt MathFinance Workshop*, pg.11, 2002.

- [28] Abonyi, J., & Madar, J., "Evolutionary Algorithms", Hungary: University of Pannonia, 2005, pg.7,12
- [29] Rechenberg, I., "Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution", Frommann-Holzboog, Stuttgart, 1973.
- [30] Schwefel, H.P., "Numerical Optimization of Computer Models", Chichester: John Wiley & Sons, 1981.
- [31] Fogel, L.J., Owens A.J., & Walsh, M.J., "Artificial Intelligence through Simulated Evolution", New York, John Wiley & Sons, 1966.
- [32] Buseti, F., "Simulated Annealing Overview", 2003, pg.1-4
- [33] Eberhart, R.C., Hu, X ., & Shi, Y., "Engineering Optimization with Particle Swarm", *Proceedings of the IEEE Swarm Intelligence Symposium 2003*, Indianapolis, USA, 2003, pg.1
- [34] Cushman, D.L., "A Particle Swarm Approach to Constrained Optimization Informed by "Global Worst"", Pennsylvania: Pennsylvania State, 2007, pg.1-3
- [35] Saab, Y., & Rao, V., "Stochastic evolution: A fast effective heuristic for some generic layout problems" *27th ACM/IEEE Design Automation Conference*, pg.26-31, 1990.
- [36] Thamizhmani, N, "An Empirical Study of the Stochastic Evolution Algorithm for the VLSI Cell Placement Problem", MASc Thesis Paper, Montana State University, Montana, pg.6
- [37] Saab, Y., & Rao, V., "Combinatorial Optimization by Stochastic Evolution", *IEEE Transcript on Computer-Aided Design*, 10(4), pg.525-535, 1991.
- [39] Glover, F., "Tabu Search–Part I", *ORSA Journal on Computing*, 1(3), pg.190-206, 1989.
- [40] Glover, F., "Tabu Search–Part II", *ORSA Journal on Computing*, 1(3), pg.4-32, 1989.
- [41] McCulloch, W.S., & Pitts, W.H., "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, 5, pg.115–133, 1943.
- [42] The Math Works, Inc., "Neural Network Toolbox User's Guide", United States, 2009, pg.1-2
- [43] Abraham, A., "Artificial Neural Networks", *Handbook of Measuring System Design*, 0-470-02143-8, pg.901-908, 2005.
- [44] Kaveh, A., "Graphs and Structures", *Computers & Structures*, 40(4), pg.893-901, 1991.

- [45] Malone, J.G., "Automated Mesh Decomposition and Concurrent Finite Element Analysis for Hypercube Multiprocessor Computers", *Computer Methods in Applied Mechanics and Engineering*, 70, pg.27-58, 1988.
- [46] Everstine, G.C., "The BANDIT computer program for the reduction of matrix bandwidth for NASTRAN", Tech. Rept. 3872, Naval Ship Research and Development Center, Washington, DC, 1972.
- [47] Gibbs, N.E., Poole W.G., & Stockmeyer, P.K., "A comparison of several bandwidth and profile reduction algorithms", *ACM Transactions on Mathematical Software*, 2(4), pg.322-330, 1976.
- [48] Farhat, C., Lesoinne, M., & Pierson, K., "A scalable dual-primal domain decomposition method", *Numerical Linear Algebra with Application*, (7), pg.687-714, 2000.
- [49] Farhat, C., & Roux, F.X. "A method of finite element tearing and interconnecting and its parallel solution algorithm", *International Journal for Numerical Methods in Engineering*, 32, pg.1205-1227, 1991.
- [50] Farhat, C., Mandel, J., & Roux, F.X., "Optimal convergence properties of the FETI domain decomposition method", *Computational Methods Applied to Mechanical Engineering*, 115, pg.367-388, 1994.
- [51] Farhat, C., & Roux, F.X., "Implicit parallel processing in structural mechanics", *Compt. Mech. Adv.*, 1, pg.121-220, 1994.
- [52] Farhat, C., & Rixen, D.J., "A Simple and Efficient Expansion of a Class of Substructure Based Preconditions to Heterogeneous Structural Mechanics Problems", *Int. J. Meth. Eng.*, 44, pg.489-516, 1999.
- [53] Farhat, C., Bavestrello, H., & Avery, P., "Incorporation of Linear Multipoint Constraints in Domain-Decomposition-Based Iterative Solvers – Part II: Blending FETI-DP and Mortar Methods and Assembling Floating Substructures", *Comput. Methods Appl. Mech. Engrg*; 196, pg.1347-1368, 2007.
- [54] LeTallec, P., Manel, J., & Vidrascu, M., "A Neumann-Neumann domain decomposition algorithm for solving plate and shell problems", *SIAM J. Numer. Anal.*, 35, pg.836-867, 1998.
- [55] Farhat, C., & Mandel, J., "The two-level FETI method for static and dynamic plate problems – Part I: an optimal iterative solver for biharmonic systems", *Compt. Meth. Appl. Eng.*, 155, pg.129-152, 1998.

