# Development and Integration of Geometric and Optimization Algorithms for Packing and Layout Design 

Santosh Tiwari<br>Clemson University, tiwaris@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations
Part of the Engineering Mechanics Commons

## Recommended Citation

Tiwari, Santosh, "Development and Integration of Geometric and Optimization Algorithms for Packing and Layout Design" (2009). All Dissertations. 388.
https://tigerprints.clemson.edu/all_dissertations/388

# Development and Integration of Geometric and Optimization Algorithms for Packing and Layout Design 

| A Dissertation |
| :---: |
| Presented to |
| the Graduate School of |
| Clemson University |
| In Partial Fulfillment |
| of the Requirements for the Degree |
| Doctor of Philosophy |
| Mechanical Engineering |
| by |
| Santosh Tiwari |
| August 2009 |
| Accepted by: |
| Dr. Georges Fadel, Committee Chair |
| Dr. Gregory Mocko |
| Dr. Joshua Summers |
| Dr. Margaret Wiecek |
| Dr. Mary Kurz |
| Dr. Vincent Blouin |

## ABSTRACT

The research work presented in this dissertation focuses on the development and application of optimization and geometric algorithms to packing and layout optimization problems. As part of this research work, a compact packing algorithm, a physically-based shape morphing algorithm, and a general purpose constrained multi-objective optimization algorithm are proposed. The compact packing algorithm is designed to pack threedimensional free-form objects with full rotational freedom inside an arbitrary enclosure such that the packing efficiency is maximized. The proposed compact packing algorithm can handle objects with holes or cavities and its performance does not degrade significantly with the increase in the complexity of the enclosure or the objects. It outputs the location and orientation of all the objects, the packing sequence, and the packed configuration at the end of the packing operation. An improved layout algorithm that works with arbitrary enclosure geometry is also proposed. Different layout algorithms for the SAE and ISO luggage are proposed that exploit the unique characteristics of the problem under consideration. Several heuristics to improve the performance of the packing algorithm are also proposed. The proposed compact packing algorithm is benchmarked on a wide variety of synthetic and hypothetical problems and is shown to outperform other similar approaches. The physically-based shape morphing algorithm proposed in this dissertation is specifically designed for packing and layout applications, and thus it augments the compact packing algorithm. The proposed shape morphing algorithm is based on a modified mass-spring system which is used to model the morphable object. The shape morphing algorithm mimics a quasi-physical process similar to the inflation/deflation of a balloon filled with air.

The morphing algorithm starts with an initial manifold geometry and morphs it to obtain a desired volume such that the obtained geometry does not interfere with the objects surrounding it. Several modifications to the original mass-spring system and to the underlying physics that governs it are proposed to significantly speed-up the shape morphing process. Since the geometry of a morphable object continuously changes during the morphing process, most collision detection algorithms that assume the colliding objects to be rigid cannot be used efficiently. And therefore, a general-purpose surface collision detection algorithm is also proposed that works with deformable objects and does not require any preprocessing. Many industrial design problems such as packing and layout optimization are computationally expensive, and a faster optimization algorithm can reduce the number of iterations (function evaluations) required to find the satisfycing solutions. A new multi-objective optimization algorithm namely Archive-based Micro Genetic Algorithm (AMGA2) is presented in this dissertation. Improved formulation for various operators used by the AMGA2 such as diversity preservation techniques, genetic variation operators, and the selection mechanism are also proposed. The AMGA2 also borrows several concepts from mathematical sciences to improve its performance and benefits from the existing literature in evolutionary optimization. A comprehensive benchmarking and comparison of AMGA2 with other state-of-the-art optimization algorithms on a wide variety of mathematical problems gleaned from literature demonstrates the superior performance of AMGA2. Thus, the research work presented in this dissertation makes contributions to the development and application of optimization and geometric algorithms.

## DEDICATION

This dissertation is dedicated to my late father, Ashwini Kumar Tiwari, and my mother, Vindhyawasini Tiwari who have been instrumental in shaping my life, my thought process, and have inculcated in me a desire to learn and succeed. They have always been and continue to remain a source of inspiration to me.

## ACKNOWLEDGEMENTS

This dissertation that you hold in your hands would not have been possible without the help, support, and continuous guidance of some extraordinary individuals.

I will forever remain indebted to my parents, who have supported and guided me in all my endeavors. It is their hard-work and upbringing of me that is primarily responsible for this successful completion of my PhD. I am also very thankful to my younger brother Pankaj for his moral support and encouragement and taking care of all the responsibilities while I worked on my PhD.

Equally encouraging has been the support and active guidance of my advisor Dr. Georges Fadel. He has been very patient and flexible, and has allowed me exceptional freedom to pursue my own research interests. I have greatly benefited from his wisdom and knowledge for which I shall forever remain thankful. He has been instrumental in my awesome educational experience at Clemson University. I would like to thank the members of my advising committee: Dr. Gregory Mocko, Dr. Joshua Summers, Dr. Margaret Wiecek, Dr. Mary Kurz, and Dr. Vincent Blouin for their positive feedback, helpful and thought-provoking questions, different perspectives on my research, and their constructive suggestions. I would also like to thank Dr. Cecil Huey, Dr. Kalyanmoy Deb, and Dr. Paul Joseph for helping me with my studies and research while at Clemson University. The input of all the above mentioned individuals has significantly enhanced the quality of my research and I was able to assimilate their knowledge and experience into my PhD dissertation. It has been a great learning experience talking to each one of them, and I have learned from them as much outside the classroom as I have inside the classroom. They deserve more
thanks than I can give them.
I would especially like to thank Dr. Peter Fenyes from General Motors Research \& Development for his insightful inputs on my research and for their generous financial support. I would also like to thank the Department of Mechanical Engineering at Clemson University for supporting me through the Teaching Fellowship program, the Graduate School at Clemson University for supporting me through the Distinguished Graduate Research Fellowship program, and the Automotive Research Center (ARC) for sponsoring our research.

I would also like to thank my fellow students at the Clemson Research in Engineering Design and Optimization (CREDO) laboratory. It has been a great pleasure working with them and I am indebted to them as a colleague. I would especially like to thank Dr. Hong Dong, Dr. Sudhakar Teegavarapu, and Dr. Vladimir Gantovnik for their active collaboration and help with my research. I would also like to thank my office-mates Chris Czech, Daniel McCullough, James Gibert, and Wenshan Wang for bearing with me. It was a great pleasure to have them as my office-mates and I will truly miss them.

## TABLE OF CONTENTS

TITLE PAGE ..... i
ABSTRACT ..... ii
DEDICATION ..... iv
ACKNOWLEDGEMENTS ..... v
LIST OF TABLES ..... ix
LIST OF FIGURES ..... x
1 PREFACE ..... 1
1.1 Dissertation Outline ..... 3
2 A FAST AND EFFICIENT COMPACT PACKING ALGORITHM FOR SAE AND ISO LUGGAGE PACKING PROBLEMS ..... 4
2.1 Introduction ..... 4
2.2 Literature Review ..... 6
2.3 Description of the Packing Algorithm for the SAE Luggage ..... 8
2.4 Description of the Packing Algorithm for the ISO Boxes ..... 25
2.5 Simulation Results ..... 29
2.6 Discussion ..... 36
2.7 Conclusion ..... 37
3 A PHYSICALLY BASED SHAPE MORPHING ALGORITHM FOR PACKING AND LAYOUT APPLICATIONS ..... 39
3.1 Introduction ..... 39
3.2 Survey of Shape Morphing Methods ..... 42
3.3 Shape Morphing for Layout Design ..... 50
3.4 Proposed Shape Morphing Method ..... 52
3.5 Demonstration of the Morphing Algorithm ..... 65
3.6 Conclusion and Future Work ..... 69
4 AMGA2: IMPROVING THE PERFORMANCE OF THE ARCHIVE- BASED MICRO GENETIC ALGORITHM FOR MULTI-OBJECTIVE OPTIMIZATION ..... 73
4.1 Introduction ..... 73
4.2 Current State-of-the-art ..... 77
4.3 Description of the AMGA2 ..... 80
4.4 Simulation Results ..... 94
4.5 Observations and Inferences ..... 112
4.6 Conclusion ..... 115
4.7 Ideas for Future Research ..... 116
5 CONCLUSION ..... 119
5.1 Concluding Remarks ..... 119
5.2 Summary of Contributions ..... 121
5.3 Suggestions for Future Work ..... 122
REFERENCES ..... 124

## LIST OF TABLES

2.1 Simulation results: 8 -box packing problem ..... 30
2.2 Simulation results: 34 -box packing problem ..... 31
4.1 Test Problems ..... 95
4.2 The Delineation Metric (FON, HOLE, OSY, TNK) ..... 104
4.3 The Delineation Metric (ZDT1 to ZDT6) ..... 105
4.4 The Delineation Metric (DTLZ1 to DTLZ6) ..... 106
4.5 The Hypervolume Metric (FON, HOLE, OSY, TNK) ..... 107
4.6 The Hypervolume Metric (ZDT1 to ZDT6) ..... 108
4.7 The Hypervolume Metric (DTLZ1 to DTLZ6) ..... 109

## LIST OF FIGURES

2.1 Conceptual flowchart of the proposed solution strategy ..... 10
2.2 Orthogonal orientations for a prismatic object ..... 14
2.3 Four possible orientations for the same bounding box ..... 15
2.4 Sample voxelized objects ..... 20
2.5 Sample trunk geometry ..... 22
2.6 Trunk with patches marked ..... 22
2.7 Increase in area/volume due to voxelization ..... 25
2.8 Orientations of an ISO-box corresponding to a surface patch ..... 27
2.9 Packing the trunk in two different orientations ..... 28
2.10 Success rate vs population size for the 8 -box packing problem ..... 30
2.11 8-box packing problem ..... 31
2.12 34-box packing problem ..... 31
2.13 SAE luggage set ..... 33
2.14 Completely General Problem ..... 34
2.15 ISO-box packing problem ..... 35
2.16 ISO-box packing problem with a simpler trunk geometry ..... 36
3.1 Before scaling and Boolean operations ..... 45
3.2 After scaling and Boolean operations ..... 45
3.3 Morphing Box in ANSA ..... 49
3.4 Mass-spring model in 2D ..... 54
3.5 Vertices in the counter-clockwise direction ..... 55
3.6 Forces acting on a mass point ..... 55
3.7 Pressure force directed towards outward normals ..... 57
3.8 Interference of nodes ..... 62
3.9 Retraction of the mass points onto the surface of the constraint object ..... 63
3.10 Retraction distance ..... 64
3.11 Flowchart of the Morphing Process ..... 66
3.12 Case study 1: initial configuration ..... 67
3.13 Case study 1: final configuration ..... 68
3.14 Case study 2: initial configuration ..... 68
3.15 Case study 2: final configuration ..... 69
3.16 Case study 2 : morphed object ..... 70
3.17 Case study 2 : morphed object ..... 70
3.18 Case study 2: morphed object after denoising ..... 71
3.19 Case study 2: morphed object after denoising ..... 71
4.1 Objective space plot for the zdt4 test problem ..... 84
4.2 Depiction of crowding distance computation ..... 86
4.3 Delineation Metric ..... 102
4.4 Hypervolume Metric ..... 103
4.5 NSGA-II on ZDT4 ..... 107
4.6 FastPGA on ZDT4 ..... 107
4.7 MOEA/D on ZDT4 ..... 110
4.8 GDE3 on ZDT4 ..... 110
4.9 AMGA on ZDT4 ..... 110
4.10 AMGA2 on ZDT4 ..... 110
4.11 NSGA-II on DTLZ6 ..... 110
4.12 FastPGA on DTLZ6 ..... 110
4.13 MOEA/D on DTLZ6 ..... 111
4.14 GDE3 on DTLZ6 ..... 111
4.15 AMGA on DTLZ6 ..... 111
4.16 AMGA2 on DTLZ6 ..... 111

## Chapter 1

## PREFACE

Optimization as a discipline finds applications in almost all branches of science, engineering, and commerce. Many real-world design problems involve posing and solving an optimization problem. The term optimization refers to the process of narrowing down to a solution (or a set of solutions) from a large pool of potential candidate solutions, such that the chosen solution(s) is/are better than the rest in certain respects. Optimization therefore comes into existence in any scenario where a choice has to be made. Often, the choices are made without explicitly resorting to specialized optimization methods and tools. In such a case, the process of optimization is implicit, and the decisions are made based on the intuition, expertise, and knowledge of the designer. In engineering problems such as packing and layout optimization (also referred to as configuration design), implicit optimization is not possible. This is due to the fact that either the set of potential candidate solutions is too large, or it is extremely difficult to efficiently explore a good representation of the entire search space. And thus, to solve this problem, specialized optimization methods and tools are required. The geometry is an important aspect in the design of components, subsystems, and systems. Many design optimization problems involve some form of geometric optimization. To find a better optimized solution for such problems, integration of geometric and optimization algorithms is required. The integration of geometric algorithms can not only improve the quality of the obtained solutions but is also sometimes necessary to perform
optimization [1, 2]. Thus geometric algorithms also act as enablers in the design process. In this dissertation, the integration of geometric algorithms with an optimization algorithm is presented in the context of layout design. When solving an engineering optimization problem, the users often start from an existing optimization algorithm and customize it for the specific problem instead of developing a new algorithm from scratch. The existence of general purpose high-performing optimization algorithms thus provides a good starting point for the customization and application. It is thus desirable to develop high-performing generic optimization algorithms that can be easily customized if desired.

This dissertation is a compilation of three journal papers that deal with geometric and optimization algorithms. The three algorithms presented in this dissertation are as follows.

1. A compact packing algorithm: This algorithm is designed to compactly pack freeform objects with full rotational freedom inside an arbitrary enclosure. It has several modules which are: an optimization algorithm, CAD algorithms, and layout heuristics. It also requires integrating all the modules together to accomplish the packing process. The publications related to this research are: [3], [4], and [5].
2. A physically-based shape morphing algorithm: This algorithm is designed to modify the shape of the deformable objects to achieve a desired volume. It is a free-form mesh-based shape morphing algorithm that works with arbitrary manifold geometries. It is fully automated and does not require any human intervention during the morphing process. This algorithm is specifically designed for layout optimization and thus incorporates features to avoid interference with the surrounding objects during the morphing process. It can be integrated with layout algorithms to obtain a better solution [6]. This algorithm is described in [7].
3. A general-purpose constrained multi-objective optimization algorithm: This is a new algorithm designed to solve an arbitrary constrained multi-objective optimization problem. It is based on evolutionary principles and also benefits from the existing
literature on optimization. It incorporates several novel concepts to achieve fast and reliable convergence on large optimization problems. It is a general-purpose optimizer that can be easily customized (if required) for a given engineering problem. The publications related to this research are: [8], [9], and [10].

### 1.1 Dissertation Outline

In this first chapter, a brief description of the three algorithms included in this dissertation is provided. The second chapter contains a description of the compact packing algorithm. In chapter 3, the physically-based shape morphing algorithm suitable for packing and layout applications is described. In the next chapter, the archive-based micro genetic algorithm (AMGA2) is presented. Finally, in chapter 5, concluding remarks, list of contributions, and suggestions for future research are presented.

## Chapter 2

## A FAST AND EFFICIENT <br> COMPACT PACKING

## ALGORITHM FOR SAE AND

ISO LUGGAGE PACKING PROBLEMS

### 2.1 Introduction

Packing problems are widely studied by various researchers primarily because of their importance in many real-world applications. In several industrial applications such as the packaging industry and automotive luggage loading, it is required to pack a set of objects in a prescribed volume such that the total area or volume of the encompassing region is minimized. Also, it is often required to determine a subset of objects that can be packed inside a given volume to achieve the highest packing efficiency. Compactness therefore is one of the objectives encountered in many packing optimization problems. It involves
minimizing the void space between the objects (this form of packing is often referred to as dense packing) and/or the wasted space (empty space not necessarily in between the packed objects). There could be several constraints on the relative placement of objects in a packing problem. These constraints are imposed by the specific type of problem under investigation. One obvious constraint that all packing problems must include is that there should be no overlap between the packed objects and that no object or portion of an object can lie outside the container. In [4], we proposed an algorithm to pack three dimensional free-form objects (with cavities and holes) inside an arbitrary enclosure such that the packing efficiency is maximized. In this paper, we propose several improvements to the proposed algorithm and extend it to solve the ISO luggage packing problem [11]. We also present a faster layout algorithm for the case of ISO luggage packing, and a heuristics based packing algorithm for large problem instances. The compact packing algorithms presented in this paper can assist a designer in determining the cargo volume of an automobile trunk.

The trunk of a typical automobile is often designed to hold as many objects of the largest size as possible. The auto manufacturers report the trunk capacity using one of two published standards for the cargo volume which can be considered as specializations of the general 3D packing problem. The two widely used standards are the SAE J1100 standard [12] (for the USA) and the DIN 70020 standard [11] (for the European Union). The DIN 70020 is identical to the ISO 3832 standard.

- SAE J1100: The SAE standard comprises a set of distinct objects, a subset of which can be packed inside the trunk. The optimization task in this case is to determine the subset of objects which will maximize the packing efficiency.
- ISO 3832/DIN 70020: The DIN standard comprises identical prismatic objects (1 liter boxes of dimension $200 \times 100 \times 50 \mathrm{~mm}^{3}$ ) that are to be placed inside the trunk. The optimization task in this case is to find the maximum number of such boxes that can be placed inside the trunk.

In both cases, it is also required to find the orientation and the location of every object that
is placed inside the trunk. In most cases, a physical luggage packing is done to determine the trunk capacity of an automobile. However, a compact packing algorithm may be used to guide the physical packing process. The trunk packing problem is a general case of the three-dimensional rectangular packing problem. The difficulties associated with the trunk packing problem are summarized below [1].

- The trunk-packing problem is highly multi-modal and therefore has multiple isolated local optima.
- The problem does not have a mathematical formulation and therefore a closed form solution does not exist.
- Because the packing procedure is computationally expensive, performing a large number of iterations to find the optimum solution becomes impractical.
- The size of the search space increases exponentially with the number of objects to be packed.
- The placement of objects requires computationally expensive collision detections.

The remainder of this paper is organized as follows. In section 2, a brief survey of the current literature in packing optimization is presented. Section 3 describes the packing algorithm proposed in [4] which is suitable for the SAE luggage problem. Section 3 also discusses the proposed improvements to the packing algorithm. In section 4, the modified layout algorithm for the ISO luggage problem and a heuristics based packing algorithm are presented. Section 5 contains a description of the test problems and lists the simulation results. In section 6, a discussion of the simulation results is presented. Section 7 presents a brief conclusion of the study.

### 2.2 Literature Review

Extensive research has been done into the development of packing algorithms both for the case of two $[13,14,15,16,17,18,19,20]$ and three dimensions $[21,1,22,23,24,25$,

26, 27, 28]. A survey of various computational approaches to perform layout optimization can be found in [1]. A description of various encodings used to represent the solution to a packing problem can be found in [3]. A review of current literature on packing optimization reveals that the two standards ISO 3832 and SAE J1100 require different solution approaches to determine the best packing efficiency for a given trunk. Eisenbrand et al. [25, 27] have proposed an algorithm to find the maximum number of identical boxes that can be placed inside a given automobile trunk. They have also shown that the packing problem is NPcomplete [29] and thus have proposed an approximate solution procedure. Voxel based representation is used by Eisenbrand et al. [25, 27] to represent the geometry of the trunk and the identical boxes. The approach proposed by Eisenbrand et al. handles only one object type and therefore cannot be used to determine the packing efficiency for the SAE J1100 standard. In [22] and [23], a packing algorithm based on extended pattern search is proposed to find the best packing efficiency using the SAE J1100 standard. In [22] and [23], the geometry of the objects and the trunk is represented using an octree [30] based data structure. In [26], a sequential heuristic-based layout algorithm is proposed to solve the rectangular packing problem. The solution approach proposed in [26] introduces the concept of meta-boxes which is similar to a branch and bound strategy. In [26], the packing volume is divided into smaller sub-volumes and a heuristic-based layout algorithm is used to pack the sub-volumes. While such an approach breaks a bigger problem into several smaller sub-problems, it introduces new optimization variables that describe the partitioning to create the meta-boxes. The most recent approach to compute the maximum packing efficiency for the SAE J1100 standard is proposed by Althaus et al. [28]. In [28], a branch and bound algorithm is proposed to determine the maximum packing efficiency. In this paper, we present 3D packing algorithms to solve both the SAE and ISO luggage packing problem.

The description of the SAE luggage set and the suggested procedure for evaluating the trunk capacity can be found in the SAE standard J1100 [12]. The SAE luggage set has a total of 8 distinct objects, each of which have multiple copies resulting in a total of

38 objects. The ISO luggage set consists of 1 liter $\left(200 \times 100 \times 50 \mathrm{~mm}^{3}\right)$ boxes. Since the packing optimization algorithm has been proven to be NP-complete, researchers have proposed approximate solution approaches to compute the packing efficiency in a reasonable amount of time. It should also be noted that none of the algorithms proposed [23, 27, 26, 28] for the packing problem can guarantee a globally optimal solution in a finite time except for the case of orthogonal rectangular packing. The definition of orthogonal rectangular packing can be found in [3]. In [15], an efficient packing algorithm for two dimensions has been proposed. The proposed approach is based on the bottom-left strategy [18] for the placement of an object. A layout algorithm based on the bottom-left strategy cannot fill the voids created by placing relatively large objects. An improved heuristic for two dimensions, namely bottom-left-fill (BLF), is proposed in [16]. BLF heuristic can be used to fill the voids and hence generate a denser packing. The packing algorithm proposed by Dowsland et al. [15] requires the computation of the no-fit-polygon [31]. Since it is computationally expensive to compute the no-fit-polyhedron in three dimensions, we use an iterative scheme to find the best location to place an object. Our approach uses a generalization of the BLF heuristic in three dimensions to pack the objects inside the container. The 3D version of the BLF heuristic is referred to as the BLBF (bottom-left-back-fill) heuristic in this paper. For the case of ISO luggage involving identical objects, the layout algorithm is modified to not attempt to fill the voids, and therefore the associated layout heuristic is referred to as BLB (bottom-left-back) heuristic in this paper.

### 2.3 Description of the Packing Algorithm for the SAE Luggage

This section contains the description of the packing algorithm proposed in [4]. This algorithm is designed to determine the maximum packing efficiency for the general 3D container packing problem. The general container packing problem places free-form objects with full rotational freedom inside an arbitrary enclosure such that the volume of the ob-
jects inside the enclosure is maximized. Thus, the objective for the packing problem is to maximize the packing efficiency. The packing efficiency is defined as the fraction of total volume of the container that is occupied by the packed objects. Since, the packing problem is NP-complete, it is not possible to perform an optimal packing in polynomial time (as a function of the number of objects). The approach used to solve this problem is therefore motivated from the human (or robot) packing of objects inside a container. The objects to be packed are placed sequentially inside the container one after the other in a specified orientation. The optimization task, therefore, is to find the optimal packing sequence and orientations of all objects. Given the packing sequence and orientations, an algorithm is required to perform the packing and compute the packing efficiency. Thus, the general packing problem can be broken into two sub-problems.

1. The optimization algorithm: Design an optimization algorithm capable of generating an optimal packing, thereby determining the optimal packing sequence (or position in 3D space) and the optimal orientation of every object.
2. The layout algorithm: Given the packing sequence (or position in 3D space) and the orientation of every object, develop a layout algorithm that can pack free-form objects. The algorithm should be able to pack the objects according to the provided sequence, ensure that no objects collide with each other, determine which objects can be placed inside the enclosure in the specified orientation in the remaining volume, and compute the packing efficiency.

The conceptual sketch of the proposed solution strategy is shown in Figure 2.1. The solution to both sub-problems is presented. We first discuss the optimization algorithm.

### 2.3.1 The Optimization Algorithm

The proposed optimization algorithm is an evolutionary algorithm and is coupled with layout heuristics to improve its effectiveness and performance. Evolutionary algorithms (EAs) are nature inspired adaptive search techniques [32, 33] which base their working


Figure 2.1: Conceptual flowchart of the proposed solution strategy
principle on Darwin's theory of the survival-of-the-fittest. EAs are flexible and powerful optimizers which do not impose any restriction on the optimization problem. EAs can efficiently deal with problems having discreteness and multi-modality in the search space. EAs do not require the optimization problem to have a functional form and do not rely on the gradients of objectives and constraints. Genetic algorithms (GA) [34, 35, 36, 37] are one class of the evolutionary techniques that have been successfully used as an optimization tool. Since a GA can work with almost any kind of solution representation (so long as suitable genetic variation operators are provided), it facilitates designing optimization algorithms for a wide class of single and multi-objective optimization problems. The generalized packing problem is modeled as a single objective optimization problem where the packing efficiency
is to be maximized. The working principle of the proposed single-objective optimization algorithm is based on the steady-state [38] genetic algorithm. The proposed optimization algorithm also borrows concepts from several existing single-objective genetic algorithms. Some of the notable efforts in designing single-objective optimization algorithms are: Simple Genetic Algorithm (SGA) by Holland and Goldberg [35], Evolution Strategies by Schwefel and Rechenburg [39], Genitor (a steady state GA) by Whitley [40], CHC (cross-elitist generation, heterogeneous recombination, cataclysmic mutation) by Eshelman [41], and Covariance Matrix Adaptation (CMA) by Hansen and Ostermeier [42].

Since the solution to the optimization problem is a packing sequence of oriented objects, the packing optimization problem is modeled as a combinatorial optimization problem with a suitable encoding of the sequence and orientations. The solution set is represented as a permutation of objects which defines the packing sequence. The orientation of objects is modeled using mixed variables. A pre-defined set of rules are used to decode the chromosome. The pseudo-code of the proposed genetic algorithm is as follows.

The optimization algorithm
Begin
Generate the initial population randomly.
Evaluate the initial population.
Repeat
Choose two random parents.
Create one offspring from the two parents using the genetic variation
operators.
6 Evaluate the offspring solution.
7 Choose a solution randomly from the population.
8 Compare the offspring against the chosen solution; if the offspring
has better packing efficiency, then replace the chosen solution with
the offspring.
9 Compute the diversity in the population.
If the diversity in the population is lost, then store the best
solution and regenerate the remaining population.
11 Until(100 % packing efficiency is reached or number of function
evaluations is exhausted).

```
11 End

Thus, the proposed optimization algorithm is an elite preserving steady-state genetic algorithm and incorporates an explicit diversity preserving mechanism. The algorithm does not have very high selection pressure (it does not follow the best solution at every iteration) which makes it more resilient to premature convergence. The genetic variation operators used to create the offspring solution depend upon the solution representation. Since the proposed algorithm solves a single objective optimization problem, the phenomenon of genetic drift drives the entire population towards a single point which often results in the loss of diversity. The algorithm therefore incorporates a diversity preservation operator. The diversity is computed in the variable space. Representation of the optimization variables (chromosome) is described next.

\subsection*{2.3.1.1 Description of the optimization variables}

The choice and representation of optimization variables has significant impact on the performance of the optimization algorithm. The optimization variables consist of two parts.
- The packing sequence: Since combinatorial optimization is performed, the packing sequence is represented by a permutation of object numbers.
- The orientations: The orientation of objects is represented using a multi-parity bit representation.

\subsection*{2.3.1.2 The packing sequence}

Let there be \(N\) objects that are to be packed inside a container. A permutation (packing sequence) for \(N\) objects is a sequence of the form \(\pi=\pi(1), \pi(2), \pi(3), \ldots, \pi(N)\), where \(\pi(i)(i=1 ; 2 ; \ldots N)\) denotes the index of an object. Also \(\pi(i) \neq \pi(j)\) for \(i \neq j\). The object with index \(\pi(1)\) is packed first. Then the object with index \(\pi(2)\) is packed and so on. For \(N\) distinct objects, the number of permutations \(=N!\) which also represents the size of the sequence search space. If there are \(k_{1}\) objects of the first type, \(k_{2}\) objects of the second type and so on, then the size of the search space \(S\) for \(M\) distinct types is given by Equation 2.1.
\[
\begin{equation*}
S=\frac{\left(\sum_{k=1}^{M} k_{i}\right)!}{\prod_{i=1}^{M}\left(k_{i}!\right)} \tag{2.1}
\end{equation*}
\]

\subsection*{2.3.1.3 The orientations}

The orientation variables cannot be represented with real numbers since they are a circular entity ( \(0^{\circ}\) and \(360^{\circ}\) are same). The genetic operators designed to work with real numbers (non-circular) will not reflect the circular property of rotation. Also, for prismatic and free-form objects, the number of possible orientations is different, both for the orthogonal case and for the continuous case. The desired characteristics of a good representation follow.
1. It should preserve the circular property of the rotation.
2. It should have minimal redundancy in the representation.
3. All orientations should be equally probable.
4. It should not impose any pseudo-ordering on the rotation variables.

The representation of orientation variables depends upon the complexity of the objects and the desired rotational freedom. An overview of each case follows next.


Figure 2.2: Orthogonal orientations for a prismatic object

\subsection*{2.3.1.4 Scheme 1: prismatic objects with orthogonal orientations}

Let the dimensions of the object be \((l \times b \times h)\), then the six possible orientations (span along \(x, y\), and \(z\) directions) are 1. \(l-b-h, 2\). \(l-h-b, 3 . b-l-h, 4 . b-h-l, 5 . h-l-b\), and \(6 . h-b-l\). The six possible orthogonal orientations for a prismatic object are depicted in Figure 2.2. Any of the three edges ( \(1, \mathrm{~b}, \mathrm{~h}\) ) may be oriented along x , either of the two remaining edges may be oriented along y , and the remaining edge must then be oriented along z . A single bit with a parity of 6 is used for every prismatic object for which only orthogonal orientations are desired. This representation satisfies all the desirable characteristics mentioned above. Since there are 6 possible choices; for \(N\) objects, there are \(6^{N}\) different combinations.

\subsection*{2.3.1.5 Scheme 2: free-form objects with orthogonal orientations}

Consider a cuboid with all the dimensions different, and faces marked as \((1,2,3\), \(4,5,6\) ). In this case, the opposite faces are different (marked) and hence for every state mentioned above, there are sub-states. Consider Case-1 from Figure 2.2. \((l-b-h)\). The Xnormal face can be represented by either of the two opposite faces (the faces are now marked and hence they are different) and the Y-normal face can also be represented by either of


Figure 2.3: Four possible orientations for the same bounding box
the two opposite faces (a different pair). Once the first two positions are fixed, the Z-face is automatically fixed. Hence for every state mentioned above, there are 4 sub-states. The four possible orientations for Case 1 are depicted in Figure 2.3. Hence in this particular case, there are a total of 24 distinct orientations for an object. The representation for Scheme 1 can be extended to accommodate this case by adding one extra bit for every freeform object with orthogonal orientation (the bit will have a parity of 4). In this particular case, orientation is represented using two bits. Mutating the first bit (parity 6) changes the bounding box of the object (large change), whereas mutating the second bit only changes the profile visible on every face of the bounding box (small change). With this scheme, there is no redundancy or pseudo-ordering and no explicit handling of the circular property is required. For \(N\) free-form objects, there are \(24^{N}\) possible combinations.

\subsection*{2.3.1.6 Scheme 3: free-form objects with full rotational freedom}

For free-form objects with full rotational freedom, a perturbation of \(\theta\), where \(-45^{\circ} \leq\) \(\theta \leq 45^{\circ}\), can be added to the rotation of the objects in each dimension. The perturbation does not represent the orientation but the difference in orientation. Three real variables are added for every object that has full rotational freedom. The size of the search space in this case is infinite. Thus, a complete chromosome for the case of three free-form objects with
full rotational freedom looks like:
- \(3,2,1\) (object permutation);
- \(6,1,5\) (prismatic orientation - parity 6 );
- \(2,4,3\) (facial orientation - parity 4 );
- \(15^{\circ}, 23^{\circ},-12^{\circ},(\) rotational perturbation about \(x)\)
- \(21^{\circ}, 13^{\circ},-34^{\circ},(\) rotational perturbation about \(y)\)
- \(22^{\circ}, 35^{\circ},-1^{\circ}(\) rotational perturbation about \(z)\).

The genetic algorithm needs crossover and mutation operators for each variable type in the chromosome. For permutation variables, order-based crossover [43] is used. Mutation for permutation variables is modeled using the swap operator. For multi-parity bits, onepoint crossover is used and mutation is modeled using bit flipping. For real variables, simulated binary crossover [44] is used. Polynomial mutation [45] for real variables is used to maintain diversity in the population.

\subsection*{2.3.2 The Layout Algorithm}

The layout algorithm receives the packing sequence and orientation for every object from the optimization algorithm and communicates with the CAD algorithms to generate the packed configuration. Following are the sequence of steps performed by the layout algorithm.
1. Receive the packing sequence and the orientation of every object from the optimization algorithm.
2. Construct the rotation matrix for every object.
3. Use the CAD algorithms to rotate every object (in triangulated form).
4. Request the CAD algorithms to voxelize all the objects whose bounding box could fit inside the bounding box of the container.
5. Pick the objects in the order of the packing sequence and pack them using the BLBF heuristic.
6. Compute the packing efficiency based on the volume of the objects inside the container.
7. Report the packing efficiency to the optimization algorithm.

The layout algorithm can be further broken down into two parts.
1. CAD algorithms: The CAD algorithms process the 3D CAD data, perform geometric transformations, collision detection, boolean operations etc.
2. Layout heuristics: The layout heuristics consist of a set of rules that specify the movement of an object until a suitable location for it is found. The layout heuristics query the CAD algorithms for geometric operations.

We first discuss the CAD algorithms required by the layout heuristics.

\subsection*{2.3.3 The CAD Algorithms}

The CAD algorithms are primarily required to handle the 3D CAD data. The geometry of the container and the objects to be packed is provided in the form of STL (Stereolithography) files. The STL format is an industry standard for rapid prototyping and is based on tessellation. STL is a neutral file format. Most CAD models can be converted into STL format. The STL format describes the geometry in terms of triangular facets and does not contain the connectivity information. The CAD algorithms require that all the input geometries must represent manifold objects. A very important operation required by the packing algorithm is collision detection. Collision detection (overlap computation) for the case of compact packing is tricky (and involved) since the objects have to be placed such that they are touching each other. With non-convex objects having cavities and holes,
it is extremely expensive (and often impossible) to compute penetration depth and the direction of movement. To overcome this limitation, the objects are voxelized (fragmented into a large number of small identical cubes lying on an ordered three dimensional grid) and then the collision detection and boolean operations are performed. Voxelization of the objects helps in performing extremely fast collision detections, unions, intersections etc. It also provides an opportunity for code optimization to extract maximum performance. For performing the packing using voxels, several algorithms were developed and implemented. The specific voxel algorithms developed are:
- Surface voxelization: The surface voxelization engine takes a binary STL file (CAD data in triangular format) and generates the corresponding voxel data. The conceptual procedure for surface voxelization is as follows.
1. Compute the bounding box of the object.
2. Construct the three dimensional matrix which circumscribes the bounding box of the object.
3. Determine the bounding box for every facet of the triangulated object.
4. For every cell of the bounding matrix, perform a triangle-box overlap computation. If the facet intersects the cell, mark the cell as non-empty. The triangle-box intersection method proposed in [46] is used for this purpose.
- Voxel inversion: Voxel inversion is used to extract the inner volume of the container. A voxel is assumed to be inside a container, if it cannot be reached from outside the container. To determine which voxels are inside the container, parallel rays are dropped from all the six faces of the bounding box, all the voxels that get illuminated are either on the surface or outside the container. To detect the voxels that may still be outside but are not illuminated by the rays, diffusion is used. The diffusion of light rays guarantees that all the voxels that can be reached from the outside of the container are identified. The voxels that are inside the container will not
be illuminated by the light rays since they will be obstructed at the surface of the container. To perform voxel inversion, all the voxels that are outside the container or on the surface of the container are marked. All the remaining voxels constitute the inside of the container. Note that the volume obtained using voxel inversion is almost always a complete subset of the actual inner volume. Thus, this approximation gives a conservative packing.
- Volume voxelization: Volume voxelization converts surface voxel data to volume voxels using ray tracing. This is in contrast to the voxel inversion in that the rays are stopped as soon as they touch the surface voxels. The surface voxels are not assumed to be illuminated by the rays. All the voxels that are not outside the object are assumed to be either inside or on the surface of the object and thus constitute the volume of the object.
- Overlap computation: To detect if two objects are overlapping, the physical coordinates (matrix indices) for the bottom-left-back corner are determined. Accordingly, the coordinates (indices) of all the cells in the matrix are determined. Thus, for the two objects, the physical location of all the cells is known. Based on the physical location, the relative index of all the cells (as compared to the entire voxel grid) is determined. The global matrix is parsed to determine if any voxel is occupied by the two objects; If a voxel is occupied by the two objects, they overlap with each other.

Some sample voxelized objects are shown in Figure 2.4. Several auxiliary routines are also required to implement the complete packing algorithm. With voxels, intersection and union operations are trivial. Emptying the grid involves marking/unmarking of voxels. Since the voxel data is linear and ordered, any voxel inside the matrix can be accessed in constant time if its location is known. If the location is not known, a binary search on the voxel indices can be performed to locate the requisite voxel. Further, the ordering of voxels allows for jumping inside the voxel matrix and the data can be accessed in any fashion desired. This flexibility allows for designing faster parsing/decoding algorithms and also


Figure 2.4: Sample voxelized objects
provides an opportunity for code optimization.
Overlap computation is the most expensive part of the entire packing algorithm. It is also the most frequent operation performed by the packing algorithm and as such almost \(90 \%\) of the total time is consumed by the overlap computation routines. To speed-up the collision detection algorithm, the following heuristics are proposed.
- If two objects overlap with each other, then their bounding boxes intersect. When two objects overlap, then either one object is completely inside the other, or their surfaces intersect. The bounding box check can be used to determine if one object is completely inside the other and thus this operation can be performed in constant time. If bounding boxes do not intersect, the objects do not intersect. Thus, the voxel matrix is parsed only in the case when the bounding boxes intersect and the bounding box of either object is not inside the bounding box of the other object.
- Since the voxel matrix is parsed only when the surfaces of the two objects intersect, we only need to parse the surface voxels. It should be noted that the voxel matrix is always axis aligned but the objects themselves need not be aligned with the axis.

The parsing starts from the outermost layer of the voxel matrix. If no collision is found in the outermost layer, the next inner layer is parsed and so on. This strategy significantly reduces the number of checks required to detect a collision.
- The size of the voxels is significantly smaller than the size of the objects. This implies that the contact region where the overlap occurs consist of multiple voxels. Therefore, instead of linearly parsing the voxels consecutively, voxels with odd indices are parsed before the voxels with the even indices (the parsing algorithm jumps inside the voxel matrix). This strategy helps in detecting the overlap quickly and is applied to all the three indices corresponding to the three coordinate directions.

The heuristics described above can speed-up the overlap computation by up to eight times as compared to a strictly linear parsing of the voxel matrix and also guarantee an exhaustive search in that if a collision occurs between two manifold objects, it will be detected by the overlap computation algorithm.

\subsection*{2.3.4 The Layout Heuristics}

The layout heuristics consist of a set of rules which are used to determine the location of an object. The layout heuristics also determine if an object can be placed inside the container. The layout heuristic proposed here is an extension of the original bottom-leftfill (BLF) heuristic to three dimensions and is referred to as bottom-left-back-fill (BLBF) heuristic. The BLBF heuristic is as follows.
- Step 1: Place the object with index \(\pi(1)\) at the bottom-left-back corner.
- Step \(i\) : Locate the bottom-most location where the object with index \(\pi(i)\) can be placed. If there are multiple such locations, find the left-most location, if there are multiple left-most locations, find the back-most location. There cannot be more than one location after the last search. Place the object with index \(\pi(i)\) at the location found. If no such location is found, then the object cannot be placed inside the container.


Figure 2.5: Sample trunk geometry


Figure 2.6: Trunk with patches marked

To implement the BLBF heuristic, the motion is started from the bottom-left-back position instead of the top-right-front position. The motion is continued until a position with no overlap with either the container or already placed objects is found. Such a strategy ensures that every object is placed at the bottom-left-back-most position available. The BLBF heuristic generates a denser packing and attempts to fill the holes and cavities wherever possible. The computational complexity of the BLBF heuristic varies as a cubic function of the grid resolution used for packing.

\subsection*{2.3.5 Patch Alignment}

Consider the trunk shown in Figure 2.5. The trunk geometry does not have smooth surfaces. There are small deviations between neighboring facets. In order to correctly align the objects with the trunk surface, and make the packing look intuitively correct, patch aligned orientations are introduced. These orientations are treated similar to orthogonal orientations and as such do not account for full rotational freedom. In practice, many objects will be aligned with the surface of the trunk and so it is desirable to explore all such possible orientations. Following is the sequence of steps performed to get patch aligned orientations.
1. Connectivity information for all the facets is generated. The connectivity information gives the three neighbors of a triangle.
2. All facets with downward pointing normals ( \(-Z\) direction) are reflected about the origin. From a packing standpoint, the facets that have \(180^{\circ}\) difference in their orientation are identical so far as alignment is concerned.
3. To generate an almost flat surface patch; an arbitrary facet is chosen, then its three neighbors are checked if they have similar orientation to the arbitrarily chosen facet. A user specified parameter that gives the allowed tolerance between the facets is used to compare the alignment of neighboring facets. If the difference in the normal directions is less than the user specified parameter, the facets are assumed to have similar orientation. Amongst the three neighbors, the ones with similar orientation are pushed onto a stack. An element (facet) is popped from the stack and its neighbors are checked. If the neighbors have similar orientation, they are pushed onto the stack. The process is repeated unless the stack gets empty. All the facets that were pushed onto the stack constitute an almost flat connected patch. This process is repeated unless all the facets of the trunk geometry are classified into different surface patches.
4. The patches are then pruned based on the total surface area. A user defined parameter is used as a threshold for the minimum allowed surface area of a patch. The patches which have a surface area greater than or equal to the user specified threshold are selected for generating orientations.

An advantage of using the above approach for patch alignment is that the trunk (relatively large flat faces) need not be axis aligned. It can have any arbitrary orientation, and the packing algorithm will suitably align the objects with the trunk surface. Also, it is not required to explicitly generate the orthogonal orientations; if the trunk geometry has orthogonally aligned patches, the orientations corresponding to those patches will be automatically generated. The surface of the trunk (container) after the facets are classified into patches is shown in Figure 2.6.

\subsection*{2.3.6 Additional Heuristics}

Thus, the complete packing algorithm consists of i) an optimization algorithm, ii) layout heuristics, and iii) CAD algorithms. To further improve the performance of the packing algorithm, the following rules are proposed.
- Only those objects whose bounding box dimensions (in the given orientation) are less than the corresponding dimensions of the bounding box of the container are attempted by the layout algorithm.
- If two identical objects in the same orientation are present in a solution, and if one of them could not be placed inside the container, the other is not attempted. Further, if one of them could be placed inside the container, the placement for the other starts at a location right after the last identically placed object.
- If the remaining volume inside the container is less than the volume of the object, then the placement of that object is not attempted.
- Because voxelization of an object increases its volume (Figure 2.7), the objects are scaled-down before voxelization is performed. The objects are scaled by an amount equal to the size of a single voxel.
- For objects with orthogonal or patch-aligned orientation, all possible configurations ( 6 for prismatic objects and 24 for free-form objects) are pre-generated and voxelized. Thus, during the iteration of the optimization algorithm, rotation and voxelization of such objects is not required.
- If an object is placed in the void space generated due to packing a larger object, the chromosome (packing sequence) is updated to reflect this change.
- For small problem instances involving large number of function evaluations, it is possible that identical chromosomes are generated during the later stages of the evolution.


Figure 2.7: Increase in area/volume due to voxelization

Also since the optimization problem has a single objective, as the population approaches the optimum, the probability of generating identical copies of a solution significantly increases. To prevent the redundant evaluation of identical solutions, a binary search tree (BST) is implemented. The BST stores all the generated solutions (chromosomes). Whenever a new solution is generated by the GA, it is first checked for its presence in the BST. If it is already present, it is discarded and a new solution is generated. If it is not present, it is inserted into the tree and is evaluated. Use of a BST allows for adding/searching a solution in logarithmic time. Lexicographic ordering of the chromosome is used by the BST for sorting and searching.

\subsection*{2.4 Description of the Packing Algorithm for the ISO Boxes}

The ISO standard for the cargo volume of a trunk concerns with determining the number of 1 liter ( \(200 \times 100 \times 50 \mathrm{~mm}^{3}\) ) boxes that can be placed inside the trunk. For the trunk of an automobile, the number of such boxes that can be placed inside typically ranges from 300 to 600 . There are two major difficulties associated with this problem.
1. The size of the search space for a problem with \(n\) prismatic objects having only orthogonal orientation is: \(n!\times 6^{n}\) which is extremely large for \(n\) in the range of 300 to 600. For more general cases, the size of the search space increases further. An optimization algorithm would require a proportionate increase in the number of function evaluations to find a near-optimal solution. Since the packing problem is computationally expensive, it is not possible to perform that many function evaluations in a reasonable amount of time. Hence, an optimization algorithm cannot be used to obtain good solutions.
2. The empty space remaining inside the trunk after the packing can be categorized into two types. There is empty space between the packed objects and empty space between the packed objects and the trunk surface. As the size of the objects grows smaller in comparison to the size of the trunk, the proportion of empty space between the objects and the trunk grows smaller. Also, as the number of objects increase, the proportion of empty space in between the packed objects increases. Hence, most of the empty space is due to the gaps in between the packed objects. It is therefore imperative to design a packing algorithm that attempts to minimize the empty space in between the packed objects.

It is comparatively easy to eliminate the empty space in between the packed objects if the packed objects are prismatic and are stacked together. Stacking the prismatic boxes together eliminates the empty space in between the packed objects. It can also be noted that if the objects being packed are identical and have identical orientation, hole filling is not required. Thus, the layout algorithm can be modified to not attempt the hole filling. Thus, the modified layout algorithm for the ISO boxes is computationally faster than the BLBF heuristic for the general case. It should also be noted that, since all the boxes are identical, any packing sequence will give the same result, hence it is no more required. Owing to these facts, and an emphasis on reducing the empty space in between the packed objects, the following packing heuristic is proposed.


Figure 2.8: Orientations of an ISO-box corresponding to a surface patch

Using the patch alignment technique discussed in the previous section, all such patches that contribute to orientations are determined. Typically the number of such patches vary in between 3 and 6 . For every surface patch, six orientations of the ISO box are generated. The six orientations of the ISO box corresponding to an arbitrarily oriented surface patch is shown in Figure 2.8. It should be noted that if the inclination between two surface patches is orthogonal ( \(90^{\circ}\) or its multiple), they will result in the same set of orientations. Hence, all redundant patches are removed before determining the orientations.

Let the total number of patches be \(m\). Then there are ( \(6 m\) )! possible permutations of the orientations. A packing is attempted for each permutation and the corresponding packing efficiency is reported. Let one such permutation be given by \(\pi=\) \(\pi(1), \pi(2), \pi(3), \ldots, \pi(m)\). Here \(\pi(i)\) corresponds to the \(i^{t h}\) orientation. The Iso-box in the orientation \(\pi(1)\) is chosen and the modified BLB (bottom-left-back) layout algorithm is used to pack as many copies of the ISO-box as possible. Then the ISO-box in the orientation \(\pi(2)\) is chosen and the modified BLB layout algorithm is applied. The process is repeated unless all orientations in that permutation are attempted. We now discuss the modified BLB (bottom-left-back) heuristic.
- Step 1: Place the ISO-box in the given orientation at the bottom-left-back corner.


Figure 2.9: Packing the trunk in two different orientations
- Step \(i\) : Starting from the bottom-left-back corner of the previously placed ISO-box, determine the next bottom-left-back-most location where the ISO-box can be placed. If no such location is found, the layout process is finished. If a location is found, place the ISO-box and repeat this step.

The proposed BLB heuristic is faster than the BLBF heuristic because, the search for the next feasible location starts right after the currently placed object.

Another important consideration is the orientation of the container itself. For a given orientation of an ISO-box, the packing is started from the bottom-left-back corner of the trunk. If the trunk is rotated by a multiple of \(90^{\circ}\), the geometry at the bottom-left-back corner would change which may change the obtained packing efficiency. This effect is demonstrated in Figure 2.9. The two orientations of the trunk result in different packing efficiency for the same set of orientations for the ISO-box (horizontal orientation is attempted before the vertical orientation in Figure 2.9). Such an effect may be observed only on non-prismatic trunk geometry. It is therefore important to choose an appropriate orientation for the trunk. There are eight corners in the bounding box of the trunk and therefore eight possible starting locations. For a given corner, the trunk may be further oriented such that the profile visible along each of the coordinate directions is different. For a given corner, there are three possible choices. Thus, a general trunk can have 24 possible orientations; each of which may potentially result in a different packing efficiency