- [56] Farhat, C., Chen, P.S., Mandel, J., & Roux, F.X., "The two-level FETI method – Part II: extension to shell problems, parallel implementation and performance results", *Compt. Meth. Appl. Eng.*, 155, 1998, 153-118
- [57] Kaveh, A., "Spectral trisection of finite element models", *Int'l. Jour. Num. Meth. For Heat & Flu. Flow*, 11(4), pg.358-370, 2001.
- [58] Farhat, C., & Dureisseix, D., "A numerically scalable domain decomposition method for the solution of frictionless contact problems", *Int. J. Num. Meth. Eng.*, 50, pg.2643- 2666, 2001.
- [59] Kaveh, A., Bahreininejad, A., & Mostafaei, H., "A hybrid-graph-neural method for domain decomposition", *Comp. & Struct.*, 70, pg.667-674, 1999.
- [60] Vanderstraeten, D., Farhat, C., Chen, P.S., Keunings, R., & Ozone, O., "A retrofit based methodology of large-scale mesh partitions: beyond the minimum interface size criterion", *Comp. Meth. Appl. Mech. Eng.*, 133, pg.25-45, 1996.
- [61] Farhat, C., Lanteri, S., & Simon, H.D., "TOP/DOMDEC – A Software Tool for Mesh Partitioning and Parallel Processing", *Comp. Syst. In Eng.*, 6(1), pg.13-26, 1995.
- [62] Johnson, D.S., Aragon, C.R., McGeoch, L.A., & Schevon, C., "Optimization by simulated annealing: An experimental evolution; Part I, Graph Partitioning", *Operations Reaserach*, 37(6), pg.865-892, 1989.
- [63] Ghosal, S., Mandel, J., and & Tezaur, R., "Automatic Substructuring for Domain Decomposition Using Neural Networks", *Proceedings of IEEE International Conference on Neural Networks, Orlando, June, 1994*, 6, pg.3816–3821, 1994.
- [64] Walshaw, C., Cross, M., Diekmann, R., & Schlimbach, F., "Multilevel Mesh Partitioning For Optimizing Domain Shape", *The International Journal of High Performance Computing Applications*, 13(4), 1999, pg.334-353
- [65] Kaveh, A., & Shojaee, S., "Optimal domain decomposition via p-median methodology using ACO and hybrid ACGA", *Finite Element Analysis and Design*, 44, pg.505-512, 2008.
- [66] Kaveh, A., Laknegadi, K., & Zahedi, M., "A new algorithm for domain decomposition of finite element models", *International Journal for Computer-Aided Engineering and Software*, 25(5), pg.464-479, 2008.
- [67] Klimmek., T., KieBling, F., & Honlinger., H., "Multidisciplinary Wing Optimization Using a Wing Box Layout Concept and a Parametric Thickness Model", *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pg.1-10, 2002.

- [68] Schuhmacher, G., Murra, I., Wang, L., Laxander, A., O'Leary, O.J., & Herold, M., "Multidisciplinary Design Optimization of a Regional, Aircraft Wing Box", 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, pg.1-10, 2002.
- [69] Venter, G., & Sobieszczanski-Sobieski, J., "Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization", *Struct. Multidis. Optim.*, 26, pg.121-131, 2004.
- [70] Wunderlich, T.F., "Multidisciplinary wing design and optimization for transport aircraft", 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, pg.1-15, 2008.
- [71] Krog, L., Tucker, A., Kemp, M., & Boyd, R., "Topology Optimization of Aircraft Wing Box Ribs", 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, pg.1-11, 2004.
- [72] McIlhagga, M., Husbands, P., & Ives, R., "A Comparison of Search Techniques on a Wing-Box Optimization Problem", *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pg.614-623, 1996.
- [73] Ewing, M.S., & Downs, K., "Optimization of a rectangular cross-section wingbox using genetic algorithms", *AIAA Meeting Papers on Disc*, pg.1858-1867, 1996.
- [74] Yao Shen, J., & Ferguson, F., "Weight Reduction Through Optimal Arrangement of Force-Carrying Components Within Wing Box Structures", 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, pg.2181-2197, 2004.
- [75] Bondarabady, H. A. Rahimi, & Kaveh, A., "Nodal Ordering Using Graph Theory and a Genetic Algorithm", *Finite Elements in Analysis and Design*, 40, pg.1271-1280, 2004.
- [76] ANSYS, Inc., "15.5. Element Reordering", Theory Reference, Release 11.0 Documentation for ANSYS, 2007.
- [77] Hoit, M., & Wilson, E. L., "An Equation Numbering Algorithm Based on Minimum Front Criteria", *Computer and Structures*, 16, pg.225-239, 1983.
- [78] Flight International, "Piaggio Aero P180 Avanti II Data Sheet", Reed Business Information, 2007.
- [79] Rowlett, R., "Sheet Metal Thickness Gauges", University of North Carolina at Chapel Hill, <http://www.unc.edu/~rowlett/units/scales/sheetmetal.html>, 2002.
- [80] Piaggio Aero Industries, "Avanti II, P180: Specification and Description", USA: Piaggio America, Inc, 2005

- [81] Callister, W. D., "Material Science and Engineering: An Introduction", United States: John Wiley & Sons, Inc., 2003, pg.738, 741, 744, 746
- [82] Craig, R.R., "Mechanics of Materials, Second Edition", United States: John Wiley & Sons, Inc., 2000, pg.739
- [83] Anderson, J. D., "Fundamentals of Aerodynamics, Third Edition", New York: McGraw-Hill, 2001, pg.366-368
- [84] European Aviation Safety Agency, "Pratt & Whitney Canada PT6A-67 series engines", *Type Certificate Data-Sheet*, IM.E.008, 04, pg.5, 2007.
- [85] Camp, C., "Optimized Design of Two-Dimensional Structures using a Genetic Algorithm", *Journal of Structural Engineering*, 124, pg.551-560, 1998.
- [86] Rajeev, S., "Discrete optimization of structures using genetic algorithms", *Journal of Structural Engineering*, 118, pg.1233-1250, 1992.
- [87] Lewis, T., "Reduction of Genetic Algorithm Computation Time Using the Substructure Technique", Undergraduate Thesis Report, Ryerson University, 2002, pg.27, 31
- [88] Ali, N.F., "Optimization of Engineering Systems using Genetic Algorithm enhanced Computation Techniques", MAsC Thesis Report, Ryerson University, 2001, pg. 97, 103
- [89] Gellatly, R.A., & Berke, L., "Optimal Structural Design", Rep. No. AFFDL-TR-70-165, Air Force Flight Dynamics Laboratory, Wright-Patterson Air Force Base, Ohio, 1971.
- [90] Venkayya, V.B., "Design of Optimal Structures", *International Journal of Computers and Structures*, 1, pg.265-309, 1971.
- [91] Schmit, L.A., & Miura, H., "Approximation Concepts for Efficient Structural Synthesis", NASA CR-2552, 1971.
- [92] Qian, L.X., Zhang, W.X., Siu, Y.K., & Zhang, J.T., "Efficient Optimal Design of Structures – Program DDDU", *Computer Methods in Applied Mechanics and Engineering*, 30, pg.209-224, 1982.
- [93] Belegundu, A.D., "Study of mathematical programming methods for structural optimization", PhD Thesis Report, University of Iowa, Iowa City, Iowa
- [94] Hajela, P., & Berke, L., "Neurobiological computational models in structural analysis and design", *Computer and Structures*, 41(4), pg.657-667, 1991.
- [95] Coello, C.A., & Christiansen, A.D., "Multiobjective optimization of trusses using genetic algorithms," *Journal of Computers and Structures*, 75, pg.647-660, 2000.