for the same orientation of the ISO boxes. For an exhaustive search therefore, all 24 possible orientations of the trunk corresponding to a surface patch are attempted by the packing algorithm.

\subsection*{2.5 Simulation Results}

Simulation results on synthetic and hypothetical packing problems are presented in this section. In the following sub-sections, success rate is defined as the number of times \(100 \%\) packing efficiency is obtained out of 99 packing simulation runs starting with different random seeds. To make the packing algorithm easy to use, all the optimization tuning parameters are hard-coded and the only tuning parameter exposed is the number of iterations. For all the simulation results reported in this paper, the probability of crossover is set to 1.0 , and the probability of mutation is set to \(1 / N\), where \(N\) is the number of objects to be packed. In order to determine a suitable population size, rigorous benchmarking was done with the 8 -box packing problem. The plot of success rate versus population size is shown in Figure 2.10 for 1,000 iterations. It is evident from the Figure 2.10 that in general, the success rate reduces with increase in the population size. Similar trends were observed for the 34 -box packing problem. Thus, it is suggested to set the population size for the genetic algorithm equal to the number of objects to be packed. All the simulations were performed on a computer with 2 GB DDR2 667 MHz RAM and 2 GHz Intel Core 2 Duo processor.

\subsection*{2.5.1 8-box packing problem}

This is an orthogonal rectangular packing problem. If properly placed, all 8 boxes fit perfectly inside the container and \(100 \%\) packing efficiency is possible. The dimensions of the container are \(100 \times 100 \times 100\). The dimensions of the 8 boxes are \(25 \times 70 \times 75,55 \times 70 \times 75\), \(30 \times 75 \times 80,20 \times 75 \times 100,25 \times 35 \times 55,25 \times 35 \times 45,25 \times 45 \times 65\), and \(25 \times 55 \times 65\). The grid resolution used for the voxel matrix is \(5 \times 5 \times 5\). The simulation results for the 8 -box packing


Figure 2.10: Success rate vs population size for the 8 -box packing problem

Table 2.1: Simulation results: 8-box packing problem
\begin{tabular}{|r|r|}
\hline Iterations & Success Rate \\
\hline 500 & 22 \\
\hline 1,000 & 47 \\
\hline 1,500 & 66 \\
\hline 2,000 & 75 \\
\hline
\end{tabular}
problem are given in Table 2.1. The packed configuration is shown in Figure 2.11. Simulation time for this problem for 2000 iterations (function evaluations) is less than 1 second.

\subsection*{2.5.2 34-box packing problem}

This is an orthogonal rectangular packing problem involving selection. 17 out of 34 boxes fit perfectly inside the bin and \(100 \%\) packing efficiency is possible for this problem. The dimension of the container is \(13 \times 11 \times 9\). The dimensions of the boxes are (number of copies given in parenthesis): \(9 \times 6 \times 4(2), 7 \times 6 \times 5(2), 6 \times 6 \times 4(2), 6 \times 5 \times 4(2), 5 \times 4 \times 4(2)\),



Figure 2.12: 34-box packing problem

Figure 2.11: 8-box packing problem

Table 2.2: Simulation results: 34-box packing problem
\begin{tabular}{|r|r|r|}
\hline Iterations & Success Rate & Time (seconds) \\
\hline 5,000 & 19 & 4 \\
\hline 10,000 & 33 & 8 \\
\hline 15,000 & 42 & 12 \\
\hline 20,000 & 52 & 15 \\
\hline
\end{tabular}
\(6 \times 4 \times 3(2), 5 \times 4 \times 3(6), 6 \times 5 \times 2(2), 5 \times 3 \times 3(4), 5 \times 3 \times 2(4), 5 \times 2 \times 2(4)\), and \(5 \times 3 \times 1\) (2). The simulation results for the 34-box packing problem are given in Table 2.2. The packed configuration is shown in Figure 2.12.

\subsection*{2.5.3 SAE luggage set (38 objects)}

The SAE luggage packing problem involves placing objects of various shapes and sizes with handles into a hypothetical trunk. A full description of the SAE luggage set can be found in [22]. A sample packed configuration is shown in Figure 2.13. Again, 99 packing simulations are performed starting with different random seeds. Highlights of the
simulation results follow.
- Number of function evaluations: 10,000
- Grid resolution used: \(10 \times 10 \times 10\left(\mathrm{~mm}^{3}\right)\)
- Number of voxels: \(124 \times 133 \times 55=907,060\)
- Best packing efficiency: \(72.95 \%\)
- Number of objects corresponding to the best packing efficiency: 21
- Median packing efficiency: \(69.74 \%\)
- Execution time (for single simulation): 1 hr 8 mins approx

In [23], the best packing efficiency reported is \(65 \%\) with an octree level of 6 . Since the trunk used in our simulation has dimensions \(1302.02 \times 1210.07 \times 524.55 \mathrm{~mm}^{3}\), and the grid resolution used is 10 mm , the equivalent octree level is \(\log _{2}(1302.02 / 10) \approx 7\). In [23], the execution time with octree levels 4,5 , and 6 is \(0.9,5.7\), and 47.5 minutes respectively. Thus, the proposed algorithm is significantly faster in terms of execution time. It should however be noted that part of this speed-up is due to a faster CPU. In [28], for a large trunk, the maximum volume occupied by the packed objects is \(0.359 \mathrm{~m}^{3}\) (without using the H boxes). The packing efficiency has not been reported in [28]. The trunk used for our simulation has a volume \(=0.51 \mathrm{~m}^{3}\). For \(72.95 \%\) packing efficiency, the volume of the objects inside the trunk is approximately equal to \(0.37 \mathrm{~m}^{3}\). It should be noted that in general, larger trunks have larger maximum packing efficiency. The packing efficiency increases with increase in volume of the trunk because the fraction of space not utilized due to the trunk geometry reduces. It is not possible to directly compare the simulation results with other approaches since the two simulations use different trunk geometry and possibly different representations for the packed objects.


Figure 2.13: SAE luggage set

\subsection*{2.5.4 Completely general problem}

This is a hypothetical packing problem similar to the SAE luggage packing problem. This problem has 40 complex free-form objects with cavities and holes. The simulation results on this problem are reported merely to demonstrate the generality and effectiveness of the proposed packing algorithm. We use the same trunk as in the SAE luggage packing problem. The packed configuration is shown in Figure 2.14. Highlights of 99 packing simulations follow.
- Number of function evaluations: 10,000
- Grid resolution used: \(10 \times 10 \times 10\left(\mathrm{~mm}^{3}\right)\)
- Number of voxels: \(124 \times 133 \times 55=907,060\)
- Best packing efficiency: \(30.85 \%\)
- Number of objects corresponding to the best packing efficiency: 30
- Median packing efficiency: 28.9 \%


Figure 2.14: Completely General Problem
- Execution time (for single simulation): 4 hrs approx

\subsection*{2.5.5 ISO-box packing problem}

The ISO box packing problem involves packing as many 1 liter boxes as possible in a given volume. The trunk geometry used for examples C. and D. is also used for this simulation. The packed configuration is shown in Figure 2.15. Highlights of the simulation results follow.
- Deterministic packing algorithm (single simulation required)
- Grid resolution used: \(5 \times 5 \times 5\left(\mathrm{~mm}^{3}\right)\)
- Number of voxels: \(246 \times 264 \times 108=7,013,952\)
- Number of surface patches \(=1\)
- Number of function evaluations \(=1 \times 24 \times 6!=17,280\)


Figure 2.15: ISO-box packing problem
- Best packing efficiency: \(81.87 \%\)
- Number of ISO-boxes inside the trunk \(=417\)
- Total execution time \(=4\) hrs, 48 minutes approx.

The volume of the trunk used is \(508,692,398.82 \mathrm{~mm}^{3}\), which is equivalent to 509 ISO boxes. In [27], for trunks with equivalent volumes of 951 and 1201 ISO boxes, the reported packing efficiency is \(67.4 \%\) and \(67.5 \%\) respectively. The packing efficiency increases with the increase in size of the trunk due to the fact the fraction of unoccupied volume reduces. Thus, the results for the ISO-box packing reported in this paper are better than reported in the literature. It should be noted that this comparison is not completely accurate because of the difference in the trunk geometry used for the simulation.

\subsection*{2.5.6 ISO-box packing problem with a simpler trunk}

This is a hypothetical problem similar to the ISO-box but with a simpler trunk which has large flat surface patches. The presence of such flat patches results in a large number of possible orientations for the ISO boxes. The packed configuration is shown in Figure 2.16. Highlights of the simulation results follow.


Figure 2.16: ISO-box packing problem with a simpler trunk geometry
- Deterministic packing algorithm (single simulation required)
- Grid resolution used: \(5 \times 5 \times 5\left(\mathrm{~mm}^{3}\right)\)
- Number of voxels: \(257 \times 141 \times 188=6,812,556\)
- Number of surface patches \(=4\)
- Number of function evaluations \(=4 \times 24 \times 6!=69,120\)
- Best packing efficiency: \(87.99 \%\)
- Number of ISO-boxes inside the trunk \(=340\)
- Total execution time \(=19 \mathrm{hrs}, 15\) minutes approx.

\subsection*{2.6 Discussion}

As is evident from the simulation results on the completely general problem, the proposed algorithm is capable of packing free-form objects inside an arbitrary enclosure.

On the 8 -box and 34 -box synthetic problems, \(100 \%\) packing efficiency is obtained. Thus, the proposed packing algorithm can perform exact packing for orthogonal rectangular packing problems like the 8 -box and 34 -box. The proposed algorithm is computationally faster and requires fewer function evaluations to achieve results similar to those reported in [23], [26], [28], and [27]. Similarly for the same number of function evaluations, the proposed algorithm obtains a higher packing efficiency. The proposed algorithm is thus an improvement in the current state-of-the-art in packing optimization. From the simulation results on the completely general problem, it becomes evident that the performance of the algorithm is not drastically affected if the packed objects have complex geometry. It can also be inferred from the simulation results on the ISO-box packing problem with simpler trunk geometry that exploiting patch aligned orientations results in a higher packing efficiency. Although the simulation results are better than those reported in the literature, the proposed algorithm has some limitations.
- Significant distortion in the shape of objects occurs if low grid resolution is used.
- The memory requirement varies as a cubic function of the grid resolution.
- With the proposed encoding scheme, the size of the search space increases exponentially as a function of the number of objects.
- The packing efficiency obtained with full rotational freedom is poor and also consumes more time. Assigning arbitrary orientations disturbs the staggering of the objects causing large voids to appear in between the packed objects.

The limitations of the proposed algorithm provide us an opportunity for further improvement.

\subsection*{2.7 Conclusion}

In this paper, an algorithm for packing free-form objects inside an arbitrary enclosure is proposed. Separate packing algorithms for the SAE and ISO luggage set are
proposed. The proposed packing algorithm consists of an optimization algorithm, a layout algorithm, and CAD algorithms. The packing problem is modeled as a single objective optimization problem and the objective for maximization is the packing efficiency. The geometric data for the container and the objects are represented using voxels. The packing optimization algorithm is designed to be highly resilient to premature convergence. For the ISO packing with an extremely large number of objects, a purely heuristic-based packing algorithm is used. Since the enclosure is a freeform object, and a denser packing is desired, the BLBF heuristic in three dimensions is proposed for the general case. To further improve the performance of the packing algorithm, several heuristics are proposed. For the case of ISO-boxes, the fact that all the objects being packed are identical is exploited to design a faster layout algorithm. The simulation results clearly demonstrate the superior performance of the proposed packing algorithm. There is opportunity for improvement to the proposed approach, which shall be undertaken in future work.

\section*{Chapter 3}

\section*{A PHYSICALLY BASED SHAPE MORPHING ALGORITHM FOR PACKING AND LAYOUT APPLICATIONS}

\subsection*{3.1 Introduction}

The packing or layout problem (also referred to as configuration design) is concerned with the placement of components in 3D space such that they satisfy a set of constraints and meet or exceed a set of criteria. These problems are often encountered in industries such as luggage and container loading, stock cutting, electronics circuit board layout, factory layout/piping, and mechanical component layout etc. Broadly speaking, the layout design deals with methods and processes to determine the arrangement of a set of components to achieve a set of objectives without violating spatial or performance constraints. A comprehensive survey of various types of packing and layout problems and associated solution procedures can be found in \([3,21,1,22,24,26,27,28]\). Traditionally, the algorithms pro-
posed to solve the layout problem assume the components to be rigid and do not attempt to modify their shape. The feasible layout (placement) of components depends on their shape and size apart from other problem specific factors. In many applications however, modifying the shape and the size of some of the components during the layout process can result in a more efficient layout or placement. One such application is the design of reservoirs that hold fluids. During the layout design of an automobile, the fuel tank and other reservoirs that hold specific fluids are designed to have a specified volume. These components are designed to occupy a certain location whilst simultaneously satisfying constraints like ground clearance, no interference with the surrounding components, and assembly and manufacturability requirements. The morphing algorithm presented in this paper is ideally suited for the design of such components. It also is a first step in performing system design considering geometric integration as part of the design of individual components.

The current practice in industry is to design such components manually using CAD software. The designer interactively issues the instructions to the CAD software to generate the geometry of such components. After the components are designed, they are then integrated (brought) into the overall vehicle model and subsequent design and analysis is performed. This process is iterated until a suitable design is obtained. The shape morphing algorithm presented in this paper is an attempt to automate this step of the design process by automatically generating the geometry of such components. It is designed to generate an accurate representation of the geometry of the components whose shape is primarily governed by their location and the available empty space at that location. The proposed morphing algorithm is fully automated and does not require any human intervention during the morphing process. Thus it can be directly incorporated as part of the layout design process where the geometry of a morphable component can be automatically generated and fine-tuned to improve the performance characteristics of an automobile. The use of the shape morphing algorithm also significantly reduces the time to design such components thereby speeding up the layout process.

There exist many shape morphing methods in the computer graphics industry
[47, 48, 49, 50, 51, 52, 53] but most of them are not applicable to layout design. Most of these methods morph one shape into another; i.e. the initial and final shapes are known, and the morphing algorithm generates smooth interpolating frames. In the case of layout design, the final shape of a component is not known a priori. Furthermore, many of these methods require human assistance during the morphing process, a time consuming characteristic for the layout design. A morphing method is suitable for layout design if it generates a manifold geometry and takes into account the spatial constraints imposed by the surrounding components. Also, it should not make any assumption about the final shape of the morphable component. Additionally, the typical components used in an automobile are free-form and therefore parameterized morphing methods are unsuitable. A free-form morphing method, while more difficult, has the potential to achieve better performance characteristics. The geometric representation of an object directly affects the choice of the morphing algorithm. In this paper, a mesh-based surface representation satisfying the manifold condition is used to represent the geometry of the morphable object. This surface representation is used to generate the mass-spring model with a mass at every nodal point and a spring at every edge. The mass-spring model is then subjected to a physical process which is akin to inflating a balloon by blowing air inside it. The applied physical phenomenon subjects the mass points to various forces which cause the nodes to move. The collision of the mass points and the connecting edges with the surrounding objects and their subsequent response prevents any interference. Thus, the physically-based mass-spring model guides the morphing process to generate a satisfycing geometry.

The remainder of this paper is organized as follows. In section 2, a survey of different morphing methods is presented along with a discussion of their strengths and drawbacks. In section 3, the special requirements imposed by the layout problem on the morphing methods are discussed, and a set of conditions are laid out that must be satisfied by the morphing method. Section 4 contains the description of the shape morphing method and the modifications done to it to make it suitable for layout design applications. In section 5, illustrative examples that demonstrate the use of the proposed morphing algorithm are
presented. Finally, section 6 concludes the paper and presents possible future work.

\subsection*{3.2 Survey of Shape Morphing Methods}

Shape morphing is a non-affine transformation operation that changes the shape and/or size of an object. It is performed in many fields such as computer animation, structural design, aircraft/automobile external shape design, and mechanical component design. For the case of layout design, the task of the morphing process is to determine a shape that is optimal for some specified objectives whilst satisfying the volume constraints and preventing any interference with the surrounding objects. Modifying the shape of the objects during the layout process is a relatively new concept and only a few examples that incorporate this idea exist in the literature. In [54] and [55], the layout design of a VLSI circuit board is described in which the size parameters of the rectangular L-shaped and Tshaped micro-cells are changed during the floor planning stage so as to minimize the overall size of the circuit board. This approach has been referred to as soft or flexible block method by the researchers in [55]. The modification of geometry has also been incorporated in the layout design of the cross-section of an automatic transmission of a motor vehicle [56]. The automatic transmission system described in [56] consists of a set of clutches and a planetary gear train. The shape of the clutches is modified and the octree [23] representation is used to perform the morphing of the components in [56]. In [57], a bi-level layout optimization approach has been proposed. At the top (system) level, the rectangular blocks are moved to maximize the compactness, and at the component level, the dimension of the rectangular blocks is modified (parametric morphing) to minimize the overlap between the components. In [58], a 2D compact packing algorithm is described where the shape of the objects to be packed is slightly modified to achieve a higher packing efficiency. The morphing examples discussed above are in two dimensions and work with relatively simple geometry. A search of the existing literature for 3D free-form mesh-based morphing for layout and packaging applications did not yield any suitable examples. Thus, the application of shape morphing
during the layout design is a relatively new concept. However, there exist various types of morphing methods which are discussed next.

\subsection*{3.2.1 Parametrization-based Morphing}

Parametrization-based morphing is based on the boundary representation of the geometry. It is realized by altering the parameters that define the geometry of the object. Usually, the boundary is defined via some geometric form (e.g. a set of splines), and the parameters associated with this form are changed. In addition to the boundaries, certain features on the object are also identified and the associated parameters are manipulated during the morphing process. Most commercial CAD systems support some form of parametric shape morphing. This process is relatively straightforward and very convenient since the geometry can be modified by manipulating a small number of parameters that define the shape of the object. However, parametric approaches make strong assumptions on the shape of the object which limit their scope and may not always result in an optimal shape. Hence, this method is suitable for problems in which the designer already has an idea of the final shape [59]. Another limitation of this approach is that the volume cannot be expressed analytically especially if the boundary is represented using splines. Keeping the volume of the object at a specified value during the morphing process would then require solving a non-linear set of equations at every iteration. Parametrization based morphing is widely used in the aircraft industry for aerodynamic shape optimization [60, 61]. It is also widely used in structural design \([62,63]\) and automotive design \([64,65]\).

\subsection*{3.2.2 Scaling Coupled with Boolean Differences}

This process is not exactly a morphing method, but it has the potential to generate a desired geometry. The underlying assumption in this method is that it is possible to achieve a desired volume by scaling an object. The basic idea is to scale the object by an arbitrary factor such that it is larger/smaller than the desired volume. For the case of scaling up to increase the volume, the morphable object may overlap with the surrounding
objects. A Boolean difference of all the surrounding objects that overlap with the scaled object is performed to obtain the desired shape. The entire process is as follows.
- Assume that the initial size of the object corresponds to a scaling factor of 1 .
- Scale up/down the object by an arbitrary factor \(f\) such that the obtained volume is larger/smaller than the desired volume after performing the Boolean difference with the overlapping objects.
- Since, the increase/decrease in volume varies monotonically with the scaling even in the presence of Boolean differences with surrounding objects, the exact scaling factor can be determined using any zero-finding method.

This method is computationally inexpensive as compared to free-form mesh-based morphing methods and requires fewer iterations. However, the primary drawback of this method is the manufacturability aspect of the obtained geometry. Due to the Boolean operations, the obtained geometry has unnecessary artifacts (complex features) on its surface and undesired protrusions. This drawback is pictorially depicted in Figure 3.2 for the case of two dimensions. As is evident from Figure 3.2, there are unnecessary protrusions in the obtained shape. It can be observed from the Figure 3.2 that the obtained shape has an exact imprint of the surrounding geometries which may not always be desirable and may unnecessarily complicate the manufacturing process. Another disadvantage of the Boolean operation is that it may split the scaled object into multiple separate components which is undesirable.

\subsection*{3.2.3 Octree and Voxel Based Morphing}

In the octree representation, octrees are generated from the base geometry of the morphable components. Each octree has a local coordinate system attached to it and a 3-D vector is associated with it. Morphing is realized by scaling the octree model along a local axis. Yin et al. [56] studied a 2-D transmission layout problem with shape-able components based on an octree representation. In [56], the shapes of the transmission components are


Figure 3.1: Before scaling and Boolean operations


Figure 3.2: After scaling and Boolean operations
morphed by scaling the cells. With octrees, the shape morphing ability is very limited since only the scaling of individual cells is performed. It is relatively difficult to control the scaling operation with the octree since the typical objects encountered in industry when converted to an octree based representation contain a huge number of cells. Additionally the scaling of different cubes in the octree hierarchy needs to be coordinated to obtain a valid geometry. Voxel-based morphing can be interpreted as a special case of octree-based morphing where all the blocks (cubes) are of the same size. Instead of scaling the cubes as in octree, the cubes are added/removed as desired to morph the shape of the object.

\subsection*{3.2.4 Mesh-based Morphing}

Mesh-based morphing is by far the most common and robust free-form shape morphing method found in the literature. Mesh-based morphing is supported by both the surface and volumetric mesh representations. The two primary mesh-based morphing methods are geometric and physical methods.

\subsection*{3.2.4.1 Geometric Methods}

Geometric methods rely purely on geometric constructs to modify the surface or volumetric mesh. Some of the widely used geometric methods to morph a tessellated surface
involve i) directly moving the vertices of the surface mesh, ii) warping the triangles [49], and iii) using mesh transformation operators such as swap, collapse, and split [66]. A common method to morph a volume mesh is based on the extended direct surface manipulation technique [48]. Chen et al. [67] use this method to morph a CAE mesh of an automobile structure and its underlying components. In this case, the depth function [48] is used to account for the volume of the morphable components.

In mesh based morphing with geometric techniques, the variables are the coordinates of vertices of the mesh. The use of the vertices provides great flexibility to change the shape of an object when compared with parametrization based morphing. However, the morphing process is hard to control since there are a large number of design variables (coordinates). Generally, the morphing using geometric techniques is performed interactively, and the designer knows how to transform the shape. A drawback of the mesh-based techniques is their tendency to generate wiggly shapes [68]. A post-processing step usually consisting of smoothing [69] is required to achieve the desired surface quality. Often, the geometry based techniques do not involve the volume of the object as one of the considerations.

\subsection*{3.2.4.2 Physically-based Morphing Methods}

Instead of using purely geometric techniques, the computer graphics community has also explored physically-based morphing methods for modeling the deformation of objects. Physically based methods are typically based on a tessellated representation or volumetric finite element mesh. The mass-spring system is one of the physical models that has been used for modeling deformable objects [47, 70, 71, 50, 72, 51]. In the mass-spring system, an object is modeled as a collection of mass points (located at the mesh nodes) connected with springs (located on the edges of the mesh). The spring forces can be linear or nonlinear depending on the kind of behavior to be simulated. The motion of the mass points is governed by Newton's Second Law. There are many successful applications of the massspring model to represent physical objects. The mass-spring model has been used to model the motion of a fabric [72], facial simulation [51], and tissue deformation in surgery [70].

A primary advantage of mass-spring based morphing methods is that they are generally easy to implement and can be adapted for various applications. Since the model is physically-based, many different physical phenomena can be simulated to achieve the desired morphing behavior. The mass-spring model does not place any assumption on the final shape of the morphed components, and there is no need to explicitly specify the deformations. The deformations are governed by the physical phenomenon being simulated. One of the major drawbacks of the mass-spring system is that the computational models used to simulate the physical phenomenon do not often converge or require very high spatial and temporal resolution to ensure stability and accuracy of the obtained solutions. Another limitation of the mass-spring models is that the system tends to oscillate due to its iterative nature [50]. We propose a modification to the mass-spring model that reduces this oscillatory behavior. The oscillation is primarily due to the mass (inertia) of the nodes. Temporarily adjusting the velocity of the mass points to zero when at equilibrium reduces the oscillatory behavior. This modification however causes the mass-spring model to deviate from the real physics and thus the mass-spring model discussed in this paper can be considered as a quasi-physical model. It also does not accurately follow the governing physical laws in order to reduce the computational cost. The obtained geometry thus may not be exactly accurate from a physics stand-point using the proposed mass-spring model, however visually there is no noticeable difference in the obtained surface mesh. The volume control during the morphing process is achieved by controlling specific physical properties such as spring constants and the pressure [71].

\subsection*{3.2.4.3 Mesh-based Morphing Software}

There also exist several mesh-based morphing software. Most commercial meshbased morphing software are based on geometric techniques and involve a high level of interactivity. As discussed above, the common methods used for morphing the mesh in this case include moving vertices (nodes), warping of triangles, transformation operators (swap, collapse, and split), and using external geometries as drivers in the morphing process.

Some CAE pre and post-processing software that incorporate morphing capability are: HyperMorph (Altair Corp.), ANSA (Beta CAE Systems Inc.), and Meshworks/Morpher (DEP Inc.). These software support direct morphing (direct manipulation of vertices) as well as parameterized morphing which can be user defined. The ANSA from Beta CAE systems is one of the more advanced morphing software and is discussed next.
\(A N S A\) is a pre-processing tool for finite element analysis. The "Morphing Tool" in ANSA is used for the morphing process. The software modifies the shape of a 3D meshbased model by creating special geometric entities referred to as "Morphing Boxes" (see Figure 3.3). The Morphing Box is the basic entity of the Morphing Tool. The Morphing Box can include elements such as lines, shells, solids, and any combination thereof. Changing the shape of the Morphing Box morphs the elements inside the box. The elements inside the Morphing Box follow the motion/deformation of the box. The shape of the Morphing Box is controlled by the "Control Points" (shown in red color in Figure 3.3) located at its corners and edges. Thus, the morphing parameters can be specified in terms of translation of the Control Points. There are some limitations of the commercial morphing software. For example, for morphing with ANSA, the designer has to create the Morphing Boxes, define Control Points, and specify the translation of those Control Points. Thus, the designer must know which portions of the deformable object to morph and how to morph to achieve the desired shape and volume. Additionally, the morphing process is manual and the designer has to operate the software interactively in the visual mode to accomplish the morphing process. Thus, with ANSA, the shape morphing is limited and determined by the definition of control points and the morphing actions from the designer. Furthermore, this morphing technique is only suitable for small tweaks and cannot be used to make dramatic changes to the geometry, i.e. this technique cannot be used to generate desired geometries from a simple starting shape such as a sphere. The "Morphing Box" approach limits the degree of change since it does not allow the mesh elements that are inside the bounding box to go outside. Alternatively, if a significantly large "Morphing Box" is chosen to effect dramatic changes in shape and volume, the mesh quality significantly deteriorates.


Figure 3.3: Morphing Box in ANSA (taken from [73])

To summarize, the parametrization based shape morphing method is straightforward and easy to control, but has strong assumptions on the form of the final shape. It is suitable for morphing objects when their final approximate shape is known. The mesh based method is much more flexible and has the potential to dramatically change the geometry. When this method relies on purely geometric techniques to accomplish the morphing, it is performed interactively, and the huge number of control points are typically very hard to control. When using a physical model with mesh based morphing, there is no need to explicitly specify the deformations of the mesh, and the motion of nodes is governed by physical principles. The solution to the equations describing the physical principles predict the motion of the node points. However, the physical system must be integrated with the mass-spring model and very small time-steps need to be used to ensure the stability and convergence of the solution. For the parametric mesh morphing with commercial software, the morphing freedom is confined by how the control points are defined and by user-defined morphing actions. Also, only small changes are allowed by the commercial morphing software. Obviously, no morphing technique outperforms all others on all aspects; the choice of the morphing method is dependent on the application.

\subsection*{3.3 Shape Morphing for Layout Design}

This section discusses the aspects of morphing specific to layout design. Shape morphing for layout design is an exercise in modifying the form to fit the function(s). This type of morphing is functionally different than the ones typically used in the computer graphics industry in that, the shape of the object is governed by aspects such as spatial occupancy, dynamics, manufacturability, assembly requirements, and other domain specific constraints. There are three types of components typically encountered in layout design.
- Fixed Shape Components: The components such as engine, transmission, and axles do not change their shape during the layout process. The shape and size of these components is predetermined based on their functional requirements which are often external to layout design. However, during the initial phase of the design process, some approximate measure of their shape, size, and, location is used to guide the layout process. The redesign of such components is a major undertaking and therefore is done offline and not in tandem with the layout process.
- Parametrically Morphable Components: Certain components can only be morphed by modifying their control parameters. These components have limited morphing freedom and their shape cannot be adjusted arbitrarily. For example, the radiator assembly in an automobile could be modified during the layout design to maximize the cooling; but it has to satisfy certain operational characteristics which dictate its morphability. Such components are not necessarily designed offline, but are modified parametrically to better accomplish their desired function.
- Freeform Components: Components such as fluid reservoirs, fuel tank, and casings that hold components together can be morphed to any shape so long as it is easy to manufacture and assemble them. Their primary function is dictated by the component layout. Fluid reservoirs are typically designed to occupy a specified volume whilst preventing interference with neighboring components. The casings that hold the components together are designed to provide mount points and their shape is determined
based on the geometry of components contained inside. There is no assumption on the final shape of these components. The morphing algorithm presented in this paper is designed for this category of components. The geometry of such components can be accurately determined and optimized during the layout design process.

For a morphing method to be applicable to layout design, it must also incorporate collision detection and collision response to prevent interference with the surrounding components. Also, since the layout process is performed automatically with optimization algorithms, it is desirable that the shape morphing be done automatically through programming or scripting. User interaction therefore cannot be used to assist the morphing process during the layout design. The following conditions must be satisfied by the morphing method for it to be applicable to layout design problems.
1. There should be no assumption on the initial or the final shape of the morphable components.
2. The morphing method should be able to effect dramatic and significant changes to the shape of the components (infinitesimal changes will not suffice).
3. The morphing process should be automatic, robust, and should not require any user interaction during the morphing process.
4. There should be a way to control the volume of the morphable object.
5. The morphing method should be capable of working with any manifold geometry and also must output a manifold geometry.
6. The morphing method must not make any assumption about the constraint geometries (surrounding objects). The constraint geometries need not be manifold.
7. Since it is possible during the layout design process that multiple morphable components are present in the close vicinity of each other, the morphing method should allow simultaneous morphing of multiple interacting components.

\subsection*{3.4 Proposed Shape Morphing Method}

Having laid out the requirements for the morphing process in the previous section, we now discuss the modified mass-spring based shape morphing method. We modify the original mass-spring model to emulate a quasi-physical process resembling the inflation of a balloon. The modifications to the original mass-spring model are performed to speed-up the morphing process. The surface of the deformable object is modeled as a collection of mass points (located at the nodes of the surface mesh) connected with tension springs (located at the edges of the surface mesh). It is proposed to inflate the deformable surface to occupy a desired volume. The mass points forming the surface nodes are subjected to spring forces, pressure force, and contact forces upon collision with the surrounding components. Newton's second law of motion governs the motion of mass points and is used to compute the acceleration of the mass points. The set of equations describing the acceleration of mass points are integrated with respect to time to obtain the updated position of mass points. The motion of the mass points constitutes the morphing process.

In order to prevent the interference of the morphable object with the surrounding objects, collision detection is required. Most general purpose collision detection algorithms [74, 75, 76, 77] are designed to work only with rigid objects undergoing affine transformations. Since, in the present case this assumption is not true for the morphable object, most collision detection algorithms proposed in the literature cannot be efficiently used to determine the interference between the morphable object and the constraint objects. One way to use existing collision detection algorithms is to re-initialize the collision detection engine at every iteration of the morphing process. This would require pre-processing of the morphable object at every iteration which will make the collision detection computationally very expensive. We propose a simple surface-based collision detection algorithm that does not require preprocessing of the CAD data and works with deformable geometries. The proposed collision detection algorithm can perform Yes/No type queries and can also give all pairs of colliding facets if desired. Additionally, the proposed collision detection
algorithm can work with arbitrary geometries and does not require the colliding objects to be manifolds. It is easily parallelizable and thus is ideally suited for multi-core and/or multi-processor systems. Once the pair of colliding facets are obtained, the next step is to determine the vertices of the colliding facets that penetrate the surface of the constraint objects. The motion of the mass points on the deformable object after the collision is modified to prevent any interference with the surrounding objects. We now discuss the mass-spring model in detail.

\subsection*{3.4.1 The Mass-spring Model}

In the mass-spring system, an object is modeled as a collection of mass points (located at the mesh nodes) and connected with springs (located at the edges of the mesh) as shown in the Figure 3.4. The connection only exists between neighboring mass points. A linear spring is used to model a perfectly elastic object, while non-linear springs are used to model an object such as human skin that exhibits inelastic behavior. Since the objects involved in the layout design are artificially morphed, the linear spring model is appropriate to simulate their behavior during the morphing process. To control the volume of the object, gauge pressure is introduced. If the gauge pressure is zero and the compressive spring forces are nonzero, then deflation (reduction in volume) can occur. If it is desired to keep the volume at a constant value, an appropriate gauge pressure is introduced [71]. In the present case, it is proposed to continuously increase the gauge pressure until a specific target volume is achieved. The motion of the mass points is governed by the forces acting on it. A simple algorithm to generate the mass-spring model from an arbitrary manifold geometry is discussed next.

\subsection*{3.4.1.1 Generation of the mass-spring model}

A valid mass-spring model can only be generated from a manifold geometry. Any real-world object is a manifold. Let there be \(N_{f}\) facets on the surface of the morphable object. Each facet has exactly three edges and each edge is shared by exactly two facets,


Figure 3.4: Mass-spring model in 2D
hence the number of springs \(N_{s}\) in the mass-spring model is given by Equation 3.1. Since \(N_{s}\) is an integer, it is concluded that a manifold geometry contains an even number of facets. The number of mass points cannot be computed from the number of facets, since a vertex is shared by an arbitrary number of facets.
\[
\begin{equation*}
N_{s}=3 N_{f} / 2 \tag{3.1}
\end{equation*}
\]

Before the generation of the mass-spring model, all the nodes in the surface mesh are assigned a unique index which identifies the node. Once the nodes are numbered, the mass-spring model is generated by parsing the surface mesh. Since each edge is shared by two facets, it is encountered twice while parsing the surface mesh one facet at a time. Suppose a facet has vertices \(v_{1}, v_{2}\), and \(v_{3}\) in the counter-clockwise direction as shown in Figure 3.5. Then it contains edges \(\left(v_{1}, v_{2}\right),\left(v_{2}, v_{3}\right)\), and \(\left(v_{3}, v_{1}\right)\). When these edges are parsed in the counter-clockwise direction from the neighboring triangles, the vertices will be visited in the opposite order (Figure 3.5). Since we want to include an edge only once, it is suggested to add a spring corresponding to an edge if the index of the second vertex is higher than the first vertex. This simple rule will ensure that springs corresponding to all the edges are included exactly once in the mass-spring model.


Figure 3.5: Vertices in the counter-clockwise direction

\subsection*{3.4.1.2 Forces acting on mass points}

Once the mass-spring model is created, forces acting on the mass points can be computed. The mass points are located at the vertices of the surface mesh and are subjected to the force of gravity \((\vec{G})\), elastic spring force \(\left(\vec{F}_{s}\right)\), and internal pressure force \(\left(\vec{F}_{p}\right)\) as depicted in Figure 3.6. The formulation for each force term is discussed next.


Figure 3.6: Forces acting on a mass point

Gravity Force: The mass points experience the force of the gravity. However, this force is not relevant for layout purposes since the actual object is a rigid body and gravity does not play any significant role in the morphing process. Hence, the gravity force is
assumed to be zero.
\[
\begin{equation*}
\vec{G}=m g=0 \tag{3.2}
\end{equation*}
\]

Spring Force: The spring force acts along the edges between two vertices connected by an edge. The formulation for the spring force is taken from [51]. The spring is assumed to have linear elastic and damping response. The spring force is given by Equation 3.3. In Equation 3.3, \(\vec{F}_{s}\) is the spring force, \(k_{s}\) is the elastic spring constant, \(k_{d}\) is the linear damping constant, \(s\) is the original length of the spring, and \(d\) is the position vector of the mass point. The position vector \(d\) is computed using the coordinates \(V_{i}\) and \(V_{i+1}\) of mass points as given by Equation 3.4.
\[
\begin{gather*}
\vec{F}_{s}=-\left(k_{s}(|d|-s)+k_{d} \frac{d \dot{d}}{|d|}\right) \frac{d}{|d|}  \tag{3.3}\\
d=V_{i+1}-V_{i} \\
\dot{d}=\dot{V}_{i+1}-\dot{V}_{i}  \tag{3.4}\\
|d|=\left|V_{i+1}-V_{i}\right|
\end{gather*}
\]

The subscript \(i\) in Equation 3.4 denotes the index of the vertices (mass points).
Pressure Force: To increase the volume of the object, the pressure is artificially increased (equivalent to adding additional moles of gas) by a small amount at every iteration. This disturbs the equilibrium of the mass points since the pressure force now exceeds the spring forces. This increase in the pressure force causes the object to inflate. During the inflation process (iteration), both the pressure and the volume continuously change. In the original formulation [71], the inflation process obeys the ideal gas law. The pressure is updated at every iteration to reflect the change in the volume. Hence, as the volume increases, the pressure drops and vice versa. The reduced pressure reduces the rate at which the object expands. In this paper, the ideal gas law is not followed to update the value of the pressure. The pressure is artificially kept at a constant value or its value is
incremented by a small amount at every iteration. Since, the pressure force is the driver behind the morphing process, this modification speeds up the morphing process.

The pressure force acts on the facets of the surface mesh. The magnitude of the pressure force on a facet is proportional to its area. The pressure force acts in the direction of the outward normal of that facet. The direction of the pressure force is depicted in the Figure 3.7. The pressure force \(\vec{F}_{p}\) is given by Equation 3.5. In Equation 3.5, \(p\) is the gauge pressure, \(A\) is the area of facet under consideration, and \(\hat{n}\) is a unit vector directed towards the outward normal of the facet.


Figure 3.7: Pressure force directed towards outward normals
\[
\begin{equation*}
\vec{F}_{p}=p A \hat{n} \tag{3.5}
\end{equation*}
\]

The pressure force \(\vec{F}_{p}\) as given by Equation 3.5 acts on the facets and is assumed to be equally distributed between the three vertices (mass points). It should be noted that the assumption of equal distribution is invalid for a true physically based system. To obtain the actual pressure force acting on each node of the facet, the pressure force \(\vec{F}_{p}\) has to be written as the sum of the unknown forces acting on each node. The force equilibrium equation for each node then must be written in terms of the unknown pressure forces. Since the massspring system is statically indeterminate, the deformations must also be taken into account to solve for the pressure forces. Since the nodes interact with each other, the forces on a node cannot be determined independently, rather the complete system has to be formulated
as a set of matrix equations and has to be solved simultaneously. Since this step is extremely computationally expensive, the pressure force is simply divided equally between the three nodes of a triangular facet. Since most of the facets form almost equilateral triangles and the surface mesh is reasonably smooth, this approximation does not induce significant error in the mass-spring model.

The volume of the object is computed using the tessellated formulation. The volume computation assumes that the object is a manifold. Let the vertices of the facet \(i\) be \(v_{i 1}\), \(v_{i 2}\), and \(v_{i 3}\) in CCW order (as seen from the outward normal). Then, the volume of the object is given by Equation 3.6, where \(N\) is the number of facets.
\[
\begin{equation*}
V=\frac{1}{6} \sum_{i=1}^{N}\left(v_{i 1 x}+v_{i 2 x}+v_{i 3 x}\right)\binom{\left(v_{i 2 y}-v_{i 1 y}\right)\left(v_{i 3 z}-v_{i 1 z}\right)-}{\left(v_{i 2 z}-v_{i 1 z}\right)\left(v_{i 3 y}-v_{i 1 y}\right)} \tag{3.6}
\end{equation*}
\]

\subsection*{3.4.1.3 Motion of mass points}

The motion of the mass points is computed based on the forces acting on them. The acceleration of the mass points is directly computed from the forces as given by Equation 3.7.
\[
\begin{equation*}
a_{i}(t)=\frac{d^{2} \vec{r}_{i}}{d t^{2}}=\frac{\vec{F}_{i}}{m_{i}} \tag{3.7}
\end{equation*}
\]

It is desired to determine \(\vec{r}_{i}(t)\) for each mass point. The initial value (original configuration) of the position vector \(\vec{r}_{0}(t)\) is known. The first order approximation (Euler method) is used to solve the ODE described by Equation 3.7. The Taylor series expansion for \(r(t+\Delta t)\) approximated to the first term is given by Equation 3.8.
\[
\begin{equation*}
r(t+\Delta t)=r(t)+\dot{r}(t) \Delta t+\Theta(\Delta t)^{2} \tag{3.8}
\end{equation*}
\]

Hence, the error in the above approximation is of the order \(\Theta(\Delta t)^{2}\). It is thus recommended to use a very small step size to reduce the error. The error in the value of \(\vec{r}_{i}(t)\) accumulates over time, and the value of \(\vec{r}_{i}(t)\) may diverge if a suitably small step size
is not used. With this approximation, the \(\vec{r}_{i}(t+\Delta t)\) is given by the set of equations 3.9.
\[
\begin{gather*}
a_{i}(t)=\frac{F_{i}(t)}{m_{i}} \\
v_{i}(t+\Delta t)=v_{i}(t)+a_{i}(t) \Delta t  \tag{3.9}\\
r_{i}(t+\Delta t)=r_{i}(t)+\frac{v_{i}(t)+v_{i}(t+\Delta t)}{2} \Delta t
\end{gather*}
\]

\subsection*{3.4.2 Collision Detection and Response}

Till now, the surrounding objects have not yet been introduced into the morphing process. The morphable object under the influence of pressure and spring forces continues to morph until it collides with surrounding objects. When the morphable object collides with the surrounding objects, the mass points that interfere with the surrounding objects need to be identified and retracted. The prevention of interference between the morphable object and the surrounding objects is modeled as a two-step process. In the first step, for every constraint object that collides with the morphable object, all the colliding pairs of facets on both the objects are identified. In the second step, for every facet on the morphable object that interferes with any of the constraint objects, the mass points that interfere the constraint objects are identified and retracted. The mass points are retracted back to coincide with the surface of the constraint objects. Their velocity is also adjusted so as to prevent subsequent interference during the morphing process.

\subsection*{3.4.2.1 Proposed Surface Collision Detection Algorithm}

Several collision detection algorithms that work with non-convex mesh-based models have been proposed in the recent past [74, 75, 76, 77]. These collision detection algorithms employ temporal and spatial coherency to significantly speed-up the collision detection. Some of the popular collision detection algorithms are VCollide (an \(N\)-body collision detection system containing the Rapid collision detection engine [76]), Proximity Query Package (PQP) [75], Solid [78], and Swift++ [74]. Some collision detection algorithms like Proximity Query Package (PQP) [75] can additionally report all the pair of colliding facets. All these
collision detection algorithms preprocess the input mesh to generate additional information that is exploited during the collision detection process. Depending on the collision detection algorithm, the preprocessing step generates axis-aligned or object-aligned bounding boxes, and a hierarchical K-d tree [79] of such boxes. These collision detection algorithms assume the colliding objects to be rigid and only allow affine transformations. If the geometry of an object is modified, then that object must be preprocessed again. Hence, use of these collision detection algorithms for interference detection requires the preprocessing of the morphable object at every iteration of the morphing process. Invoking the preprocessing at every step makes the collision detection extremely expensive and time consuming. We propose a simple surface collision detection algorithm using axis-aligned bounding boxes (AABB) that requires minimal preprocessing and thus is ideally suited for the current problem.

The proposed surface collision detection algorithm can work with arbitrary triangulations; i.e. it does not make any assumption about the triangular mesh. It can perform Yes/No type queries and additionally can identify all pairs of colliding facets between two objects. It makes use of the fast triangle-box [46] and triangle-triangle [80] intersection algorithms. Consider two objects \(A\) and \(B\) between which interference checking is to be performed. Suppose the object \(A\) is the morphable object, and object \(B\) is the constraint object. Suppose that the objects \(A\) and \(B\) have \(N_{A}\) and \(N_{B}\) facets respectively. The following is the sequence of operations performed to identify all the pairs of colliding facets at each iteration.
1. The axis-aligned bounding box of the object \(A\) is determined. This steps takes \(\Theta\left(N_{A}\right)\) time. Since the object \(B\) is rigid, its bounding box need not be determined at every iteration.
2. The box-box overlap computation between bounding boxes of \(A\) and \(B\) is performed. Since the boxes are axis-aligned, the box-box overlap computation requires six comparisons in the worst possible case. Let \(\left(x_{A 1}, y_{A 1}, z_{A 1}\right)\) and \(\left(x_{A 2}, y_{A 2}, z_{A 2}\right)\) be the coordinates of the two diagonally opposite corners of the bounding box of object \(A\)
such that \(x_{A 1}<x_{A 2}, y_{A 1}<y_{A 2}\), and \(z_{A 1}<z_{A 2}\). Similarly let \(\left(x_{B 1}, y_{B 1}, z_{B 1}\right)\) and \(\left(x_{B 2}, y_{B 2}, z_{B 2}\right)\) be the coordinates for the object \(B\). Then the objects \(A\) and \(B\) do not collide if any of the following six comparisons are satisfied. The six comparisons are \(x_{A 2} \leq x_{B 1}, y_{A 2} \leq y_{B 1}, z_{A 2} \leq z_{B 1}, x_{A 1} \geq x_{B 2}, y_{A 1} \geq y_{B 2}\), and \(z_{A 1} \geq z_{B 2}\). The objects potentially collide if none of the six comparisons are true. This steps takes constant time.
3. If the bounding boxes of the objects \(A\) and \(B\) collide, then the collision check between each facet of object \(A\) and the bounding box of \(B\) (and vice versa) is performed. The triangle-box intersection [46] test is used to detect all the colliding facets. The computational complexity of this step is \(\Theta\left(N_{A}+N_{B}\right)\), where each operation amounts to a triangle-box intersection test. If at least one facet on object \(A\) is found to collide with the bounding box of the object \(B\), then the two objects potentially collide. In this step, the list of all facets on each object that collide with the bounding box of the other object is built. These facets lie in the intersection of the bounding boxes of both objects \(A\) and \(B\). Only the facets contained in this potential intersection list need to be considered to determine the interference between the two objects.
4. For every facet of object \(A\) in the potential intersection list, a triangle-triangle intersection test [80] is performed with all the facets of object \(B\) in the potential intersection list. It is possible that the bounding boxes of the two objects collide, but the actual objects do not. If at least one pair of colliding facets is found, then the objects \(A\) and \(B\) collide. If a Yes/No type query is performed, the process may be terminated as soon as the first pair of colliding facets is found. If all the pairs of colliding facets is desired, the process is simply continued until all facets have been checked. The worst case computational complexity of this step is \(\Theta\left(N_{A} N_{B}\right)\), where each operation amounts to a triangle-triangle intersection test.