- [96] Adeli, H., & Park, H.S., "Hybrid CPN-Neural Dynamics model for discrete optimization of steel structures", *Microcomputers in Civil Eng.*, 11, pg.355-366, 1996.
- [97] Adeli, H., & Park, H.S., "Neurocomputing for design automation", CRC Press, Boca Roca, Florida, 1998.
- [98] The MathWorks, Inc., "Genetic Algorithm and Direct Search Toolbox, Genetic Algorithm Options", Matlab R2007a Help, 2007.
- [99] Bhaskaran, R., Sibley School of Mechanical and Aerospace Engineering, Cornell University, "Semi-Monocoque Shell", <http://courses.cit.cornell.edu/ansys/shell/index.htm>, 2006.
- [100] Budgell, P., ROI Engineering, Toronto, Ontario, 2009.
- [101] ANSYS, Inc., "MATRIX50, Superelement (or Substructure)", Element Reference, Release 11.0 Documentation for ANSYS, 2007.
- [102] ANSYS, Inc., "10.2 Using Substructuring", Advanced Guide, Release 11.0 Documentation for ANSYS, 2007.
- [103] ANSYS, Inc., "17.6 Substructuring Analysis", Theory Reference, Release 11.0 Documentation for ANSYS, 2007.
- [104] Irons, B.M., "A Frontal Solution Program for Finite Element Analysis", *Journal for Numerical Methods in Engineering*, 2(1), pg.5-23, 1970.
- [105] Melosh, R. J., & Bamford, R. M., "Efficient Solution of Load-Deflection Equations", *ASCE Journal of the Structural Division*, 95(ST4), Proc. Paper 6510, pg.661-676, 1969.
- [106] ANSYS, Inc., "15.9.3 Frontal Solvers", Theory Reference, Release 11.0 Documentation for ANSYS, 2007.
- [107] Department of Transportation, FAA, TYPE CERTIFICATE DATA SHEET, Revision 18, No. A59EU, Piaggio P-180, 2007, pg.4
- [108] AbsoluteAstronomy.com, AbsoluteAstronomy.com, Piaggio P180 Avanti, http://www.absoluteastronomy.com/topics/Piaggio_P180_Avanti, 2007.
- [109] European Aviation Safety Agency, "EASA Type-Certificate Data Sheet, PT6A-66B", IM.E.008, 04, 2007, pg.5
- [110] Wadhwa, S., "Modern Methods of Teaching Physics", New Delhi: Sarup and Sons, 2001, pg. 240

564 lines of code
it in its entirety
overall process.

```
efront Auto/Geo

HeaderLines', 3

3.891];
53];
06];
```

```

total_part_eval_time = (polyval(part_poly,length(elem_num))) * pop * (gen + 1);
direct_part_time = polyval(direct_poly,length(elem_num));
time_savings = total_nonpart_eval_time - (direct_part_time + total_part_eval_time);
partition_opt_time = 0.1 * time_savings;

if time_savings > 0
    part_mod_1(core_file,q,lbfcriteria,balance_item,hybrid_option,partition_opt_time);
    struct_opt_part_mod_1(core_file,max_allowed_stress,max_allowed_disp,pop,gen);
else
    struct_opt_nonpart_mod_1(core_file,max_allowed_stress,max_allowed_disp,pop,gen);
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gov_tf = clock;
gov_diff_t = gov_tf - gov_ti;
gov_hours = gov_diff_t(1,3)*24 + gov_diff_t(1,4);
gov_minutes = gov_diff_t(1,5);
gov_seconds = gov_diff_t(1,6);
if gov_minutes < 0
    gov_hours = gov_hours - 1;
    gov_minutes = 60 + gov_minutes;
end;
if gov_seconds < 0
    gov_minutes = gov_minutes - 1;
    gov_seconds = 60 + gov_seconds;
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('Total Solution CPU Time:          %2.0d Hours, %2.0d Minutes, %6.4f Seconds\n', gov_hours,
gov_minutes, gov_seconds);
fprintf('\n\n');

```

Appendix B - ANSYS

Appendix B.1 - ANSYS Parametric Design Language (APDL) and Batch Code

For new users in ANSYS, the simplest way to perform a desired sequence of commands is to use the structured menus located in the graphical user interfaces (GUI). By navigating the tree structure, any desired command can be found. An alternative to this is to use what is called ANSYS Parametric Design Language (APDL). APDL refers to a language specific to ANSYS that allows the user to perform any command found in the menus by keying it in the command line found at the top of the GUI. Both menus and command line are identified below in Figure 62.