All the pairs of colliding facets can be determined by just performing step 4 of the above algorithm. The first three steps are designed to reduce the number of triangle-
triangle intersection tests (most expensive operation) needed to check interference detection. The above implementation was found to be faster than an established collision detection algorithm like PQP for the case of morphing. The PQP was slower primarily because preprocessing was performed at every step of the morphing process. It should be noted that the collision detection algorithm discussed above will be easily outperformed by specialized algorithms like Swift++ and PQP if the colliding objects are assumed to be rigid and only affine transformations are performed.

\subsection*{3.4.2.2 Collision Response}

Once all the pairs of colliding facets are identified, the vertices on the morphable object that penetrate the constraint object are determined and retracted. Let \(\vec{X}\) be the position vector of the vertex to be checked for interference, \(\vec{P}\) be any vertex on the colliding facet of the constraint object, and \(\hat{n}\) be the outward unit normal of the colliding facet on the constraint object, then the mass point represented by vertex \(\vec{X}\) penetrates the surface of the constraint object if the following condition is satisfied.
\[
\begin{equation*}
(\vec{X}-\vec{P}) \cdot \hat{n}<0 \tag{3.10}
\end{equation*}
\]

The vertices \(\vec{P}\) and \(\vec{X}\) and the normal vector \(\hat{n}\) are pictorially depicted in Figure 3.8.


Figure 3.8: Interference of nodes

It should be noted that the above test is valid only if the two facets are known to be colliding. The above test may give false positives in the case the facets do not collide. Once all the vertices that interfere with the surface of the constraint object are identified,
the next step is to retract those vertices to remove any interference.
Retracted Position: The nodes are retracted right onto the surface of the constraint object as shown in Figure 3.9. Their velocity is updated to prevent subsequent interference. Apart from adjusting the velocity, an additional reaction force is applied to the mass points to prevent the vertices from future interference with the constraint objects. Hence, this step also involves updating the resultant force on the mass points that interfere with the constraint objects. The collision check is performed at each iteration of the morphing process, and the position, velocity, and contact force is updated for each mass point that interferes with the constraint objects. The vertex is retracted from the position \(X\left(t_{\text {collision }}\right)\) which is inside the constraint object to the position \(X^{\prime}\left(t_{\text {collision }}\right)\) which is on the surface of the constraint object.


Figure 3.9: Retraction of the mass points onto the surface of the constraint object

The retraction distance \(d\) is shown in Figure 3.10. In Figure 3.10, the initial and final position of the vertex is denoted by \(P\) and \(P^{\prime}\) respectively. The outward unit normal is \(\hat{n}\). Let one of the vertices on the constraint facet be \(Q\). Then the distance \(d\) is the length of the projection of the vector \(\vec{P}-\vec{Q}\) along the unit normal \(\hat{n}\). Thus the distance \(d\) is given by Equation 3.11. The final location \(P^{\prime}\) of \(P\) is given by Equation 3.12.
\[
\begin{equation*}
d=|(\vec{P}-\vec{Q}) \times \hat{n}| \tag{3.11}
\end{equation*}
\]
\[
\begin{equation*}
P^{\prime}=P+d \hat{n} \tag{3.12}
\end{equation*}
\]


Figure 3.10: Retraction distance

Velocity of the mass points after collision: The velocity after the collision depends on the physical collision model simulated. For the purpose of morphing, a purely inelastic collision model is used. The velocity \(V\) of the node is resolved into the tangential component \(V_{t}\) and the normal component \(V_{n}\). In perfectly inelastic collision, there is complete loss of kinetic energy, hence the normal velocity \(V_{n}\) is set to zero after the collision. The tangential velocity \(V_{t}\) is left unchanged. Thus, the mass points after the collision continue to move in a direction tangential to the surface of the constraint object. Thus, this modification to the velocity allows the morphable object to conform to the contour of the constraint objects without colliding.

Updated force on the mass points: If the force acting on the mass points is left unchanged after the collision, it will continue to have non-zero acceleration along the normal to the surface of the constraint object. When the acceleration, velocity, and position are updated in the next iteration, the mass points will again interfere with the constraint object. To reduce the probability of this event, a reaction force is added that counters the normal component of the resultant force due to the pressure and the springs. Let the resultant of
the pressure and spring force be \(\vec{F}\), and let the unit normal to the colliding facet of the constraint object be \(\hat{n}\). Then, the updated force on the mass point is given by Equation 3.13.
\[
\begin{equation*}
\vec{F}_{\text {updated }}=\vec{F}-(\vec{F} \cdot \hat{n}) \hat{n} \tag{3.13}
\end{equation*}
\]

Thus, the force update is similar to the velocity update. The tangential component remains unchanged and the normal component is assigned a value of zero.

\subsection*{3.4.3 Flowchart of the Morphing Process}

The complete flowchart of the morphing process is given in Figure 3.11.

\subsection*{3.5 Demonstration of the Morphing Algorithm}

We present two case studies to demonstrate the morphing process. The following simulation parameters are used for reporting the results.
- Spring constant \(\left(k_{s}\right)=755 \mathrm{~N} / \mathrm{m}\)
- Damping constant \(\left(k_{d}\right)=35 \mathrm{~N}-\mathrm{s} / \mathrm{m}\)
- Time step \((\Delta t)=0.01\) seconds
- Mass \(=1.0 \mathrm{~kg}\)
- Change in pressure \(=0.5 \mathrm{~N} / \mathrm{m}^{2}\)

\subsection*{3.5.1 Sphere inside an arbitrary enclosure}

In the first case study, a sphere is contained inside an arbitrary enclosure and is morphed to fill the space inside it. This case study demonstrates the morphing process when the morphable object is contained inside an enclosure. The initial configuration is shown in Figure 3.12. The final configuration is shown in Figure 3.13. The sphere and the


Figure 3.11: Flowchart of the Morphing Process
outer enclosure have approximately 500 and 5,000 facets respectively. The total time taken by the morphing algorithm is approximately 1 minute.


Figure 3.12: Case study 1: initial configuration

\subsection*{3.5.2 Multiple constraint objects}

The second case study demonstrates the morphing process in the presence of multiple constraint objects. The initial and final configurations are shown in Figures 3.14 and 3.15 respectively. There are six constraint objects. The morphable object and the constraint objects have approximately 5,000 to 15,000 facets each. The constraint objects are nonconvex and free-form. The constraint objects need not be manifolds. Two of the constraint objects are planar sheet bodies with one of them only partially obstructing the morphable object thus forcing the morphable object to follow the edges of the sheet body. The total time taken by the morphing algorithm is approximately 25 minutes.

The morphed object is shown in Figures 3.16 and 3.17 respectively. It is evident from Figures 3.16 and 3.17 that the surface of the morphed object is not smooth and has


Figure 3.13: Case study 1: final configuration


Figure 3.14: Case study 2: initial configuration


Figure 3.15: Case study 2: final configuration
many undesirable artifacts. To obtain a smooth geometry without any noise and kinks in the surface, a feature-preserving smoothing algorithm [69] is applied as a post-processing step. The smoothing is desired from a manufacturability and aesthetics stand-point. The morphed object after denoising is shown in Figures 3.18 and 3.19.

It should be noted that the proposed shape morphing method does not give the final geometry of the component, rather it gives an accurate representation of the boundary of the component. The desired features such as mount points etc. have to be added to obtain the final geometry.

\subsection*{3.6 Conclusion and Future Work}

In this paper, a physically based shape morphing algorithm suitable for layout design is proposed. The proposed morphing algorithm is based on a mass-spring model and the morphing process is governed by physical equations. We have proposed several modifications to the mass-spring model to make it faster and more efficient. A general purpose surface collision detection algorithm is also proposed. The proposed surface collision de-


Figure 3.16: Case study 2: morphed object


Figure 3.17: Case study 2: morphed object


Figure 3.18: Case study 2: morphed object after denoising


Figure 3.19: Case study 2: morphed object after denoising
tection algorithm does not require any pre-processing and is ideally suited for morphing applications. An algorithm to generate the mass-spring model from an arbitrary manifold is also proposed. The proposed morphing algorithm incorporates volume control and does not require any user interaction during the morphing process, and thus is fully automated. The proposed algorithm therefore can be easily integrated with an automatic layout design process. The morphing method discussed in this paper is ideally suited for generating and optimizing the geometry of components which do not have any restriction on their shape and require volume control.

There is significant scope for improvement of the proposed morphing algorithm. The mesh quality deteriorates during the morphing process, especially in cases where significant changes occur in the geometry. To obtain a geometry that accurately captures the surface profile of constraint objects, a very fine mesh is needed. This would however increase the computational time required for morphing. It is therefore suggested to incorporate adaptive mesh refinement during the morphing process. To obtain a correct manifold geometry, the morphing algorithm requires the use of very small time steps. The inaccuracy in the physical mass-spring model is primarily due to the first order approximation used to compute the motion of mass points. Use of a higher order method will reduce the accumulated error thus allowing larger time steps. The proposed algorithm is capable of performing multibody morphing but this case has not been demonstrated in this paper. The multi-body morphing requires synchronization of time steps and coordinated update of position and force terms. The use of the proposed morphing algorithm for multi-body morphing shall be investigated in the future.

\section*{Chapter 4}

\section*{AMGA2: IMPROVING THE PERFORMANCE OF THE}

\section*{ARCHIVE-BASED MICRO} GENETIC ALGORITHM FOR MULTI-OBJECTIVE OPTIMIZATION

\subsection*{4.1 Introduction}

Multi-objective optimization has become main-stream in recent years and several algorithms to solve multi-objective optimization problems have been suggested. The use of multi-objective optimization in industry has been accelerated by the availability of faster processing units and the computational analysis models for various engineering problems and disciplines. Very often, the computation analysis models under consideration are suf-
ficiently difficult and cannot be solved to optimality by most classical (gradient-based) optimization algorithms. The difficulty associated with most optimization problems can be attributed to factors such as simultaneous optimization of multiple objectives, multi-modal function profiles, non-convex and discontinuous search spaces, and mixed representation of optimization variables. To alleviate the difficulties faced by the gradient-based optimization algorithms, several non-traditional optimization algorithms that can handle such difficulties effectively have been proposed in the recent past. Additionally, most non-traditional algorithms do not impose any extraneous conditions on the optimization problem such as the convexity of the objectives and constraints and existence of the derivatives. Evolutionary algorithms are one of the non-traditional methods that have seen wide acceptability because these algorithms can handle the difficulties outlined above with relative ease. Furthermore, most engineering problems are NP-hard [29] and therefore a quick computation of approximate solutions is often desirable. Evolutionary algorithms (EAs) are adaptive search techniques inspired from nature and their working principle is based on the Darwin's theory of the survival-of-the-fittest [32,33, 34]. The adaptive nature of EAs can be exploited to design optimization algorithms by designing suitable variation operators and an appropriate fitness function. During the optimization process, the fitness is adapted as the evolution proceeds. The Genetic Algorithm (GA) [35, 36, 81, 82, 83] is one of the evolutionary techniques that has been successfully used as an optimization tool. Typically a GA works with a population (a set of solutions) instead of a single solution (individual). This property of a GA makes it an ideal candidate for solving multi-objective optimization problems where the outcome (in most cases) is a set of solutions rather than a single solution. The population approach of a GA also makes it resilient to premature convergence, thereby making it a powerful tool for handling highly non-linear and multi-modal functions.

In this paper, an improved version of the Archive-based Micro Genetic Algorithm [8] for constrained multi-objective optimization is proposed. The improved version is referred to as AMGA2 in this paper. The design of AMGA2 is primarily motivated by the fact that in most optimization scenarios, almost the entire time is spent by the analysis routines and
the user does not have the computational resources to perform a large number of function evaluations. The actual time spent by the optimization algorithm in performing selection, crossover, mutation, and diversity assessment is often negligible compared to the total simulation time. Another important guiding principle that has shaped the design of AMGA2 is the fact that, the user is often satisfied if a good enough non-dominated solution set is obtained. In most engineering applications, once a reasonable solution quality is achieved, a global optimizer (e.g. a genetic algorithm) is replaced by a local optimizer (e.g. a gradientbased method) which has a faster local convergence rate and can generate solutions with higher accuracy. Often the desired function value is not known; in such cases, the global optimizer is terminated when the convergence rate falls below a certain threshold, or after a prescribed number of function evaluations is reached, or there is no noticeable improvement in the obtained solution set. Another motivating factor behind the design of AMGA2 is that most optimization users may not be able or willing to accurately fine-tune the parameters of an optimization algorithm. The AMGA2 therefore uses a set of rules (heuristics) to choose suitable tuning parameters automatically. The only optimization tuning parameters that are exposed to the user are the maximum allowed number of function evaluations and the number of solutions desired at the end of the optimization. The remaining tuning parameters are chosen by the AMGA2 based on the problem size, the number of function evaluations allowed, and the desired number of solutions. The AMGA2 also benefits from the existing literature in that it borrows several concepts like formulation for crossover, mutation, two-tier fitness assignment mechanism, ranking strategy, preservation of elite and diverse solutions etc. from the existing evolutionary optimization algorithms. We also propose several modifications and improvements to the genetic variation operators and diversity assessment techniques discussed in the literature. As is the hallmark of evolutionary algorithms, the AMGA2 does not impose any extraneous condition on the optimization problem. The AMGA2 can also be used to solve single-objective optimization problems without any modification; however it might exhibit poor performance on such problems. It can also be used to solve constraint satisfaction problems where the aim is to find a feasi-
ble solution to the optimization problem. The AMGA2 also obeys one of the fundamental constraints of solving any real engineering optimization problem. It does not require or use the value of the objectives if a solution is infeasible. Thus, it is suggested to evaluate the constraints before the objectives, and if the constraints are violated, the objectives need not be evaluated. The formal statement for the optimization problem that the AMGA2 attempts to solve is stated in Equation 4.1.
\[
\begin{array}{ll}
\text { Minimize } & \left(f_{1}(\mathbf{x}), f_{2}(\mathbf{x}), \ldots, f_{M}(\mathbf{x})\right) \\
\text { Subject to } & g_{j}(\mathbf{x}) \leq 0, \quad j=1,2, \ldots, J  \tag{4.1}\\
& h_{k}(\mathbf{x})=0, \quad k=1,2, \ldots, K \\
& x_{i}^{(L)} \leq x_{i} \leq x_{i}^{(U)}, \quad i=1,2, \ldots, N
\end{array}
\]

Unless otherwise stated, the concept of Pareto domination [84] is used for comparing two solutions. The usual definition of Pareto-domination [84] that is used in the present context is as follows: A feasible solution a dominates another feasible solution b for a \(M\)-objective minimization problem, if the following conditions are met:
1. \(f_{i}^{\mathrm{a}} \leq f_{i}^{\mathrm{b}}\) for all \(i=1,2, \ldots, M\),
2. \(f_{i}^{\mathrm{a}}<f_{i}^{\mathrm{b}}\) for at least one \(i \in\{1, M\}\).

Additionally, a feasible solution is always preferred to (dominates) an infeasible solution. If two solutions being compared are infeasible, then the solution with a smaller value for the overall constraint violation is preferred. The overall constraint violation for a solution \(s\) is given by Equation 4.2.
\[
\begin{equation*}
\mathrm{CV}(\mathbf{s})=\sum_{j=1}^{J}\left\langle g_{j}(\mathbf{s})\right\rangle+\sum_{k=1}^{K}\left|h_{k}(\mathbf{s})\right|, \tag{4.2}
\end{equation*}
\]

The remainder of this paper is organized as follows. In section II, a brief survey of the relevant literature on multi-objective optimization is presented. Section III contains the description of the proposed optimization algorithm. In section IV, benchmark problems,
algorithm tuning parameters, and simulation results are presented. Section V discusses the obtained simulation results and section VI concludes this study. The ideas for future research and possible future improvements to AMGA2 are proposed in section VII.

\subsection*{4.2 Current State-of-the-art}

Much of the current research in the development of optimization algorithms focuses on achieving faster convergence on uni-modal problems, and efficiently and reliably solving multi-modal problems. To gain a better understanding of different optimization algorithms and to understand the reasons behind their design, it is important to study the nature and types of difficulties encountered while solving most optimization problems. The difficulties associated with solving an optimization problem can be attributed to factors such as the type of search space (discontinuous, discrete, non-convex), non-linearity and multi-modality of objectives and constraints, ratio of feasible to infeasible search space, and the size of the optimization problem (number of variables, objectives, and constraints). In the recent past, evolutionary algorithms have been used with an increased degree of success to solve multiobjective and multi-modal problems. The evolutionary optimization algorithms rely on finding the globally optimal solutions by chance, and incorporate methods (heuristics) to reduce the possibility of getting trapped inside a locally optimal basin. Some of the notable efforts in designing multi-objective evolutionary algorithms (MOEAs) are Strength Pareto Evolutionary Algorithm (SPEA2) [85], Pareto-Envelope Based Selection Algorithm (PESAII) [86], Non-dominated Sorting Genetic Algorithm (NSGA-II) [87], Neighborhood Cultivation Genetic Algorithm (NCGA) [88], Intelligent Multi-Objective Evolutionary Algorithm (IMOEA) [89], \(\epsilon\)-Multi-objective Evolutionary Algorithm ( \(\epsilon\)-MOEA) [90], Omni-Optimizer (OmniOpt) [91], and Fast Pareto Genetic Algorithm (FPGA) [92]. A historical and comprehensive survey of MOEAs can be found in [81]. A survey of the current evolutionary algorithms proposed for constrained multi-objective optimization reveals the following key concepts employed in their design.
- Population Approach: Recent trends in the use of population focus on aspects such as dynamic population sizing used in the FastPGA [92], use of a very small population size (a micro genetic algorithm) [93], and the use of an external archive [85] that stores the best found approximation to the Pareto-optimal front. Use of a dynamic or a very small population size helps to reduce the number of function evaluations required to obtain the desired convergence whilst an external archive can store a large number of non-dominated solutions to accurately approximate the Pareto-optimal front. It should be possible to combine both these approaches into a single optimization algorithm.
- Selection Mechanism: The two most popular techniques to perform the selection operation are the non-dominated sorting [87] and the strength Pareto approach [85]. An improvement to the strength Pareto approach is proposed in FastPGA which takes into account not only the number of solutions an individual dominates, but also the number of solutions it is dominated by. Further, since the domination principle alone cannot classify all the solutions, multi-level fitness mechanisms are used for the classification. Generally, the primary fitness is based on the domination level, and the secondary fitness is based on some measure of a solution's diversity. Emphasizing the domination level (rank) over the diversity may not always be a good idea - especially for multi-modal problems.
- Diversity Assessment: There exist a large number of methods to assess the diversity of a solution. Some of these techniques that have been successfully used to assess the diversity are the fitness-sharing [94, 95], crowding distance metric [87, 91, 96], K-mean clustering [97], \(\epsilon\)-domination [90], cell-based (hyper-grid) methods [86, 98, 99], and fast pruning of crowded solutions using efficient nearest neighbor search [100, 101]. A metric like crowding distance can be used to assign a quantitative measure of diversity to a solution, whereas methods based on pruning of crowded solutions cannot. An evolutionary algorithm may require more than one measure of diversity depending
upon the context. We propose improvements to the crowding distance metric as well as to the pruning method based on efficient nearest neighbor search and use them in the AMGA2 for assessing the diversity of a solution.
- Variation Operators: Some of the desired characteristics of a genetic variation operator are: parent-centric property, self-adaptivity, invariance to affine transformations of the search space, and disruptiveness to impart random behavior and resilience to premature convergence. The crossover operators such as unimodal normal distribution crossover [102], simulated binary crossover [44], and parent-centric crossover [103] are designed to work with real variables and have some kind of parent-centric property. These crossover operators however lack self-adaptability (cannot automatically adjust their step size depending on the problem and the distribution of the population). The differential evolution (DE) [104, 105, 106] operator exhibits excellent self-adaptive characteristics and in our tests was found to consistently outperform other crossover operators. The DE operator has been combined with traditional genetic algorithms [107] in the past. Traditionally the parents for use with the DE are chosen randomly [106]. The choice of the parents used with the DE operator has a significant impact on the performance of the algorithm. We propose a new selection strategy to suitably choose the parents for use with the DE which is the crossover operator used in AMGA2. The AMGA2 uses the modified formulation for the polynomial mutation proposed in [91] for its high disruptiveness. The probability of mutation is adapted using the method proposed in [108].
- Knowledge Integration: We use the phrase Knowledge Integration to refer to borrowing and incorporating concepts and ideas from domains external to the evolutionary optimization literature. Customizing optimization algorithms by developing specialized representation for the genotypes is an effective method to exploit the problem information. Such customizations however are problem specific and cannot be used in a general purpose optimizer. Hybridization of evolutionary algorithms by coupling
mathematical programming techniques \([109,110,111,112]\) is a widely used method to improve the performance of an optimization algorithm. Such algorithms are often referred to as memetic algorithms in the optimization literature. Other examples of knowledge integration are the use of Covariance Matrix Adaptation to adapt the mutation step of an evolution strategy [42], incorporating model building (meta-modeling, estimation of distribution) as part of the optimization process [113, 114] , and biasing the search process using human decision makers (interactive evolutionary algorithms) [115, 116].

A more comprehensive survey and references on evolutionary optimization can be found in [117] and [82] respectively. The Archive-based Micro Genetic Algorithm version 2 (AMGA2) proposed in this paper attempts to combine the best features and practices found in the literature and incorporates improvements to existing selection, diversity assessment, and genetic variation operators.

\subsection*{4.3 Description of the AMGA2}

The AMGA2 proposed in this paper is a significant improvement over the original Archive-based Micro Genetic Algorithm [8], but shares the same basic flowchart for the generation scheme. The generation scheme of AMGA2 can be classified as generational since during a particular iteration (generation), only the solutions created before that iteration take part in the selection process. The AMGA2 works with a very small population size and maintains a large external archive of good solutions obtained. Using an external archive that stores a large number of solutions provides useful information about the search space as well as tends to generate a large number of Pareto points at the end of the simulation. At every iteration, a small number of solutions are created using the genetic variation operators. The newly created solutions are then used to update the archive. The strategy used to update the archive relies on the domination level and the diversity of the solutions, and the current size of the archive, and is based on the non-dominated sorting concept borrowed
from NSGA-II. This process is repeated until the allowed number of function evaluations is exhausted.

AMGA2 uses a two-tier fitness assignment mechanism; the primary fitness is the domination rank of the solution in the population, and the secondary fitness is the diversity measure of the solution in the population. Both the rank and the diversity measure are used by AMGA2 whilst updating the archive and during the creation of the parent population. During the archive update, AMGA2 performs iterative pruning of the most crowded solutions using the efficient nearest neighbor search strategy. During the creation of the parent population however, a numerical measure of the diversity is needed, and the crowding distance metric is used for this purpose. The mating pool for AMGA2 is created using the parent population as well as the external archive. The AMGA2 uses the concept of primary parent and auxiliary parents. The primary parents comprise the parent population, whereas the auxiliary parents are selected randomly from the archive. During the initial stages of the search, most solutions present in the archive are dominated and therefore very few solutions from the archive are included in the parent population. Similarly, during the later stages of the search, most solutions present in the archive are non-dominated, and only few solutions which are very diverse (have large gaps around them) are included in the parent population. This reduces the number of function evaluations required to find a good approximation to the Pareto-optimal front. The design of AMGA2 is independent of the solution representation, and thus it can work with almost any kind of optimization variables so long as suitable genetic variation operators are provided to it. The pseudo-code of the AMGA2 is as follows.

\section*{The AMGA2 procedure:}

\section*{Begin}

2 Generate initial population.
3 Evaluate initial population.
4 Update the archive (using the initial population).
5 repeat
```