Figure 62 – ANSYS GUI

For experienced users that are familiar with APDL, a great deal of time can be saved by typing in specific commands instead of navigating the menu system, which can get quite cluttered at times. Furthermore, in problems that require re-valuation due to a few parameter changes, using conventional means can be quite time consuming since the geometry and mesh must be generated each time through the menus. By using APDL, the commands can be listed in a text file and changed very easily. Once the text file has been updated, it can be copied into the command line and the problem can be re-solved. Beyond just performing commands within the GUI, APDL can also be used in ANSYS batch mode, which allows a series of commands to be run in sequence without even having the ANSYS GUI open. For applications that require integration of more than one program, this can be a very powerful tool since any desired FE computation can be done remotely. All that is required in this sort of analysis is a text file listing all commands.

As far as learning APDL, there are several methods one can go about doing this. The first one to mention is that of the ANSYS Parametric Design Language Guide found in the **HELP** menu. This is a very thorough document and answers most if not all possible questions one might have regarding the topic. Even though a great deal of information is provided in this document, it may be difficult to apply some commands. Fortunately, an alternative method of acquiring APDL commands exists. Simply perform a desired command through the menu structure, and then open up the log file as shown in Figure 63.

Figure 63 – ANSYS Log File

The log file essentially lists all commands carried out but in an APDL format. Thus, any sequence of commands can be performed and determined through this feature to aid in the learning process. A simple example of APDL application is shown below, with a brief explanation given for each step.

<code>/PREP7</code>	- Enter the preprocessor directory
<code>et, 1, LINK8</code>	- Define element type 1 as spar element
<code>et, 2, SHELL63</code>	- Define element type 2 as shell element
<code>et, 3, SOLID45</code>	- Define element type 3 as solid element
<code>r, 1, 0.015</code>	- Define real constant 1
<code>r, 2, 0.015</code>	- Define real constant 2
<code>MP, EX, 1, 7.24E+10</code>	- Define material model 1 modulus of elasticity
<code>MP, PRXY, 1, 0.33</code>	- Define material model 1 poisson's ratio
<code>MP, DENS, 1, 2770</code>	- Define material model 1 density
<code>FINISH</code>	- Exit the preprocessor directory

Appendix B.2 - Substructuring Analysis

The substructuring feature in ANSYS was applied to this work in order to evaluate the partitioned meshes in a series format. A bottom-up approach was carried out as described in Section 10.2 of the ANSYS Advanced Guide [102]. This essentially means that the individual subdomains (superelements in ANSYS) were generated individually and combined later, as

opposed to creating the entire model, selecting a portion of it, and generating the superelement from that (top-down approach). A simple flowchart describes the bottom-up method in Figure 64 for an application dealing with 3 superelements.

Figure 64 – ANSYS Substructuring Analysis Flow Chart

To make the explanation of this process a little easier, a simple example will be shown as it is applied through the various phases for the model depicted in Figure 65 at $q = 3$. The explanation given here is essentially a simplified version of Section 10.2 of the ANSYS Advanced Guide [102].

Figure 65 – Simple FE Example

Assuming a partition has been defined, the first step in the bottom-up approach is called the generation (GEN) pass. Here, each superelement is modeled individually with boundary conditions. Each superelement must have master degrees of freedom defined, which in this application are the interface nodes of the generated partition. Once these have been defined, a substructure analysis type is selected and the problem is solved. This process is carried out for all the superelements sequentially, and results in 3 superelement matrix files (.sub). The GEN phase for this example is shown below in Figure 66.