6 Create parent population from the archive.
7 Create mating pool from the parent population and the archive.
8 Create offspring population from the mating pool by crossover
followed by mutation.
Evaluate the offspring population.
Update the archive (using the off-spring population).
until (termination)
Report desired number of solutions from the archive.
13 End

```

The above mentioned pseudo-code of AMGA2 clearly separates all the conceptual steps of the algorithm. The pseudo-code can also be viewed as a functional decomposition of the AMGA2, and every step of the AMGA2 can be designed, fine-tuned, and benchmarked separately. It facilitates using the best concepts and ideas proposed in literature to accomplish each of the tasks. It also encapsulates the working of multi-objective optimization algorithms like NSGA-II and SPEA2 in that, both of these algorithms can be constructed by choosing suitable techniques for each step. We now discuss each step in the pseudo-code of the AMGA2 in more detail.

\subsection*{4.3.1 Generation of the initial population (step 2)}

The initial population \(\left(P_{0}\right)\) can be generated in multiple ways. It can be either generated randomly such that all the variables are inside the search space or can be uniformly sampled. We choose to create the initial population using Latin hypercube (LH) sampling [118] coupled with unbiased Knuth shuffling since it gives a good overall random (unbiased) distribution of the population in the genotypic (variable) space and does not require any objective or constraint function evaluation. The LH sampling process generates solutions randomly inside identical sized bins which span the entire search space. Let the size of the initial population be \(N_{0}\). Let the lower bound of variable \(i\) be \(l_{i}\) and the upper bound be \(u_{i}\). To generate a LH sample, the variable range is divided into \(N_{0}\) equal segments of size
\(\frac{u_{i}-l_{i}}{N_{0}}\) each, and a real random number is generated in every segment. The random number generator used must generate the random numbers with uniform probability distribution. Then a random permutation of integers from 1 to \(N_{0}\) is generated and the individual with index \(i\) is assigned a value located at \(\pi(i)^{t h}\) position in the permutation. This process is repeated for all the variables. This ensures that the resultant population spans the entire genotypic space, is sufficiently random and is free from any bias. In order to generate a random permutation, unbiased Knuth shuffling is iteratively applied for each variable on the same sequence. Omni-optimizer [91] also uses the same strategy to generate the initial population.

In step 3, the initial population is evaluated. Initially, the archive is empty, and in the step 4 of AMGA2, the initial population is simply copied to the archive. Steps 6 through 10 form the main iteration loop of the AMGA2.

\subsection*{4.3.2 Creation of the parent population from the archive (step 6)}

This is an important step of the AMGA2 since the size and choice of the parent population has a significant impact on the performance of the algorithm. To illustrate a key concept used in the generation of parent population from the archive, we refer to the two-objective problem ZDT4 [36]. This problem has 100 distinct Pareto-optimal fronts out of which only one is globally Pareto-optimal. The plot of the objective space for the ZDT4 function after the first generation is shown in Figure 4.1. Only 6 solutions out of the 100 belong to the first rank. In a given population (archive), it may not be worthwhile to include every solution in the parent population. Significant number of function evaluations can be saved if only the solutions in the first rank are chosen for the parent population. It is thus noted that a reduction in number of function evaluations can be obtained if a small number of better solutions are included in the parent population. Furthermore, during the later stages of evolution, when most solutions belong to rank 1, only the most diverse solutions should be included in the parent population so that the exploration of the region around those solutions is facilitated. In the case of AMGA2, the size of the parent population
\((\approx 4)\) is much smaller than the size of the archive \((\approx 100)\). Unless the problem is highly multi-modal, the parent population almost exclusively contains solutions belonging to rank 1. Also, the size of the parent population must be greater than or equal to the number of objectives for AMGA2. The following procedure to generate the parent population is suggested.


Figure 4.1: Objective space plot for the zdt4 test problem

The parent population is created from the archive. Let the desired size of the parent population be \(N_{p}\). All the solutions in the archive belonging to rank 1 are identified (call this the set \(S\) and let its size be \(N_{s}\) ). If \(N_{s} \geq N_{p}\), then the parent population can be generated from the set \(S\). If \(N_{s}<N_{p}\), then solutions belonging to rank 2 are extracted from the archive and added to the set \(S\). This process is repeated until \(N_{s} \geq N_{p}\). The solutions in the set \(S\) that have the minimum value for any objective function are extracted and copied to the parent population. The number of such solutions will always be less than or equal to the number of objectives. Let the number of such solutions be \(N_{e}\). Then, \(N_{p}-N_{e}\) slots in the parent population remain vacant. To fill these slots, the crowding distance metric is
used. Since the set \(S\) may potentially contain solutions belonging to more than one rank, the crowding distance measure as proposed in NSGA-II [87] cannot be used. In the Omnioptimizer [91], a modification to the crowding distance assignment mechanism is proposed that works with the solutions belonging to multiple ranks. In the original AMGA [8], two improvements to the crowding distance measure were suggested which are described next with a slight modification. We first present the modification.

In the original AMGA, the extreme solutions were assigned a diversity of \(\infty\). With the AMGA2, the extreme solutions belonging to rank 1 are already extracted from the set \(S\) and included in the parent population. Hence, the extreme solutions in the remaining set \(S\) are not the true extrema of the current population, and hence are assigned the crowding measure based on their nearest neighbors corresponding to every objective. This modification (deliberate inclusion of all the extreme solutions in the parent population) is done because in the original AMGA, each solution took part in exactly one binary tournament selection, and the binary tournament selection operator could compare two extreme solutions (with \(\infty\) diversity) and may inadvertently remove one of those extreme solutions from the mating pool (which would amount to not exploring one of the extremes of the Pareto-optimal front). The modification proposed in this paper guarantees the inclusion of extreme solutions in the parent population and the mating pool. The improvements to the crowding distance measure are discussed next.

Consider Figure 4.2. Suppose that solution \(B\) has left and right neighbors \(A\) and \(C\) respectively. The usual formulation (proposed in NSGA-II) for the crowding distance (CD) measure gives \(\mathrm{CD}(B)=l_{1}+r_{1}+l_{2}+r_{2}\). Larger the value of \(\mathrm{CD}(B)\), the more diverse is the solution. It should be noted that \(\operatorname{CD}(B)\) depends only on the location of \(A\) and \(C\) and does not depend on the location of \(B . B\) can be anywhere inside the bounding box defined by its neighbors \(A\) and \(C\). Ideally, it is desirable that solution \(B\) lies at the center of the bounding box for good diversity. It is also desirable that larger the dimension of the bounding box, larger the value of \(\mathrm{CD}(B)\). We thus need a formulation for CD which is maximized if the dimension of the bounding box is maximized, or for a given size of the bounding box, CD


Figure 4.2: Depiction of crowding distance computation
is maximized if the solution \(B\) lies at the center of its bounding box. We thus propose the modified formulation for crowding distance computation as given in Equation 4.3.
\[
\begin{equation*}
\mathrm{CD}(B)=\sum_{i=1}^{M} l_{i} r_{i} \tag{4.3}
\end{equation*}
\]

In Equation 4.3, \(M\) is the number of objectives. The product \(l_{i} r_{i}\) is maximized for a constant size of the bounding box if \(l_{i}=r_{i}\); i.e. if \(B\) lies at the center of its neighbors. Also, as the size of the bounding box grows, the value of the product increases. Thus, the formulation given by Equation 4.3 more accurately captures the diversity of \(B\). The extreme solutions will have only left or right neighbor, but not both. If a solution has only a left neighbor for a specific objective, then the quantity \(l_{i}^{2}\) corresponding to that objective is added; similarly if only the right neighbor is present, the quantity \(r_{i}^{2}\) is added.

The above formulation has a potential shortcoming. What if there is a solution \(B^{\prime}\) identical to \(B\). In that case, the formulation described by Equation 4.3 would give a value of zero for diversity for both \(B\) and \(B^{\prime}\) since distance to the nearest neighbor is zero in all directions (since at least one of \(l_{i}\) or \(r_{i}\) would evaluate to zero). The original formulation
(as proposed in NSGA-II) would give non-negative values to both \(B\) and \(B^{\prime}\). Depending upon their actual position in the sorted array, the crowding distance for solutions \(B\) and \(B^{\prime}\) would evaluate to one of the values from \(\left(l_{1}+l_{2}, l_{1}+r_{2}, r_{1}+l_{1}, r_{1}+r_{2}\right)\). This situation also is not desirable, since the obtained values do not accurately reflect the diversity of the two solutions. We therefore propose further modification which can be applied to the original as well as proposed crowding distance formulation. We suggest that all (but one) copies of an identical solutions be removed and assigned a value of zero for crowding distance before applying the formulation given in Equation 4.3. All identical copies can be removed in \(\Theta(N \log N)\) time if there are \(N\) solutions in a non-dominated set. Thus the proposed refinements do not alter the complexity of the crowding distance computation proposed in [87].

\subsection*{4.3.3 Creation of the mating pool (step 7)}

In the original AMGA [8], the mating pool was created from the parent population using the binary tournament selection. The AMGA2 does not use binary tournament selection and the mating pool comprises solutions from the parent population as well as the archive. The creation of the mating pool must take into account the crossover operator used with the optimization algorithm. We use the Differential Evolution (DE) as the crossover operator. The DE crossover operator takes 4 (1 primary and 3 auxiliary) parents and creates one offspring. The solutions in the parent population comprise the primary parents. For each primary (index) parent, three solutions from the archive are chosen randomly such that they are not identical to the index parent and are mutually different. In the literature [106], all the four parent solutions are chosen randomly, however with AMGA2, only auxiliary parents are chosen randomly. In our tests, it was found that choosing the index parents based on the domination level and diversity consistently outperformed the random selection. This is expected since the DE is a self-adaptive crossover operator, and enabling it to inherit from a potentially better solution increases the probability of discovering better solutions.

\subsection*{4.3.4 Genetic variation operators (step 8)}

The mating pool contains the index parents and the auxiliary parents. If the size of the parent population is \(N_{p}\), then the number of auxiliary parents is \(3 N_{p}\). To create the offspring population, the crossover operator is applied to the mating pool followed by the mutation of the offspring solutions. There exist several crossover operators designed to work with real variables \([44,103,102]\). The performance of the simulated binary crossover (SBX) [44], parent-centric crossover (PCX) [103], and the DE operator [106] was compared and it was observed that the DE operator consistently outperformed the other two. The DE operator has the additional advantage of not requiring a distribution index and it is self-adaptive in that the step size is automatically adjusted depending upon the distribution of the solutions in the search space. The following formulation (taken from [106]) for the DE operator is used with the AMGA2. Let the index parent be \(p_{i}\), and let the auxiliary parents be \(a_{1}, a_{2}\), and \(a_{3}\). Let the number of variables be \(N\), and let \(j_{r}\) be a random number (uniformly distributed) between 1 and \(N\). The DE operator uses two tuning parameters \(F=0.1\) and \(C R=0.5\). Let \(u_{j}\) be a uniformly distributed random number in \([0,1]\). Let the offspring solution be \(o\) and the subscript \(j\) denote the \(j^{\text {th }}\) variable, then \(o\) is given by Equation 4.4.
\[
o_{j}= \begin{cases}\left(a_{3}\right)_{j}+C R\left(\left(a_{1}\right)_{j}-\left(a_{2}\right)_{j}\right) & \text { if } u_{j}<F \text { or } j=j_{r}  \tag{4.4}\\ \left(p_{i}\right)_{j} & \text { otherwise }\end{cases}
\]

Thus, we get an offspring corresponding to each index parent. Hence, the size of the offspring population is the same as the size of the parent population. The offspring population is mutated before evaluation. The modified polynomial mutation operator proposed in [91] is used for this purpose. The modification proposed in [91] improves the resilience to premature convergence. The following formulation (taken from [91]) for the polynomial mutation is used with AMGA2. Let \(l_{j}\) and \(u_{j}\) be the lower and upper bound of the variable \(j\). Let \(x_{j}\) and \(x_{j}^{\prime}\) be the value of variable \(j\) before and after the mutation. Let \(r_{j}\) be a uniformly distributed random number in \([0,1]\). Let \(\eta_{m}\) be the distribution index for mutation,
then \(x_{j}^{\prime}\) is given by the set of Equations 4.5.
\[
\begin{align*}
& \delta_{1}=\frac{x_{j}-l_{j}}{u_{j}-l_{j}} \\
& \delta_{2}=\frac{u_{j}-x_{j}}{u_{j}-l_{j}} \\
& \delta_{q}=\left\{\begin{array}{rr}
{\left[2 r_{j}+\left(1-2 r_{j}\right)\left(1-\delta_{1}\right)^{\eta_{m}+1}\right]^{\frac{1}{\eta_{m+1}}}-1,} \\
& \text { if } r_{j} \leq 0.5 \\
1-\left[2\left(1-r_{j}\right)+2\left(r_{j}-0.5\right)\left(1-\delta_{2}\right)^{\eta_{m}+1}\right]^{\frac{1}{\eta_{m}+1}}
\end{array}\right. \\
& x_{j}^{\prime}=x_{j}+\delta_{q}\left(u_{j}-l_{j}\right) \tag{4.5}
\end{align*}
\]

The tuning parameter \(\eta_{m}\) can be set by the user. After rigorous benchmarking on a wide variety of test problems, the vale of \(\eta_{m}\) is set to 15 . This value is used to report all the simulation results in this paper. The mutation operator also requires a probability of mutation. The default value of the probability of mutation is the standard heuristic \(1 / N\), where \(N\) is the number of optimization variables. In [108], it has been shown and argued that the standard heuristic \(1 / N\) may not always be the best choice. The ideal value of the probability of mutation depends upon the landscape of the optimization problem and the rank of the solution in the population, and that it may not be constant during the optimization process. We use the formulation proposed in [108] to compute the probability of mutation for every solution. The following formulation to compute the probability of mutation (taken from [108]) is used with AMGA2. Let the size of the archive be \(N_{a}\), and let the rank of a given parent solution in the archive be \(R_{p}\), then the probability of mutation for the corresponding offspring is given by Equation 4.6.
\[
\begin{equation*}
p=p_{\min }+\left(p_{\max }-p_{\min }\right) *\left(R_{p}-1\right) /\left(N_{a}-1\right) \tag{4.6}
\end{equation*}
\]

In Equation 4.6, the \(p_{\min }\) and \(p_{\max }\) are the minimum and maximum probabilities of mutation respectively. Since the rank of a solution is always greater than or equal to 1
and \(R_{p} \leq N_{a}\), the probability of mutation always lies in [ \(p_{\text {min }}, p_{\text {max }}\) ]. In [108], \(p_{\text {min }}=0.0\) and \(p_{\max }=1.0\) is suggested. With this value of \(p_{\min }\), if the rank of a solution is 1 , then the probability of mutation for that solution is 0 , and hence that solution does not undergo any mutation. There are two differences as compared to [108] in the way the above formulation is applied to AMGA2.
1. Unless the problem is highly multi-modal, most solutions in the parent population belong to rank 1 , and hence would not undergo any mutation if \(p_{\text {min }}=0\). Using \(p_{\text {min }}=0\) is akin to not using mutation at all, which affects the performance of AMGA2. We use \(p_{\text {min }}=1 / N\), where \(N\) is the number of optimization variables. With this value of \(p_{\text {min }}\), the solutions belonging to rank 1 (elite solutions) have the default probability of mutation, and the inferior solutions (rank 2 and beyond) have a higher probability of mutation which favors the optimization process.
2. The mutation is applied to the offspring obtained after the crossover operation. In [108], the offspring are evaluated after the crossover operation, then their rank is determined, and based on their rank, the probability of mutation is computed. Evaluating the offspring after the crossover and again after the mutation operation would effectively double the number of function evaluations. In the case of AMGA2, it is assumed for the purpose of computing the probability of mutation that the offspring has an identical rank to that of its parent. Hence, the rank of its parent is used in Equation 4.6.

Some of the above mentioned modifications to the crossover and mutation operators were performed based on the empirical simulation results on a set of carefully chosen benchmark problems. In step 9, the obtained offspring population is evaluated.

\subsection*{4.3.5 Archive update (step 10)}

The archive maintains a pool of good solutions obtained during the search process. The offspring population is used to update the archive. Initially the archive is empty, and
in step 4, the initial population is copied to the archive. At every iteration, the offspring population is simply added to the archive unless the number of empty slots in the archive population is less than the size of the offspring population. If the size of the initial population is the same as the archive, then in step 10, the archive is already filled, if not, the archive gets filled during the first few generations. The non-dominated sorting [87] procedure is employed to update the archive once the offspring population cannot be completely accommodated in the archive. The archive and the offspring are combined and non-dominated sorting is performed on the combined population. When the solutions of a particular rank cannot be included in the archive, then a pruning method proposed in [101] is used to remove the crowded solutions. In [101], the authors have proposed a novel method for pruning of crowded solutions based on the efficient nearest neighbor search method. At every iteration, two nearest neighbors are found. The solution which has the least value for the second nearest neighbor is deleted. If the second nearest neighbors also have the same Euclidean distance, third nearest neighbors (and so on) are searched. In practice, it is sufficient to search for two nearest neighbors. If one of the solutions from the nearest neighbor pair happens to be a solution on the boundary of the Pareto-optimal front, then the other one is deleted from the set. The process is repeated unless the combined population is trimmed to the maximum allowed size for the archive. This strategy for archive update is similar to (but faster than) the environmental selection procedure proposed in [85].

This archive update strategy favors the domination level (rank) over the diversity. A potential limitation of this approach is exposed when all the solutions in the archive and the offspring population occupy a region in the search space which points to a locally optimal front. The phenomenon of genetic drift will then guide the search towards that locally optimal frontier. Unless, the crossover and mutation operators create a solution that lies inside the globally optimal basin and also belongs to the first rank, the search would lead to a premature convergence. It is practically impossible to detect this phenomenon since the only information available about the search space is contained in the archive which now has all the solutions that belong to a locally optimal basin. Further, it can never be guaranteed
that with the proposed (or any other optimization) algorithm, the global convergence would be achieved. The proposed scheme relies on the discovery of at least one solution in the globally optimal basin which also happens to belong to the first rank (if it does not belong to the first rank, it may be removed whilst updating the archive). There always exists a trade-off between the selection pressure and the diversity required by an evolutionary optimization algorithm. From the rigorous benchmarking conducted, it was concluded that the method proposed in this paper performs better overall on the problems chosen for the benchmark study.

\subsection*{4.3.6 Worst case complexity of AMGA2}

Let the size of the initial, parent, and the archive populations be \(N_{i}, N_{p}\), and \(N_{a}\) respectively. The size of the mating pool and the offspring population would then be \(4 N_{p}\) and \(N_{p}\) respectively. Let the total number of function evaluations be \(T\) and the number of objectives be \(M\). Steps 2, 3, and 4 of AMGA2 take \(\Theta\left(N_{i}\right)\) time each. Step 6 of the AMGA2 involves creation of the parent population from the archive. The solutions belonging the best rank can be extracted in \(\Theta\left(M N_{a}^{2}\right)\) time. The crowding distance assignment for the parent population takes \(\Theta\left(M N_{p} \log N_{p}\right)\) time. Since \(N_{p} \ll N_{a}\), the complexity of step 6 is \(\Theta\left(M N_{a}^{2}\right)\). Each of the steps 7, 8 , and 9 take \(\Theta\left(N_{p}\right)\) time. In step 10 , archive update is performed using non-dominated sorting and pruning based on efficient nearest neighbor search. The complexity of step 10 is \(\Theta\left(N_{a}^{2} \log N_{a}\right)\) assuming \(\log N_{a}>M\) [101]. Thus step 10 is the most time consuming step of the algorithm. Hence, the per iteration complexity of AMGA2 is \(\Theta\left(N_{a}^{2} \log N_{a}\right)\). For \(T\) function evaluations, the number of generations of AMGA2 is \(\frac{\left(T-N_{i}\right)}{N_{p}}\). In general, \(N_{i} \ll T\), hence the number of generations is \(\frac{T}{N_{p}}\). If the desired number of solutions is \(N_{d}\left(<N_{a}\right)\), then step 12 takes \(\Theta\left(N_{a}^{2} \log N_{a}\right)\) time in the worst case. Hence the overall complexity of AMGA2 is \(\Theta\left(\frac{T N_{a}^{2} \log N_{a}}{N_{p}}\right)\).

\subsection*{4.3.7 Differences between AMGA and AMGA2}

Following is the list of differences between AMGA (earlier version) [8] and AMGA2.
- Parent population: In AMGA, the parent population was selected from the archive purely based on the diversity, whereas in AMGA2, the parent population is selected based on the domination level (rank) as well as the diversity.
- Creation of the mating pool: In AMGA, the mating pool is created from the parent population using binary tournament selection, whereas AMGA2 does not use binary tournament selection. In AMGA2, the mating pool contains solutions from the parent population as well as the archive. All the solutions in the parent population are included in the mating pool in the case of AMGA2. The primary (index) parents are from the parent population and auxiliary parents are selected randomly from the archive.
- Composition of the mating pool: AMGA2 ensures that the extreme solutions belonging to rank 1 are always included in the mating pool, whereas AMGA does not. The binary tournament selection in AMGA can compare two extreme solutions and thus eliminate one of them.
- Population sizes: In AMGA, the size of the offspring population is half the size of the parent population, whereas in AMGA2, the size of the parent and the offspring population is the same.
- Diversity assessment: The crowding distance formulation used in AMGA assumes that all the solutions for which the crowding distance metric is being computed have identical rank, whereas in AMGA2, it does not make such an assumption and allows the solutions to have separate ranks.
- Phenotypic versus genotypic niching: AMGA uses diversity assessment in both the genotypic and phenotypic space. In AMGA, the archive update was performed using diversity assessment in the phenotypic space, whereas during the selection of the parent population from the archive, the diversity assessment in the genotypic space was used. AMGA2 always performs diversity assessment in the phenotypic space.
- Genetic variation operators: AMGA uses the simulated binary crossover (SBX) operator, whereas AMGA2 uses the differential evolution (DE) operator for crossover.
- Probability of mutation: In AMGA, the probability of mutation is user-specified (the default value is based on the \(1 / N\) heuristic), whereas in AMGA2, the probability of mutation is based on the rank of the parent solution and is not constant. In AMGA2, the probability of mutation is always greater than or equal to \(1 / N\).

\subsection*{4.4 Simulation Results}

To assess the relative performance of AMGA2, it is benchmarked against some of the state-of-art multi-objective optimization algorithms. The performance of AMGA2 is compared with the AMGA [8], NSGA-II [87], FastPGA [92], MOEA/D [119], and GDE3 [106]. It has been shown in several studies that NSGA-II and SPEA2 have similar performance characteristics [87, 85, 90] and therefore SPEA2 is not included in this benchmark study. The primary reason why these algorithms were chosen for the benchmark study is their superior performance and public availability in the form of jMetal [120] optimization software. The performance of an algorithm depends significantly on the quality of implementation. It is therefore important to choose high quality reference implementations. jMetal [120] is a JAVA based library that provides implementations of the above mentioned algorithms. It has been shown in [120] that the performance of jMetal is similar to the original implementations by the authors of the aforementioned algorithms. Table 4.1 lists the test problems chosen for this study. In Table 4.1, Feval refers to number of function evaluations and | | refers to dimensionality.

The description of the test problems follows next.

Table 4.1: Test Problems
\begin{tabular}{|l|r|r|r|r|l|}
\hline Name & Feval & \(|\mathbf{x}|\) & \(|\mathbf{f}|\) & \(|\mathrm{g}|\) & Remarks \\
\hline FON [36] & 5,000 & 10 & 2 & 0 & Skewed \\
\hline HOLE [121] & 2,000 & 2 & 2 & 0 & Non-convex \\
\hline OSY [36] & 5,000 & 6 & 2 & 6 & Constrained \\
\hline TNK [36] & 2,000 & 2 & 2 & 2 & Constrained \\
\hline ZDT1 [36] & 5,000 & 30 & 2 & 0 & Convex \\
\hline ZDT2 [36] & 5,000 & 30 & 2 & 0 & Non-convex \\
\hline ZDT3 [36] & 5,000 & 30 & 2 & 0 & Discontinuous \\
\hline ZDT4 [36] & 10,000 & 10 & 2 & 0 & Multi-modal \\
\hline ZDT6 [36] & 5,000 & 10 & 2 & 0 & Skewed \\
\hline DTLZ1 [122] & 20,000 & 7 & 3 & 0 & Multi-modal \\
\hline DTLZ2 [122] & 5,000 & 12 & 3 & 0 & Convex \\
\hline DTLZ3 [122] & 20,000 & 12 & 3 & 0 & Multi-modal \\
\hline DTLZ4 [122] & 5,000 & 12 & 3 & 0 & Non-convex \\
\hline DTLZ5 [122] & 5,000 & 12 & 3 & 0 & Non-convex \\
\hline DTLZ6 [122] & 20,000 & 12 & 3 & 0 & Skewed \\
\hline
\end{tabular}

\subsection*{4.4.1 Test Problems}

Test Problem FON
Variable bounds: \(-4 \leq x_{i} \leq 4 \forall i \in\{1, \ldots, n\}\)
\[
\begin{align*}
& f_{1}(x)=1-\exp \left(-\sum_{i=1}^{n}\left(x_{i}-\frac{1}{\sqrt{n}}\right)^{2}\right) \\
& f_{2}(x)=1-\exp \left(-\sum_{i=1}^{n}\left(x_{i}+\frac{1}{\sqrt{n}}\right)^{2}\right) \tag{4.7}
\end{align*}
\]

Test Problem HOLE
Variable bounds: \(-1 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, n\}\)

Problem parameters: \(h=2, \alpha=\pi / 4, p=2, q=0.2, d_{0}=0.02\)
\[
\begin{align*}
& \delta=1-1 / \operatorname{sqrt}(2) \\
& x^{\prime}=x_{1}+\delta \\
& y^{\prime}=x_{2}-\delta \\
& x^{\prime \prime}=x^{\prime} \cos (\alpha)+y^{\prime} \sin (\alpha) \\
& y^{\prime \prime}=-x^{\prime} \sin (\alpha)+y^{\prime} \cos (\alpha) \\
& x^{\prime \prime \prime}=\pi x^{\prime \prime} \\
& y^{\prime \prime \prime}=\pi y^{\prime \prime} \\
& u=\sin \left(x^{\prime \prime \prime} / 2\right) \\
& v=\sin ^{2}\left(y^{\prime \prime \prime} / 2\right) \\
& u^{\prime}= \begin{cases}u^{h} & \text { if } \quad u \geq 0 \\
-(-u)^{h} & \text { otherwise }\end{cases}  \tag{4.8}\\
& v^{\prime}=v^{1 / h} \\
& t=u^{\prime} \\
& a=2 p v^{\prime} \\
& b= \begin{cases}(p-a) e^{q} & \text { if } \quad a \leq p \\
0 & \text { otherwise }\end{cases} \\
& d=d_{0}+a q / 2 \\
& c=q / d^{2} \\
& f_{1}=(t+1)^{2}+a+b e^{-c(t-d)^{2}} \\
& f_{2}=(t-1)^{2}+a+b e^{-c(t+d)^{2}}
\end{align*}
\]

Test Problem OSY
Range of variables 1,2 , and 6: \([0,10]\)