Figure 66 – GEN Pass

The next phase of the substructuring process is that of the USE pass, which essentially uses the superelements generated in the GEN pass. The superelements are all imported into the same analysis by selecting the element MATRIX50. Care must be taken when importing each superelement to ensure that they are properly lined up and that they have proper nodal numbering offsets, unless they were all uniquely numbered in the GEN pass. Once all three superelements have been imported along with any applicable boundary conditions, the reduced degree of freedom problem is solved. This then creates a solution file (.dsub) that lists the solutions of only master degrees of freedom. Even though this application only used a static analysis type for the USE pass, it is also possible to perform a modal and transient analysis as well. A visual representation of the reduced superelements in the USE pass is shown below in Figure 67.

- Master Degrees of Freedom

Figure 67 – USE Pass

The final step in the substructuring process is to find the solution of all degrees of freedom in each superelement. This is handled by performing an expansion (EXP) pass. Given the master degree of freedom solutions already calculated in the '.dsub' file, a back substitution is performed to calculate all local degrees of freedom by turning on the EXPASS option under Analysis Type. The EXP pass shown in Figure 68 is repeated for each superelement sequentially to find results for the entire model.

Superelement 1

Superelement 2

Superelement 3

- Master Degrees of Freedom

Figure 68 – EXP Pass

It should be noted that when a particular model with q subdomains requires solution, a total of $2q + 1$ individual evaluations must be carried out to obtain results for the entire mesh. For a thorough explanation of the mathematics behind the static substructuring method used in ANSYS, refer to Section 17.6 in the Theory Reference documentation [103].

Appendix B.3 - Frontal Solvers

Frontal solvers were first discussed by Irons [104] and Melosh et al. [105]. The key feature distinguishing this solver from other sparse matrix solvers is the way in which it eliminates degrees of freedom in a given problem. This elimination process assembles the equation matrix based on the degrees of freedom contained in the nodes as it passes from one element to the next. A given degree of freedom remains active in the assembly matrix until the solver passes through the last element containing that particular degree of freedom. Once this is the case, the degree of freedom can be eliminated by way of gauss elimination. At any given point, the total number of active degrees of freedom is referred to as the wavefront. Due to the nature of this method, the assembly matrix shrinks and grows as it passes through each element sequentially based on numbering. It should be obvious at this point that element order plays a key role in reducing the wavefront. This is very important since the solution time for a frontal solver is proportional to the square of the wavefront. ANSYS uses an automatic reordering routine before each application of the frontal solver based on Hoit et al. [77]. In simplest terms, this reordering method tries to find 3 elements located at the extremities of the FE model and proceeds to renumber all elements sequentially from all 3 start points. In the end, the reorder method with the lowest wavefront is chosen. On top of this, ANSYS also incorporates a geometric reordering method [76] that sequentially numbers elements based on the 3 major co-ordinate axes. Again, the method with the lowest wavefront is selected. A flowchart for the frontal solver used in ANSYS is shown below in Figure 69 [106].

Figure 69 – Frontal Solver Flow Chart

Appendix C - Wingbox Load Calculations

The following material outlines the process in which loads were calculated for the wingbox optimization problem that used the Piaggio P180 Avanti II as a reference aircraft.

a) Limit Load Factor (FAR 23.337)

- Maximum Take-off Weight: 10810 lb [107]
- Equation:
$$n = 2.1 + ((24000)/(W + 10000))$$

$$n = 2.1 + ((24000)/(10810 + 10000))$$

$$n = 3.253$$

b) Total Lift (n = 3.253)

- Maximum Take-off Weight: 10810 lb or 4902.494 kg [107]
- Equation: $L = 9.81(W)(n)$
 $L = 9.81(4902.494)(3.253)$
 $L = 156462.08 \text{ N}$

c) Lift Generated by Main Wing

- Assume fuselage generates 20% of lift [108] and remaining proportional to surface area
- Main wing area: 16 m^2 [80]
- Canard wing area: 2.19 m^2 [80]
- Tail wing area: 3.83 m^2 [80]
- Total wing area (calculated): 22.02 m^2
- Main wing lift (both wings): $0.8(156462.08)(16/22.02) = 90949.80 \text{ N}$
- Main wing lift (one wing): 45474.90 N

d) Lift Distribution

- Elliptical lift distribution assumed [83]:

$$L'(y_w) = \rho_{\infty} V_{\infty} \Gamma_o \sqrt{1 - \left(\frac{2y_w}{b}\right)^2} \quad \text{Eq.18}$$

- Free stream density (ρ_{∞}): 101325 Pa
- Free stream velocity (V_{∞}): 368 km/h or 102.22 m/s (maneuver speed) [80]
- Wingspan (b): $5.963(2) = 11.926 \text{ m}$
- Span co-ordinate (y_w) $\text{Varied for 305 points along span (0.000 – 5.963 m)}$
- Circulation at wing root (Γ_o) was iterated through Equation 18 in spreadsheet until lift vectors summed up to lift produced by one wing (45474.90 N)
 - o $\Gamma_o = 1.835 \times 10^{-5} \text{ m}^2/\text{s}$
- Calculated elliptical lift distribution was applied in the positive z direction to 305 nodes on upper wing surface at $1/4$ chord location as shown in Figure 70.

Figure 70 – Loads Applied To Wingbox Problem

e) Induced Drag Distribution

- Induced angle of attack [83]:

$$\alpha_i = \frac{\Gamma_o}{2bV_\infty} \quad \text{Eq.19}$$

- Induced angle of attack calculated using Equation 19 based on information determined in section d)
 - o 7.526×10^{-9} [rad] or 4.312×10^{-7} [deg] (very small angle)
- Induced drag [83]:

$$D'_i = L'_i \alpha_i \quad \text{Eq.20}$$

- Induced drag calculated based on induced angle of attack and elliptical lift distribution at 305 nodes with Equation 20 in spreadsheet (L'_i is the lift distribution, i.e. L')
- Drag vectors applied at same location as the lift vectors, acting in the negative y direction

f) Engine Loads

- Weight: 220 kg [109]
- Number of mount points: 3
- Force at each mount point: $220(9.81)(3.253)/3 = 2340.418$ N (in negative z direction)
- Mount points correspond to rib locations on the upper skin of wing as shown in Figure 71.

Figure 71 – Engine Loads Applied to Wingbox

g) Fuel Loads

- Lower wing skin fuel area: 2.1636 m^2 (calculated)
- Fuel capacity in wing: 462.3 L [78]
- Density of kerosene: 0.81 kg/L [110]
- Total force due to fuel: $462.3(0.81)(9.81)(3.253) = 11950.909 \text{ N}$
- Distributed pressure on lower skin: $(11950.909)/(2.1636) = 5523.622 \text{ Pa}$

h) Inertial Load

- Acceleration: $3.253(9.81) = 31.912 \text{ m/s}^2$
- This inertial load was applied to the actual wing structure in ANSYS

Appendix D - Pseudo-Code for RGB Methods

Shown below is pseudo-code for the recursive graph bisection (RGB) approach used in the direct method by Vanderstraeten et al. [4]. Two versions of the code are given that correspond to topologically general decomposition and topologically 1-D decomposition [4].

a) Topologically General Decomposition:

Compute a diameter (n_1, n_2) of G

for each vertex i of G **do**

 Compute distances $d(i, n_1)$ and $d(i, n_2)$

Compute the distance parameter D

 Initialize to zero the number of vertices N_I in G_I

 Compute A_j as the number of vertices i in G such that

$$d(i, n_1) - d(i, n_2) = j$$

 Initialize distance parameter D :

$$D \leftarrow \min(j) \text{ over all non-zero } A_j$$

while $n_1 < N/2$ **and** $|(N_I + A_D) - N/2| \leq |N_I - N/2|$ **do**

$$N_I \leftarrow N_I + A_D, D \leftarrow D + 1$$

Return D

for each vertex i of G **do**

if $d(i, n_1) \leq d(i, n_2) + D$ **then** accept vertex i in G_I

else accept vertex i in G_2

b) Topologically 1-D Decomposition:

Initialize G_q with G

Compute a diameter (n_1, n_2) of G_q

Put n_1 in vertex list L

for $j = 1$ to $q - 1$ **do**

 Find vertex n_2 in G_q that is farthest from vertices of L

for each vertex i of G_q **do**

 Compute distances $d(i, L)$ and $d(i, n_2)$

 Compute the distance parameter D

for each vertex i of G_q **do**

if $d(i, L) \leq d(i, n_2) + D$ **then** accept vertex i in G_j

else keep vertex i in G_q

 Start building G_{j+1} with vertices in G_q that are adjacent to G_j

 Replace content of list L with vertices put in G_{j+1}