Range of variables 3 and 5: [1, 5]
Range of variable 4: \([0,6]\)
\[
\begin{align*}
& f_{1}(x)=-\left(25\left(x_{1}-2\right)^{2}+\left(x_{2}-2\right)^{2}+\left(x_{3}-1\right)^{2}+\left(x_{4}-4\right)^{2}+\left(x_{5}-1\right)^{2}\right) \\
& f_{2}(x)=x_{1}^{2}+x_{2}^{2}+x_{3}^{2}+x_{4}^{2}+x_{5}^{2}+x_{6}^{2} \\
& c_{1}(x)=x_{1}+x_{2}-2 \geq 0 \\
& c_{2}(x)=6-x_{1}-x_{2} \geq 0  \tag{4.9}\\
& c_{3}(x)=2+x_{1}-x_{2} \geq 0 \\
& c_{4}(x)=2-x_{1}+3 x_{2} \geq 0 \\
& c_{5}(x)=4-\left(x_{3}-3\right)^{2}-x_{4} \geq 0 \\
& c_{6}(x)=\left(x_{5}-3\right)^{2}+\left(x_{6}-4\right) \geq 0
\end{align*}
\]

\section*{Test Problem TNK}

Variable bounds: \(-\pi \leq x_{i} \leq \pi \forall i \in\{1, \ldots, 2\}\)
\[
\begin{align*}
& f_{1}(x)=x_{1} \\
& f_{2}(x)=x_{2} \\
& c_{1}(x)=x_{1}^{2}+x_{2}^{2}-1-0.1 \cos \left(16 \tan ^{-1} \frac{x_{1}}{x_{2}}\right) \geq 0  \tag{4.10}\\
& c_{2}(x)=\left(x_{1}-0.5\right)^{2}+\left(x_{2}-0.5\right)^{2} \leq 0.5
\end{align*}
\]

Test Problem ZDT1
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 30\}\)
\[
\begin{align*}
& f_{1}(x)=x_{1} \\
& g(x)=1+\frac{9}{n-1} \sum_{i=2}^{n} x_{i}  \tag{4.11}\\
& f_{2}(x)=g\left(1-\sqrt{\frac{f_{1}}{g}}\right)
\end{align*}
\]

Test Problem ZDT2

Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 30\}\)
\[
\begin{align*}
& f_{1}(x)=x_{1} \\
& g(x)=1+\frac{9}{n-1} \sum_{i=2}^{n} x_{i}  \tag{4.12}\\
& f_{2}(x)=g\left(1-\left(\frac{f_{1}}{g}\right)^{2}\right)
\end{align*}
\]

Test Problem ZDT3
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 30\}\)
\[
\begin{align*}
f_{1}(x) & =x_{1} \\
g(x) & =1+\frac{9}{n-1} \sum_{i=2}^{n} x_{i}  \tag{4.13}\\
f_{2}(x) & =g\left(1-\sqrt{\frac{f_{1}}{g}}-\frac{f_{1}}{g} \sin \left(10 \pi f_{1}\right)\right)
\end{align*}
\]

Test Problem ZDT4
Range of variable 1: \(0 \leq x_{1} \leq 1\)
Range of other variables: \(-5 \leq x_{i} \leq 5 \forall i \in\{2, \ldots, 10\}\)
\[
\begin{align*}
& f_{1}(x)=x_{1} \\
& g(x)=1+10(n-1)+\sum_{i=2}^{n}\left(x_{i}^{2}-10 \cos \left(4 \pi x_{i}\right)\right)  \tag{4.14}\\
& f_{2}(x)=g\left(1-\sqrt{\frac{f_{1}}{g}}\right)
\end{align*}
\]

Test Problem ZDT6
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 10\}\)
\[
\begin{align*}
& f_{1}(x)=1-\exp \left(-4 x_{1}\right) \sin ^{6}\left(6 \pi x_{1}\right) \\
& g(x)=1+9\left(\frac{\sum_{i=2}^{n} x_{i}}{9}\right)^{0.25}  \tag{4.15}\\
& f_{2}(x)=g\left(1-\left(\frac{f_{1}}{g}\right)^{2}\right)
\end{align*}
\]

Test Problem DTLZ1

Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 7\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{7}\left(\left(x_{i}-0.5\right)^{2}-\cos \left(20 \pi\left(x_{i}-0.5\right)\right)\right) \\
g^{\prime} & =100(g+5) \\
f_{1} & =0.5 x_{1} x_{2}\left(1+g^{\prime}\right)  \tag{4.16}\\
f_{2} & =0.5 x_{1}\left(1-x_{2}\right)\left(1+g^{\prime}\right) \\
f_{3} & =0.5\left(1-x_{1}\right)\left(1+g^{\prime}\right)
\end{align*}
\]

Test Problem DTLZ2
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 12\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{12}\left(x_{i}-0.5\right)^{2} \\
f_{1} & =\cos \left(x_{1} \pi / 2\right) \cos \left(x_{2} \pi / 2\right)(1+g)  \tag{4.17}\\
f_{2} & =\cos \left(x_{1} \pi / 2\right) \sin \left(x_{2} \pi / 2\right)(1+g) \\
f_{3} & =\sin \left(x_{1} \pi / 2\right)(1+g)
\end{align*}
\]

Test Problem DTLZ3
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 12\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{12}\left(\left(x_{i}-0.5\right)^{2}-\cos \left(20 \pi\left(x_{i}-0.5\right)\right)\right) \\
g^{\prime} & =100(g+10) \\
f_{1} & =\cos \left(x_{1} \pi / 2\right) \cos \left(x_{2} \pi / 2\right)\left(1+g^{\prime}\right)  \tag{4.18}\\
f_{2} & =\cos \left(x_{1} \pi / 2\right) \sin \left(x_{2} \pi / 2\right)\left(1+g^{\prime}\right) \\
f_{3} & =\sin \left(x_{1} \pi / 2\right)\left(1+g^{\prime}\right)
\end{align*}
\]

Test Problem DTLZ4

Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 12\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{12}\left(x_{i}-0.5\right)^{2} \\
f_{1} & =\cos \left(\frac{x_{1}^{\alpha} \pi}{2}\right) \cos \left(\frac{x_{2}^{\alpha} \pi}{2}\right)(1+g)  \tag{4.19}\\
f_{2} & =\cos \left(\frac{x_{1}^{\alpha} \pi}{2}\right) \sin \left(\frac{x_{2}^{\alpha} \pi}{2}\right)(1+g) \\
f_{3} & =\sin \left(\frac{x_{1}^{\alpha} \pi}{2}\right)(1+g)
\end{align*}
\]

Test Problem DTLZ5
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 12\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{12}\left(x_{i}-0.5\right)^{2} \\
\theta_{1} & =\frac{\pi\left(1+2 g x_{1}\right)}{4(1+g)} \\
\theta_{2} & =\frac{\pi\left(1+2 g x_{2}\right)}{4(1+g)}  \tag{4.20}\\
f_{1} & =\cos \left(\theta_{1} \pi / 2\right) \cos \left(\theta_{2} \pi / 2\right)(1+g) \\
f_{2} & =\cos \left(\theta_{1} \pi / 2\right) \sin \left(\theta_{2} \pi / 2\right)(1+g) \\
f_{3} & =\sin \left(\theta_{1} \pi / 2\right)(1+g)
\end{align*}
\]

Test Problem DTLZ6
Variable bounds: \(0 \leq x_{i} \leq 1 \forall i \in\{1, \ldots, 12\}\)
\[
\begin{align*}
g & =\sum_{i=3}^{12} x_{i}^{0.1} \\
\theta_{1} & =\frac{\pi\left(1+2 g x_{1}\right)}{4(1+g)} \\
\theta_{2} & =\frac{\pi\left(1+2 g x_{2}\right)}{4(1+g)}  \tag{4.21}\\
f_{1} & =\cos \left(\theta_{1} \pi / 2\right) \cos \left(\theta_{2} \pi / 2\right)(1+g) \\
f_{2} & =\cos \left(\theta_{1} \pi / 2\right) \sin \left(\theta_{2} \pi / 2\right)(1+g) \\
f_{3} & =\sin \left(\theta_{1} \pi / 2\right)(1+g)
\end{align*}
\]

\subsection*{4.4.2 Performance Indicators}

Two unary performance indicators are used for the comparison of different algorithms. The two performance indicators are the delineation metric and the hypervolume metric. In order to use these performance indicators, the true Pareto-optimal front must be known. A smaller value for a performance indicator implies a better solution set. Ideally if the original Pareto optimal front is used as the solution set, all the performance indicators should evaluate to zero. Since a finite number of points (approximately 1,000 and 10,000 points for the case of two and three objectives respectively) are used to represent the true Pareto-optimal frontier, a value of 0.01 or less for a performance indicator implies that the obtained solution set is virtually indistinguishable from the Pareto optimal front. If the value of the performance indicator is 0.1 or more, it implies that an acceptable solution set was not obtained. All the objectives are normalized (the Pareto-optimal front for the problem is mapped to the range \([0,1]\) ) before the performance indicators are computed. Only the non-dominated solutions belonging to rank 1 are considered for computing the performance indicators. For the purpose of computing the hypervolume metric, the nadir objective vector for all the problems is taken as [1.1, 1.1] and [1.1, 1.1, 1.1] for two and three objectives respectively. Any solution that is dominated by the nadir objective vector is not included in the computation of the performance indicators. For constrained problems, only the feasible solutions are considered for computing the performance indicators. If in a solution set, there are no points that dominate the nadir objective vector, all performance indicators are assigned a value of 1.0 for that set. A brief description of each performance indicator follows next.

Delineation Metric: This metric measures "how well is the Pareto-optimal front represented by the obtained solution set". To quantify this information, a large set of evenly spaced points on the Pareto-optimal front is generated. Let the size of this set be \(H\). The minimum Euclidean distance of each point in this set from the obtained solution set is computed. Let this distance be \(l_{i}\) for the \(i^{\text {th }}\) element of the Pareto-optimal set. Then
the delineation metric is given by Equation 4.22.
\[
\begin{equation*}
\text { Delineation metric }=\frac{1}{H} \sum_{i=1}^{H} l_{i} \tag{4.22}
\end{equation*}
\]

The delineation metric for the case of two objectives is pictorially depicted in Figure 4.3.


Figure 4.3: Delineation Metric

Hypervolume Metric: This metric measures the fraction of the search space not dominated by the obtained solution set in comparison to the true Pareto-optimal set. It is the ratio of the areas/volumes dominated by the obtained solution set and the Pareto optimal set subtracted from 1. Let the area dominated by the Pareto-optimal set be \(A_{1}\) and the area dominated by the obtained solution set be \(A_{2}\). Then the hypervolume metric is given by Equation 4.23.
\[
\begin{equation*}
\text { Hypervolume Metric }=1-\frac{A_{2}}{A_{1}} \tag{4.23}
\end{equation*}
\]

The hypervolume metric for the case of two objectives is pictorially depicted in Figure 4.4. The area \(A_{1}\) is the combined area \(A_{2}\) and \(A_{3}\) in Figure 4.4.


Figure 4.4: Hypervolume Metric

Both of the above mentioned metrics measure the convergence (proximity to the Pareto-optimal front) and the spread of the obtained solution set.

\subsection*{4.4.3 Simulation Parameters}

Identical parameter settings (wherever possible) are used for all the algorithms. The parameter settings used for each algorithm are as follows.

Common parameter settings for all the algorithms:
- Size of initial population \(=100\)
- Crossover probability \(=1.0\)
- Mutation probability \(=1 / N\), where \(N\) is the number of variables
- Crossover distribution index \(=15.0\)
- Mutation distribution index \(=20.0\)

For FastPGA, AMGA, and AMGA2, the maximum population size and the size of the archive is set to 100 . The size of the parent population for the case of AMGA is set to

8 and 12 for two and three objectives respectively. For the case of AMGA2, the size of the parent population is set to \(2 M\), where \(M\) is the number of objectives. GDE3, MOEA/D, and AMGA2 use the DE operator. The parameters \(C R\) and \(F\) for the DE operator are set to 0.1 and 0.5 respectively. For the case of AMGA2, the number of solutions desired as the outcome of the optimization process is set to 100 .

\subsection*{4.4.4 Results in tabulated format}

30 random simulations are performed for each test problem. Tables 4.2, 4.3, and 4.4 give the mean, median, and standard deviation (s.d.) of the delineation metric, and Tables 4.5, 4.6, and 4.7 give the hypervolume metric.

Table 4.2: The Delineation Metric (FON, HOLE, OSY, TNK)
\begin{tabular}{|c|c|c|c|c|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline \multirow{6}{*}{FON} & NSGA-II & 0.033328 & 0.033423 & 0.004882 \\
\hline & FastPGA & 0.038495 & 0.035410 & 0.011395 \\
\hline & MOEA/D & 0.009778 & 0.009759 & 0.001526 \\
\hline & GDE3 & 0.035392 & 0.036432 & 0.007791 \\
\hline & AMGA & 0.091890 & 0.093707 & 0.013160 \\
\hline & AMGA2 & 0.019208 & 0.019038 & 0.002042 \\
\hline \multirow{6}{*}{HOLE} & NSGA-II & 0.018750 & 0.017917 & 0.003473 \\
\hline & FastPGA & 0.027980 & 0.025792 & 0.008890 \\
\hline & MOEA/D & 0.013574 & 0.013771 & 0.001036 \\
\hline & GDE3 & 0.025673 & 0.026131 & 0.002410 \\
\hline & AMGA & 0.084684 & 0.086311 & 0.016726 \\
\hline & AMGA2 & 0.031642 & 0.031468 & 0.004848 \\
\hline \multirow{6}{*}{OSY} & NSGA-II & 0.056680 & 0.042740 & 0.034848 \\
\hline & FastPGA & 0.122845 & 0.103005 & 0.081251 \\
\hline & MOEA/D & 1.000000 & 1.000000 & 1.000000 \\
\hline & GDE3 & 0.080992 & 0.073561 & 0.037653 \\
\hline & AMGA & 0.066301 & 0.048541 & 0.046777 \\
\hline & AMGA2 & 0.068687 & 0.042963 & 0.074098 \\
\hline \multirow{6}{*}{TNK} & NSGA-II & 0.019581 & 0.019182 & 0.002851 \\
\hline & FastPGA & 0.023293 & 0.022671 & 0.004973 \\
\hline & MOEA/D & 0.948530 & 0.948530 & 0.000000 \\
\hline & GDE3 & 0.029341 & 0.029218 & 0.003866 \\
\hline & AMGA & 0.048896 & 0.048223 & 0.013248 \\
\hline & AMGA2 & 0.034625 & 0.033944 & 0.004420 \\
\hline
\end{tabular}

Table 4.3: The Delineation Metric (ZDT1 to ZDT6)
\begin{tabular}{|c|c|c|c|c|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline \multirow{6}{*}{ZDT1} & NSGA-II & 0.124241 & 0.115124 & 0.028282 \\
\hline & FastPGA & 0.070746 & 0.068536 & 0.024201 \\
\hline & MOEA/D & 0.957797 & 0.960316 & 0.178803 \\
\hline & GDE3 & 0.177899 & 0.178106 & 0.016735 \\
\hline & AMGA & 0.005664 & 0.005625 & 0.000709 \\
\hline & AMGA2 & 0.004329 & 0.004342 & 0.000168 \\
\hline \multirow{6}{*}{ZDT2} & NSGA-II & 0.336245 & 0.292257 & 0.157124 \\
\hline & FastPGA & 0.596523 & 0.525064 & 0.381111 \\
\hline & MOEA/D & 1.872206 & 1.899713 & 0.168253 \\
\hline & GDE3 & 0.313339 & 0.315867 & 0.040313 \\
\hline & AMGA & 0.005544 & 0.005501 & 0.000704 \\
\hline & AMGA2 & 0.004425 & 0.004388 & 0.000215 \\
\hline \multirow{6}{*}{ZDT3} & NSGA-II & 0.068866 & 0.068499 & 0.014277 \\
\hline & FastPGA & 0.060136 & 0.054861 & 0.026190 \\
\hline & MOEA/D & 0.609860 & 0.613550 & 0.093892 \\
\hline & GDE3 & 0.122049 & 0.125044 & 0.021497 \\
\hline & AMGA & 0.003910 & 0.003652 & 0.000521 \\
\hline & AMGA2 & 0.004601 & 0.004423 & 0.000646 \\
\hline \multirow{6}{*}{ZDT4} & NSGA-II & 0.583238 & 0.554355 & 0.280825 \\
\hline & FastPGA & 0.306142 & 0.252223 & 0.208404 \\
\hline & MOEA/D & 14.927236 & 14.787940 & 4.113457 \\
\hline & GDE3 & 0.964375 & 0.905508 & 0.254419 \\
\hline & AMGA & 0.155994 & 0.135525 & 0.116532 \\
\hline & AMGA2 & 0.019508 & 0.005786 & 0.033344 \\
\hline \multirow{6}{*}{ZDT6} & NSGA-II & 1.495550 & 1.508978 & 0.206766 \\
\hline & FastPGA & 0.612522 & 0.635897 & 0.146051 \\
\hline & MOEA/D & 2.074136 & 2.104987 & 0.827680 \\
\hline & GDE3 & 0.007958 & 0.007516 & 0.002246 \\
\hline & AMGA & 0.013612 & 0.014033 & 0.003160 \\
\hline & AMGA2 & 0.003913 & 0.003879 & 0.000136 \\
\hline
\end{tabular}

The two-objective test problem ZDT4 is multi-modal and therefore different algorithms achieve different levels of convergence. The obtained solution set for all the 30 runs is plotted for all the algorithms in Figures 4.5 to 4.10 . Similarly, the three objective test problem DTLZ6 has a heavily biased distribution of points in the objective space and thus there is a wide difference in performance of different algorithms. The plots of the obtained solution set for all the 30 runs for all the algorithms are shown in Figures 4.11 to 4.16 .

Table 4.4: The Delineation Metric (DTLZ1 to DTLZ6)
\begin{tabular}{|c|c|c|c|c|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline \multirow{6}{*}{DTLZ1} & NSGA-II & 0.518550 & 0.604787 & 0.402408 \\
\hline & FastPGA & 0.389875 & 0.095852 & 0.438646 \\
\hline & MOEA/D & 0.555108 & 0.377047 & 0.307625 \\
\hline & GDE3 & 0.048683 & 0.049002 & 0.001231 \\
\hline & AMGA & 0.459413 & 0.127564 & 0.488491 \\
\hline & AMGA2 & 0.041199 & 0.041032 & 0.000871 \\
\hline \multirow{6}{*}{DTLZ2} & NSGA-II & 0.073785 & 0.073363 & 0.004181 \\
\hline & FastPGA & 0.070218 & 0.070212 & 0.002842 \\
\hline & MOEA/D & 0.472554 & 0.477116 & 0.020505 \\
\hline & GDE3 & 0.062893 & 0.063201 & 0.002238 \\
\hline & AMGA & 0.309295 & 0.333152 & 0.067378 \\
\hline & AMGA2 & 0.054588 & 0.054468 & 0.000792 \\
\hline \multirow{6}{*}{DTLZ3} & NSGA-II & 16.552686 & 15.784300 & 6.284219 \\
\hline & FastPGA & 9.319947 & 8.842502 & 4.895225 \\
\hline & MOEA/D & 8.996618 & 3.529262 & 15.477025 \\
\hline & GDE3 & 0.258066 & 0.061528 & 0.463700 \\
\hline & AMGA & 7.440907 & 7.502221 & 3.466576 \\
\hline & AMGA2 & 0.207031 & 0.058179 & 0.334602 \\
\hline \multirow{6}{*}{DTLZ4} & NSGA-II & 0.082297 & 0.069592 & 0.066827 \\
\hline & FastPGA & 0.269045 & 0.071456 & 0.316734 \\
\hline & MOEA/D & 0.656529 & 0.823565 & 0.176036 \\
\hline & GDE3 & 0.099036 & 0.078760 & 0.046190 \\
\hline & AMGA & 0.073391 & 0.074372 & 0.005672 \\
\hline & AMGA2 & 0.067973 & 0.055246 & 0.069276 \\
\hline \multirow{6}{*}{DTLZ5} & NSGA-II & 0.010143 & 0.010337 & 0.000863 \\
\hline & FastPGA & 0.007949 & 0.007805 & 0.000729 \\
\hline & MOEA/D & 0.018428 & 0.018320 & 0.000985 \\
\hline & GDE3 & 0.011636 & 0.011324 & 0.002268 \\
\hline & AMGA & 0.081682 & 0.065278 & 0.045547 \\
\hline & AMGA2 & 0.006802 & 0.006852 & 0.000327 \\
\hline \multirow{6}{*}{DTLZ6} & NSGA-II & 1.814110 & 1.797024 & 0.083191 \\
\hline & FastPGA & 1.542518 & 1.540620 & 0.149125 \\
\hline & MOEA/D & 0.015706 & 0.015704 & 0.000005 \\
\hline & GDE3 & 0.005289 & 0.005286 & 0.000082 \\
\hline & AMGA & 4.455544 & 4.426605 & 0.240810 \\
\hline & AMGA2 & 0.004928 & 0.004926 & 0.000030 \\
\hline
\end{tabular}

Table 4.5: The Hypervolume Metric (FON, HOLE, OSY, TNK)
\begin{tabular}{|l|l|r|r|r|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline \multirow{20}{*}{ FON } & NSGA-II & 0.126551 & 0.127475 & 0.013365 \\
& FastPGA & 0.133967 & 0.139028 & 0.021580 \\
& MOEA/D & \(\mathbf{0 . 0 4 0 8 3 9}\) & \(\mathbf{0 . 0 3 9 2 8 1}\) & 0.009361 \\
& GDE3 & 0.133323 & 0.133761 & 0.019887 \\
& AMGA & 0.295983 & 0.299912 & 0.042073 \\
& AMGA2 & 0.064552 & 0.064344 & 0.006502 \\
\hline \multirow{6}{*}{ OSY } & NSGA-II & 0.023394 & 0.021343 & 0.004891 \\
& FastPGA & 0.029900 & 0.029046 & 0.007229 \\
& MOEA/D & \(\mathbf{0 . 0 1 4 4 0 6}\) & \(\mathbf{0 . 0 1 4 2 3 1}\) & 0.002116 \\
& GDE3 & 0.033702 & 0.032436 & 0.004099 \\
& AMGA & 0.139212 & 0.134561 & 0.037726 \\
& AMGA2 & 0.041974 & 0.041331 & 0.007571 \\
\hline \multirow{5}{*}{ TNK } & NSGA-II & \(\mathbf{0 . 0 5 2 1 7 8}\) & 0.049751 & 0.018146 \\
& FastPGA & 0.143545 & 0.075790 & 0.145632 \\
& MOEA/D & 1.000000 & 1.000000 & 1.000000 \\
& GDE3 & 0.116113 & 0.082553 & 0.166879 \\
& AMGA & 0.215348 & 0.068164 & 0.317091 \\
& AMGA2 & 0.098435 & \(\mathbf{0 . 0 4 8 4 7 4}\) & 0.131722 \\
\hline & NSGA-II & \(\mathbf{0 . 0 6 0 3 0 0}\) & \(\mathbf{0 . 0 5 8 6 9 8}\) & 0.007167 \\
& FastPGA & 0.064440 & 0.062455 & 0.011430 \\
& MOEA/D & 1.000000 & 1.000000 & 1.000000 \\
& GDE3 & 0.091830 & 0.090822 & 0.012842 \\
& AMGA & 0.145134 & 0.143252 & 0.039542 \\
& AMGA2 & 0.100375 & 0.101111 & 0.009048 \\
\hline
\end{tabular}


Figure 4.5: NSGA-II on ZDT4


Figure 4.6: FastPGA on ZDT4

Table 4.6: The Hypervolume Metric (ZDT1 to ZDT6)
\begin{tabular}{|l|l|r|r|r|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline & NSGA-II & 0.207196 & 0.208715 & 0.052790 \\
& FastPGA & 0.117291 & 0.116519 & 0.040817 \\
ZDT1 & MOEA/D & 0.977323 & 1.000000 & 0.054455 \\
& GDE3 & 0.280129 & 0.297759 & 0.053907 \\
& AMGA & 0.010508 & 0.010402 & 0.001683 \\
& AMGA2 & \(\mathbf{0 . 0 0 6 2 7 6}\) & \(\mathbf{0 . 0 0 6 3 6 3}\) & 0.000359 \\
\hline \multirow{21}{*}{ ZDT2 } & NSGA-II & 0.700448 & 0.663080 & 0.154196 \\
& FastPGA & 0.786750 & 0.889481 & 0.238002 \\
& MOEA/D & 1.000000 & 1.000000 & 1.000000 \\
& GDE3 & 0.694884 & 0.704786 & 0.076690 \\
& AMGA & 0.016070 & 0.015647 & 0.002930 \\
& AMGA2 & \(\mathbf{0 . 0 1 0 1 3 5}\) & \(\mathbf{0 . 0 1 0 1 5 8}\) & 0.000744 \\
\hline & NSGA-II & 0.178085 & 0.183334 & 0.036314 \\
& FastPGA & 0.154901 & 0.140984 & 0.054652 \\
& MOEA/D & 0.891509 & 0.909226 & 0.068592 \\
& GDE3 & 0.245671 & 0.248364 & 0.062358 \\
& AMGA & 0.006140 & 0.005838 & 0.001183 \\
& AMGA2 & \(\mathbf{0 . 0 0 4 2 3 3}\) & \(\mathbf{0 . 0 0 4 0 9 6}\) & 0.000694 \\
\hline & NSGA-II & 0.692898 & 0.752109 & 0.268926 \\
& FastPGA & 0.455393 & 0.390064 & 0.262794 \\
& MOEA/D & 1.000000 & 1.000000 & 1.000000 \\
& GDE3 & 0.932448 & 0.987881 & 0.121536 \\
& AMGA & 0.253258 & 0.218962 & 0.187481 \\
& AMGA2 & \(\mathbf{0 . 0 3 2 3 2 2}\) & \(\mathbf{0 . 0 1 1 1 9 7}\) & 0.052631 \\
\hline & NSGA-II & 1.000000 & 1.000000 & 1.000000 \\
& FastPGA & 0.922934 & 0.960403 & 0.081214 \\
ZDT6 & MOEA/D & 0.993003 & 1.000000 & 0.037678 \\
& GDE3 & 0.016311 & 0.015345 & 0.004043 \\
& AMGA & 0.039509 & 0.040341 & 0.009272 \\
& AMGA2 & \(\mathbf{0 . 0 0 8 4 2 6}\) & \(\mathbf{0 . 0 0 8 0 3 5}\) & 0.000937 \\
\hline
\end{tabular}

Table 4.7: The Hypervolume Metric (DTLZ1 to DTLZ6)
\begin{tabular}{|c|c|c|c|c|}
\hline Problem & Algorithm & Mean & Median & S.D. \\
\hline \multirow{6}{*}{DTLZ1} & NSGA-II & 0.580963 & 0.822832 & 0.387997 \\
\hline & FastPGA & 0.431443 & 0.116650 & 0.392443 \\
\hline & MOEA/D & 0.709326 & 0.611224 & 0.187369 \\
\hline & GDE3 & 0.045040 & 0.045144 & 0.003155 \\
\hline & AMGA & 0.448058 & 0.127742 & 0.447200 \\
\hline & AMGA2 & 0.031655 & 0.031526 & 0.000987 \\
\hline \multirow{6}{*}{DTLZ2} & NSGA-II & 0.157148 & 0.156651 & 0.011779 \\
\hline & FastPGA & 0.143555 & 0.143509 & 0.010360 \\
\hline & MOEA/D & 0.639878 & 0.641498 & 0.031353 \\
\hline & GDE3 & 0.126472 & 0.125976 & 0.006174 \\
\hline & AMGA & 0.729883 & 0.789439 & 0.204085 \\
\hline & AMGA2 & 0.076977 & 0.076616 & 0.002352 \\
\hline \multirow{6}{*}{DTLZ3} & NSGA-II & 1.000000 & 1.000000 & 1.000000 \\
\hline & FastPGA & 1.000000 & 1.000000 & 1.000000 \\
\hline & MOEA/D & 0.943981 & 1.000000 & 0.089959 \\
\hline & GDE3 & 0.278921 & 0.112272 & 0.342319 \\
\hline & AMGA & 1.000000 & 1.000000 & 1.000000 \\
\hline & AMGA2 & 0.239726 & 0.079818 & 0.338699 \\
\hline \multirow{6}{*}{DTLZ4} & NSGA-II & 0.159517 & 0.149807 & 0.051814 \\
\hline & FastPGA & 0.293693 & 0.149202 & 0.263984 \\
\hline & MOEA/D & 0.630743 & 0.690828 & 0.073948 \\
\hline & GDE3 & 0.135246 & 0.118175 & 0.049836 \\
\hline & AMGA & 0.087014 & 0.086678 & 0.004098 \\
\hline & AMGA2 & 0.088922 & 0.077942 & 0.062460 \\
\hline \multirow{6}{*}{DTLZ5} & NSGA-II & 0.033578 & 0.033758 & 0.003757 \\
\hline & FastPGA & 0.024866 & 0.024442 & 0.003078 \\
\hline & MOEA/D & 0.070690 & 0.068854 & 0.007149 \\
\hline & GDE3 & 0.037275 & 0.037967 & 0.006106 \\
\hline & AMGA & 0.293194 & 0.237189 & 0.155638 \\
\hline & AMGA2 & 0.019995 & 0.020155 & 0.001289 \\
\hline \multirow{6}{*}{DTLZ6} & NSGA-II & 1.000000 & 1.000000 & 1.000000 \\
\hline & FastPGA & 1.000000 & 1.000000 & 1.000000 \\
\hline & MOEA/D & 0.043718 & 0.043718 & 0.000030 \\
\hline & GDE3 & 0.011717 & 0.011704 & 0.000249 \\
\hline & AMGA & 1.000000 & 1.000000 & 1.000000 \\
\hline & AMGA2 & 0.011432 & 0.011411 & 0.000145 \\
\hline
\end{tabular}


Figure 4.7: MOEA/D on ZDT4


Figure 4.9: AMGA on ZDT4


Figure 4.11: NSGA-II on DTLZ6


Figure 4.8: GDE3 on ZDT4


Figure 4.10: AMGA2 on ZDT4


Figure 4.12: FastPGA on DTLZ6


Figure 4.13: MOEA/D on DTLZ6


Figure 4.14: GDE3 on DTLZ6


Figure 4.15: AMGA on DTLZ6


Figure 4.16: AMGA2 on DTLZ6

The hypervolume measure captures the convergence, the spread, and the uniformity of the points along the Pareto-optimal frontier. Thus, it is a good metric to compare different algorithms. In Tables 4.5, 4.6, and 4.7, the mean, median, and standard deviation of 30 simulation runs are reported. To gain more confidence in the values of the hypervolume metric, a statistical analysis using ANOVA (analysis of variance) [123] is presented. There are six algorithms and 15 test problems, thus we have six groups, each with a sample size of 15. The mean value of the hypervolume metric is used to assess the confidence level in the obtained performance metrics. The single factor ANOVA is used to determine if there is a statistically significant difference between the six algorithms. The significance level used for this test is 0.05 (i.e. \(5 \%\) probability that the statement "means are equal" is true). The P-value obtained after the single factor analysis is 0.0001112 , which is significantly smaller than the significance level of 0.05 . Thus, it can be concluded that the probability of means being equal is negligible. Hence, the mean values for the hypervolume metric reported in Tables \(4.5,4.6\), and 4.7 are statistically different.

\subsection*{4.5 Observations and Inferences}

As is evident from the simulation results, the AMGA2 has better performance on the ZDT and DTLZ set of test problems. On the test problems FON and OSY, its performance is similar to the best performing algorithm. On the test problems HOLE and TNK, which have two variables, the AMGA2 does not perform as well as the others. The test problems used in this study offer a variety of function landscapes and varying levels of difficulty to an optimization algorithm. Still, these problems represent only a small subset of problems that may be encountered in practice. The test problems ZDT4, DTLZ1, and DTLZ3 are multimodal, whereas the test problems FON, ZDT6, and DTLZ6 have a highly skewed search space and thus challenge an optimization algorithm's ability to find the global Paretooptimal frontier. Hence, the test problems FON, ZDT4, ZDT6, DTLZ1, DTLZ3, and DTLZ6 are considered to be good benchmark problems, and it is suggested to use these
problems to compare different algorithms.
AMGA2, GDE3, and MOEA/D use the DE operator to create new solutions. These three algorithms in general have superior performance as compared to NSGA-II, FastPGA, and AMGA. We can thus infer from the simulation results that the DE operator is a robust and reliable crossover operator to create the offspring population. In AMGA2, the parent population is selected based on the domination level and diversity whereas in the literature [106], the DE operator is fed with randomly selected parents. It was observed during the development and testing of AMGA2 that the choice of the parent population has a significant impact on the performance of the algorithm. Further, during the initial stages of the search when the population is still moving towards the Pareto-optimal frontier, using a large parent population tends to waste function evaluations. During the later stages of the search, when the population is very close to the Pareto-optimal frontier, exploring the neighborhood of all the solutions in the archive also wastes function evaluations. Only those solutions which have relatively large gaps around them need to be explored. It was observed that using a very small parent population greatly speeds up the search process. Decreasing the size of the parent population beyond a threshold also hampers the performance of the algorithm. In our simulation results, setting the size of the parent population to be the twice of number of objectives yielded an optimum balance between convergence rate and robustness of the AMGA2. On multi-modal problems ZDT4 and DTLZ3, it was observed that using the rank-based mutation probability [108] improved the performance of AMGA2. On DTLZ1 (which is also multi-modal) and other problems which are not multi-modal, no noticeable improvement was observed. For heavily multi-modal problems, during the search process, individuals with multiple ranks are present in the archive. Some of these individuals not belonging to rank 1 when included in the parent population are mutated with a higher probability which adds to the disruptiveness of the search process; a feature desired for multi-modal problems. With the exception of DTLZ1, this observation is in accordance with the expected benefits of using rank-based mutation probability as stated in [108].

It was also observed that if the maximum number of function evaluations is sig-
nificantly increased ( \(\approx 100,000\) ), the MOEA/D eventually outperforms other algorithms. However, in most real-world optimization scenarios, the computational resources are limited and therefore AMGA2 may be a better choice since it has a much faster convergence rate. Also, for very large number of function evaluations ( \(\approx 100,000\) ), it was observed that NSGAII, FastPGA, GDE3, AMGA, and AMGA2 have nearly identical performance. It can be attributed to that fact that full convergence was achieved and the algorithms had sufficient function evaluations at their disposal to find a good distribution along the Pareto-optimal frontier.

AMGA2 places a high emphasis on the diversity aspect during the selection of solutions for inclusion in the parent population. The binary tournament selection emphasizes the domination level (primary fitness) over diversity (secondary fitness), and may compare two extreme solutions thereby eliminating one of them. The AMGA2 gets rid of the binary tournament selection and thus exercises full control over the selection of the parent population. It reduces the randomness in the search process which might not be desirable. To mitigate this issue, the AMGA2 selects auxiliary parents randomly. The auxiliary parents seed the change in the optimization variables during the DE based crossover. The effect of such a strategy is to enable AMGA2 to continuously try to stretch the approximation to the Pareto-optimal front in all directions and to enable it to place more emphasis on filling the gaps in the Pareto-optimal front. The diversity operator used in AMGA2 is of \(\Theta\left(N_{a}^{2} \log N_{a}\right)\) complexity and thus increasing the size of the archive significantly increases the execution time. A faster yet equally efficient diversity preservation technique can help in reducing the computational complexity of AMGA2. Further, in most MOEAs, increasing the population size while keeping the number of generations constant increases the total number of function evaluations. AMGA2 removes this coupling and thus facilitates independent tuning of the population size (size of the archive) and the number of function evaluations.

\subsection*{4.6 Conclusion}

In this paper, an improved version of the Archive-based Micro Genetic Algorithm (referred to as AMGA2) is proposed. The AMGA2 incorporates several modifications and improvements to the AMGA, and has significantly better performance on the problems chosen for the benchmark study. It benefits from the existing literature in that it borrows (and improves upon) several novel concepts from existing algorithms. Thus, it is also an exercise in combining the best features and concepts into a single optimization algorithm. Based on the design of AMGA2, simulation results obtained, and our observations and inferences from those results, the following conclusions can be drawn from this study.
- Micro-genetic algorithms working with a very small population size tend to reduce the number of function evaluations required to achieve similar results.
- The differential evolution operator is the most robust and high-performing recombination operator tested in this study. It clearly outperforms SBX [44] and PCX [103].
- Instead of using a universal heuristic \((1 / N)\) for the probability of mutation, using a rank-based mutation probability [108] is suggested. It can have significant impact on the performance especially on the multi-modal problems.
- The diversity aspect is very crucial and important to the performance of AMGA2. It also is the most computationally expensive operation performed by AMGA2.
- The pseudo-code for the AMGA2 proposed in this paper functionally decomposes all the conceptual steps of an evolutionary multi-objective optimization algorithm. The pseudo-code can be used as a recipe to create customized optimization algorithms. It is also possible to automate the creation of new optimization algorithms using a set of rules.
- Use of an external archive to store good solutions, and a separate parent population which is gleaned from the archive decouples the allowed number of function evaluations
and the desired elite size. AMGA2 can report a significantly larger non-dominated set (thereby artificially inflating its performance characteristics) for the same number of function evaluations.

Another design goal of AMGA2 was to automatically fine-tune all the tuning parameters (except the number of function evaluations and the desired number of solutions) required by the optimization algorithm. AMGA2 achieves this goal by using the problem size as the metric to determine the suitable values of the tuning parameters. All the simulation results in this paper are reported using the automatic tuning feature of AMGA2.

\subsection*{4.7 Ideas for Future Research}

There exists significant scope for future improvement of AMGA2. A faster diversity assessment operator can significantly speed-up its execution time. The AMGA2 does not make use of any explicit knowledge integration characteristics like use of meta-modeling and estimation of distribution as part of the optimization process, and use of correlations and past search history to predict the step sizes and the preferred search directions. Not every modification/idea attempted improved the performance of AMGA2. The following is a list of some of the concepts that were implemented but did not yield any noticeable improvement in the performance of AMGA2.
- Excessive emphasis on diversity: Whilst updating the archive, the diversity of the solutions can be further increased by choosing solutions that belong to rank 2 and beyond such that they are farthest from the solutions in rank 1. To achieve this, after the solutions belonging to rank 1 are identified, the pruning method can be modified such that if the pair of closest neighbors contains a solution belonging to rank 1 , the other is removed. With this modification, only those inferior solutions will be included in the archive that are very far away from the best solutions. This strategy however inhibits the convergence rate and stalls the search process.
- Explicit handling of infeasible solutions: All the optimization algorithms used in this study work under the assumption that any feasible solution is better than any infeasible solution. While this property may be desired whilst reporting the final set of solutions, during the optimization, such a strategy is not always desirable. For constrained test problems which have a portion of their Pareto-front on a constraint boundary, a solution that is infeasible but is very close to the boundary has a higher probability of generating a better offspring than a solution that is feasible but is very far from the Pareto-optimal front. Because of this reason, very often during the search process, evolutionary algorithms find it difficult to obtain a fine-grained fitness along the portion of the Pareto-optimal front that lies on the constraint boundary. To remedy this behavior, a tolerance in the constraint violation was introduced which treated infeasible solutions as feasible if the constraint violation was less than a specified threshold. Since the objective values of infeasible solutions are not available (cannot be used), the diversity assessment in the decision space was done. It had the effect of distributing the solutions along the entire constraint boundary (portions of which were far away from the Pareto-optimal front). Since the objective values for infeasible solutions are not available, it is very difficult to predict which portions of constraint boundary should be favored. Improved constraint handling in evolutionary multi-objective optimization algorithms can significantly improve their performance on constrained test problems.
- Explicit regeneration: In the case of multi-modal problems, the evolution often stalls and gets stuck at a local optimum. The AMGA2 can detect if the search has stalled (number of solutions belonging to rank 1 do not increase/no solutions belonging to rank 1 are demoted/the diversity measure does not improve when all the solutions in the archive belong to rank 1). If there was no improvement for a given number of generations, the entire population was regenerated and/or the entire population was mutated. These modifications however resulted in a population which was generally inferior to the existing best solutions. Through the phenomenon of genetic drift,
the non-dominated solutions guide the search towards themselves which leads the population into the same locally optimal basin. After regeneration, if the existing set of best solutions are not used to guide the search process, it would be akin to performing a new optimization.

These issues outlined above need further investigation and are the focus of continuing future research.

\section*{Chapter 5}

\section*{CONCLUSION}

\subsection*{5.1 Concluding Remarks}

In this dissertation, we have presented three algorithms and demonstrated their application on suitable hypothetical and real-world problems. The body of research presented in this dissertation (compact packing and shape morphing) focuses on the integration of geometric and optimization algorithms with the available information about the packing and layout problem. This knowledge infusion is an important concept used in many applications of optimization to configuration design [124, 125, 73, 126]. A careful analysis of these examples reveals the following common characteristics about knowledge infusion.
1. In each case, the solution representation was customized. The customization of the solution representation consisted of designing a suitable encoding to represent the solution set and a set of rules to manipulate/interpret the encoding.
2. The customization of the solution representation lead to a reduction in the size of the search space. It also significantly increased the ratio of feasible to infeasible search space. Thus, this customization made the original problem easier to solve.
3. In certain cases \([124,4,126]\), the customized solution representation allowed for artificial correction of the solution during the evaluation process. This modification made
many infeasible solutions feasible and also improved the performance of the solution.
4. It was also observed that the working principle of the optimizer was not modified in any of the examples. The optimizer retained its global and local search characteristics.

The above mentioned characteristics could potentially be used to guide future attempts at solving configuration design problems for various industrial applications. It has been shown in [73] that the integration of the shape morphing algorithm with the layout algorithm helps in improving the performance (design objectives) of the generated layout. The shape morphing is a geometric algorithm and thus its application further exemplifies the importance of integrating geometric and optimization algorithms.

The review of the existing literature on optimization algorithms and the design and development of the general-purpose optimizer AMGA2 and its subsequent benchmarking leads to the following desirable features that should be present in a high-performing optimization algorithm.
1. Its applicability should not be limited to a specific class of problems; i.e., it should perform consistently on a wide variety of test problems.
2. It should incorporate explicit mechanisms to detect and mitigate premature convergence.
3. It should not require parameter tuning for different problems. It should have selfadaptive operators which can automatically adjust themselves based on the problem size and the objective and constraint function profiles.
4. It should not impose any conditions such as continuity, differentiability, and convexity etc. on the optimization problem or make any assumptions about it.
5. It should be able to monitor its convergence rate and modify its working principle accordingly. This feature is especially desired for the algorithms that are to be used with problems offering varying levels of difficulty.
6. It should facilitate easy customization of the solution representation, and such a customization should not significantly affect its performance. The compact packing problem is a practical engineering application that requires the customization of the optimization algorithm, and thus it is desirable that the optimization algorithm is amenable to modifications.

The AMGA2 partially exhibits some of the above mentioned characteristics. It does not impose any conditions or make any assumptions about the optimization problem. It has a highly disruptive mutation operator to reduce the probability of premature convergence. It uses DE which is self-adaptive and does not require parameter tuning. The probability of mutation in AMGA2 is computed based on the rank of its parent solution and thus AMGA2 monitors its convergence rate (change in the rank of the solutions). Hence, it can be concluded that the AMGA2 is a high-performing optimization algorithm.

\subsection*{5.2 Summary of Contributions}

The research work presented in this dissertation has made the following contributions to the existing state-of-the-art in geometric and optimization algorithm development and their integration to solve packing and layout problems.
1. An algorithm to compactly pack three-dimensional free-form objects with full rotational freedom inside an arbitrary enclosure.
(a) A complete voxel-based CAD engine that can perform surface and volume voxelization, Boolean operations, ray tracing, and ISO-surface extraction.
(b) A Bottom-Left-Back-Fill (BLBF) heuristic that works with arbitrary enclosure geometry and also attempts to fill the voids created by placing relatively large objects.
2. A physically-based free-form shape morphing algorithm to generate a geometry with a desired volume whilst taking into account the spatial constraints.
(a) A general purpose surface collision detection algorithm that works with arbitrary triangulations and deformable geometries. It does not require any preprocessing and can perform Yes/No type queries and also report all the pairs of colliding facets if desired.
(b) An algorithm to automatically generate the mass-spring model from an arbitrary manifold geometry.
3. A fast and efficient constrained multi-objective optimization algorithm AMGA2
(a) A new selection mechanism to create the parent population
(b) An improved formulation for the crowding distance operator
(c) An improved polynomial mutation operator

\subsection*{5.3 Suggestions for Future Work}

Related future work for each of the algorithms presented in this dissertation is described in their respective chapters. In this dissertation, the aspect of integration has only been examined in the context of configuration design (packing and layout problems). It is suggested to explore other diverse avenues for the integration of geometry, optimization, and problem-specific information. The useful information gleaned from a large set of diverse application studies can be used to propose a formalism for knowledge infusion, which can then be applied to solve other similar problems. In particular, it is suggested to explore different kinds of integration techniques dealing with different representations and identify the common principles involved. The configuration design problems are encountered in many industrial applications (trunk loading, shipping, containerization etc.) and therefore many potential applications of the algorithms described in this dissertation exist. It is suggested to explore potential applications that can benefit from the algorithms presented in this dissertation. These applications will further increase the value and usefulness of the research work presented here. The shape morphing can not only be used with layout design,
but also for the automated design of freeform components. It is thus suggested to explore potential applications that can be automated using the shape morphing algorithm. The general purpose optimizer AMGA2 presented in this dissertation is shown to be faster on a set of hypothetical test problems gleaned from the literature. It has not been benchmarked on practical engineering problems. Conducting a benchmark study involving real-world examples will further increase the confidence in the performance of the AMGA2. It is also suggested to perform a more rigorous analysis of the obtained performance metrics using formal statistical tools. The use of such tools will provide greater insight and confidence into the values of the reported metrics and will further increase the credibility of the benchmark study. These and other suggestions are the focus of continuing future research into geometric and optimization algorithms.

To summarize, the research work presented in this dissertation makes useful contributions to the broad area of optimization and geometric algorithms. It also is an effort towards the integration of geometric and optimization algorithms especially in the context of packing and layout design. The author's hope in writing this manuscript is that this collection of work will benefit the optimization and design community, and will spawn future research towards a formalism for the integration of geometry, optimization, and available design domain knowledge.

\section*{REFERENCES}
[1] J. Cagan, K. Shimada, and S. Yin. A survey of computational approaches to threedimensional layout problems. Computer-Aided Design, 34:597-611, 2002.
[2] P. Grignon and G. M. Fadel. A GA based configuration design optimization method. Transactions of ASME, Journal of Mechanical Design, 126(1):6-15, 2004.
[3] S. Tiwari, G. Fadel, and V. Gantovnik. A survey of various encoding schemes and associated placement algorithms applied to packing and layout problems. In Proc. of ASME IDETC \& CIE Conference, 2006.
[4] S. Tiwari, G. Fadel, and P. Fenyes. A fast and efficient compact packing algorithm for free-form objects. In Proc. of ASME IDETC \& CIE Conference, 2008.
[5] S. Tiwari, G. Fadel, and P. Fenyes. A fast and efficient compact packing algorithm for SAE and ISO luggage packing problems. Accepted for publication in Transactions of ASME, Journal of Computing and Information Science in Engineering.
[6] H. Dong and G. Fadel. A bi-level packing with shape morphing. Submitted for publication in Transactions of ASME, Journal of Computing and Information Science in Engineering.
[7] H. Dong, S. Tiwari, G. Fadel, P. Fenyes, and A. Kloess. A physically-based shape morphing algorithm for layout design applications. Submitted for publication in Transactions of ASME, Journal of Computing and Information Science in Engineering.
[8] S. Tiwari, G. Fadel, P. Koch, and K. Deb. AMGA: An archive-based micro genetic algorithm for multi-objective optimization. In Proc. of ACM Genetic and Evolutionary Computation Conference, pages 729-736, 2008.
[9] S. Tiwari, G. Fadel, P. Koch, and K. Deb. Perforamance assessment of the hybrid archive-based micro genetic algorithm (AMGA) on the CEC09 test problems. In Proc. of IEEE Congress on Evolutionary Computation, 2009. In Press.
[10] S. Tiwari, G. Fadel, and K. Deb. AMGA2: Improving the performance of archivebased micro genetic algorithm for multi-objective optimization. Submitted for publication in IEEE Transactions on Evolutionary Computation.
[11] DIN. Deutsches institut fur normung e. v. DIN 70020, teil 1, strabenfahrzeuge, kraftfahrzeugbau, begriffe von abmessungen. Technical report, 1993.
[12] SAE. SAE standard J1100, motor vehicle dimensions. Technical report, 2005.
[13] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. European Journal of Operational Research, 141:241-252, 2002.
[14] L. Zhang and U. Kleine. A novel bottom-left packing genetic algorithm for analog module placement. Advances in Radio Sciences, pages 191-196, 2003.
[15] K. A. Dowsland, S. Vaid, and W. B. Dowsland. An algorithm for polygon placement using a bottom-left strategy. European Journal of Operational Research, 141:371-381, 2002.
[16] E. Hopper and B. C. H. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research, 128:34-57, 2001.
[17] E. Hopper and B. Turton. A genetic algorithm for a 2D industrial packing problem. Computers \(\mathcal{B}\) Industrial Engineering, 37:375-378, 1999.
[18] D. Liu and H. Teng. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operational Research, 112:413-420, 1999.
[19] S. Jakobs. On genetic algorithms for the packing of polygons. European Journal of Operational Research, 88:165-181, 1996.
[20] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest. Orthogonal packing in two dimensions. SIAM Journal on Computing, 9:846-855, 1980.
[21] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. Operations Research, 48(2):256-267, 2000.
[22] S. Yin and J. Cagan. An extended pattern search algorithm for three-dimensional component layout. Transactions of ASME, Journal of Mechanical Design, 122:102108, 2000.
[23] Q. Ding and J. Cagan. Automated trunk packing with extended pattern search. In Proc. of SAE World Congress E Exhibition, 2003.
[24] I. Ikonen, W. E. Biles, A. Kumar, R. K. Ragade, and J. C. Wissel. A genetic algorithm for packing three-dimensional non-convex objects having cavities and holes. In Proc. of \(7^{\text {th }}\) International Conference on Genetic Algorithms, pages 591-598, 1997.
[25] F. Eisenbrand, S. Funke, J. Reichel, and E. Schomer. Packing a trunk. In Proc. of \(11^{\text {th }}\) Annual European Symposium, pages 618-629, 2003.
[26] V. Y. Blouin, G. M. Fadel, J. D. Summers, and P. A. Fenyes. Three-dimensional packing by a heuristic-based sequential genetic algorithm. In Proc. of \(11^{\text {th }}\) AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, 2006.
[27] F. Eisenbrand, S. Funke, A. Karrenbauer, J. Reichel, and E. Schomer. Packing a trunk - now with a twist. In Proc. of ACM Symposimum on Solid and Physical Modeling, pages 197-206, 2005.
[28] E. Althaus, T. Baumann, E. Schomer, and K. Werth. Trunk packing revisited. Lecture Notes in Computer Science, (4525):420-432, 2007.
[29] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979.
[30] M. E. Mortenson. Geometric Modeling. 1997.
[31] J. A. Bennell and X. Song. A comprehensive and robust procedure for obtaining the no-fit polygon using minkowski sums. Technical report, University of Southampton, 2005.
[32] R. Dawkins. The Selfish Gene. Oxford University Press, New York, 1976.
[33] N. Eldredge. Macro-Evolutionary Dynamics: Species, Niches, and Adaptive Peaks. McGraw-Hill, New York, 1989.
[34] J. H. Holland. Adaptation in Natural and Artificial Systems. Ann Arbor, MI: MIT Press, 1975.
[35] D. E. Goldberg. Genetic Algorithms for Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.
[36] K. Deb. Multi-objective Optimization Using Evolutionary Algorithms. Chichester, UK: Wiley, 2001.
[37] C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge Information Systems, 1(3):269-308, 1999.
[38] F. Vavak and T. C. Fogarty. Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In Proc. of IEEE Congress on Evolutionary Computation, pages 192-195, 1996.
[39] T. Back, F. Hoffmeister, and H. Schwefel. A survey of evolution strategies. In Proc. of \(4^{\text {th }}\) International Conference on Genetic Algorithms, pages 2-9, 1991.
[40] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Proc. of Third International Conference on Genetic Algorithms, pages 116-121, 1989.
[41] L. J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Proc. of Foundations of Genetic Algorithms 1, pages 265-283, 1991.
[42] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaption. In Proc. of IEEE Congress on Evolutionary Computation, pages 312-317, 1996.
[43] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the travelling salesman problem. In Proc.of International Conference on Genetic Algorithms and their Applications, 1987.
[44] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. Complex Systems, 9(2):115-148, 1995.
[45] K. Deb and M. Goyal. A combined genetic adaptive search GeneAS for engineering design. Computer Science and Informatics, 26(4):30-45, 1996.
[46] T. A. Moller. Fast 3D triangle-box overlap testing. Journal of Graphics Tools, 6(1):2933, 2001.
[47] M. Alexa. Recent advances in mesh morphing. Computer Graphics Forum, 21(2):173198, 2002.
[48] Y. Chen, P. J. Stewart, P. Buttolo, and F. Ren. A real-time, interactive method for fast modification of large-scale CAE mesh models. In Proc. of ASME IDETC \& CIE Conference, 2000.
[49] F. Lazarus and A. Verroust. Three-dimensional metamorphosis: A survey. The Visual Computer, 14(8-9):373-389, 1998.
[50] P. Moore and D. Molloy. A survey of computer based deformation models. In Proc. of Machine Vision and Image Processing Conference, pages 55-66, 2007.
[51] D. Terzopoulos and K. Waters. Physically based facial modeling, analysis, and animation. Journal of Visualization and Computer Animation, 1(2):73-80, 1990.
[52] A. Gregory, A. State, M. C. Lin, D. Manocha, and M. A. Livingston. Interactive surface decomposition for polyhedral morphing. The Visual Computer, 15(9):453470, 1999.
[53] G. Wolberg. Image morphing: A survey. The Visual Computer, 14(8-9):360-372, 1998.
[54] C. N. Chu and E. F. Y. Young. Nonrectangular shaping and sizing of soft modules for floorplan design improvement. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 23(1):71-79, 2004.
[55] M. Kang and W. W. M. Dai. General floor planning with L-shaped, T-shaped, and soft blocks based on bounded slicing grid structure. In Proc. of \(43^{3 r} d\) AIAA Aerospace Sciences Meeting and Exhibit, 2005.
[56] S. Yin, J. Cagan, and P. Hodges. Layout optimization of shapeable components with extended pattern search applied to transmission design. Journal of Mechanical Design, 126(1):188-201, 2004.
[57] S. Faulkenberg. Bilevel Mathematical Programming: Methodology and Application in Packaging. PhD thesis, 2005.
[58] J. Kim and F. Pellacini. Jigsaw image mosaics. ACM Transactions on Graphics, 21(3):657-664, 2002.
[59] M. Kegl. Parameterization based shape optimization: Theory and practical implementation aspects. Engineering Computations, 22(5-6):646-663, 2005.
[60] D. M. Fudge and D. W. Zingg. A CAD-free and CAD-based geometry control system for aerodynamic shape optimization. In Proc. of \(43^{r} d\) AIAA Aerospace Sciences Meeting and Exhibit, 2005. AIAA 2005-451.
[61] J. A. Samareh. Survey of shape parameterization techniques for high-fidelity multidisciplinary shape optimization. AIAA Journal, 39(5):877-884, 2001.
[62] J. Alonso, J. Martins, J. Reuther, R. Haimes, and C. Crawford. High fidelity aero structural design using a parametric CAD based model. In Proc. of \(16^{\text {th }}\) AIAA Computational Fluid Dynamics Conference, 2003.
[63] W. Zhang, P. Beckers, and C. Fleury. A unified parametric design approach to structural shape optimization. International Journal for Numerical Methods in Engineering, 38(13):2283-2292, 2005.
[64] M. E. Botkin. Three-dimensional shape optimization using fully automatic mesh generation. AIAA Journal, 30(7):1932-1934, 1992.
[65] P. A. Fenyes, J. A. Donndelinger, and J. F. Bourassa. A new system for multidisciplinary analysis and optimization of vehicle architectures. In Proc. of \(9^{t h}\) AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, 2002.
[66] W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. In Proc. of ACM SIGGRAPH, pages 247-256, 1994.
[67] Y. Chen, B. Tonshal, and A. Saeed. Geometric surface features applied to volumetric CAE mesh models. In Proc. of ASME IDETC \& CIE Conference, 2005.
[68] F. Daoud, F. Jurecka, and K. U. Bletzinger. Filtering and regularization shape optimization techniques for preliminary design. In Proc. of \(46^{\text {th }}\) AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics \& Materials Conference, 2005. AIAA 2005-2209.
[69] T. R. Jones, F. Durand, and M. Desbrun. Non-iterative, feature-preserving mesh smoothing. ACM Transactions on Graphics, 22(3):943-949, 2003.
[70] R. M. Koch, M. H. Gross, F. R. Carls, D. F. Von Buren, G. Fankhauser, and Y. I. H. Parish. Simulating facial surgery using finite element models. In Proc. of ACM SIGGRAPH, pages 421-428, 1996.
[71] M. Matyka and M. Ollila. A pressure model for soft body simulation. In Proc. of SIGRAD, UMEA, 2003.
[72] D. Terzopoulos. Elastically deformable models. ACM SIGGRAPH Computer Graphics, 21(4):205-214, 1987.
[73] H. Dong. Physics-based Shape Morphing and Packing for Layout Design. PhD thesis, 2008.
[74] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. Computer Graphics Forum, 20(3):500-510, 2001.
[75] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. In Proc. of International Conference on Robotics and Automation, pages 3719-3726, 2000.
[76] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: a hierarchical structure for rapid interference detection. In Proc. of \(23^{r}\) d Annual Conference on Computer Graphics and Interactive Techniques, pages 171-180, 1996.
[77] M. C. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In Proc. of IMA Conference on Mathematics of Surfaces, pages 37-56, 1998.
[78] G. V. D. Bergen. Proximity queries and penetration depth computation on 3D game objects. In Proc. of Game Developers Conference, 2001.
[79] W. S. Althoff and K. D. Derwand G. Using k-d trees to improve retrieval step in case-based reasoning. Lecture Notes in Computer Science, 837:167-181, 1994.
[80] T. A. Moller. A fast triangle-triangle intersection test. Journal of Graphics Tools, 2:25-30, 1997.
[81] C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowledge Information Systems, 1(3):269-308, 1999.
[82] C. A. Coello Coello. List of references on evolutionary multiobjective optimization. Technical report, CINVESTAV-IPN, 2009.
[83] D. E. Goldberg. The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, 2002.
[84] Y. Sensor. Pareto optimality in multi-objective problems. Applied Mathematics and Optimization, 4(1):41-59, 1977.
[85] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In Proc. of EUROGEN2001 Conference, pages 95-100, 2001.
[86] D. W. Corne, J. D. Knowles, and M. J. Oates. The pareto envelope-based selection algorithm for multi-objective optimization. In Proc. of Parallel Problem Solving from Nature, pages 839-848, 2000.
[87] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2):182-197, 2002.
[88] S. Watanabe, T. Hiroyasu, and M. Miki. NCGA: Neighborhood cultivation genetic algorithm for multi-objective optimization problems. In Proc. of ACM Genetic and Evolutionary Computation Conference, pages 458-465, 2002.
[89] S. Y. Ho, L. S. Shu, and J. H. Chen. Intelligent evolutionary algorithms for large parameter optimization problems. IEEE Transactions on Evolutionary Computation, 8(6):522-541, 2004.
[90] K. Deb, M. Mohan, and S. Mishra. Evaluating the \(\epsilon\)-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. Evolutionary Computation Journal, 13(4):501-525, 2005.
[91] K. Deb and S. Tiwari. Omni-optimizer, a generic evolutionary algorithm for single and multi-objective optimization. European Journal of Operational Research, 185(3):10621087, 2008.
[92] H. Eskandari, C. D. Geiger, and G. B. Lamont. FastPGA: A dynamic population sizing approach for solving expensive multiobjective optimization problems. In Proc. of Evolutionary Multiobjective Optimization Conference, LNCS-4403, pages 141-155, 2007.
[93] C. A. Coello Coello and G. Toscano Pulido. A micro-genetic algorithm for multiobjective optimization. In Proc. of Genetic and Evolutionary Computation Conference, pages 126-140, 2001.
[94] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In Proc. of International Conference on Genetic Algorithms and Their Applications, pages 41-49, 1987.
[95] J. Horn, N. Nafploitis, and D.E. Goldberg. A niched pareto genetic algorithm for multi-objective optimization. In Proc. of IEEE Congress on Evolutionary Computation, pages 82-87, 1994.
[96] S. Kukkonen and K. Deb. Improved pruning of non-dominated solutions based on crowding distance for bi-objective optimization problems. Technical report, Indian Institute of Technology Kanpur, India, 2006.
[97] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. Genome Research, 9(11):1106-1115, 1999.
[98] F. Jimenez, A. F. Gomez-Skarmeta, G. Sanchez, and K. Deb. An evolutionary algorithm for constrained multi-objective optimization. In Proc. of IEEE Congress on Evolutionary Computation, volume 2, pages 1133-1138, 2002.
[99] G. G. Yen and L. Haiming. Dynamic multiobjective evolutionary algorithm: Adaptive cell-based rank and density estimation. IEEE Transactions on Evolutionary Computation, 7(3):253-274, 2003.
[100] M. Soleymani and S. Morgera. An efficient nearest neighbor search. IEEE Transaction on Communications, 35(6):677-679, 1987.
[101] S. Kukkonen and K. Deb. A fast and effective method for pruning of non-dominated solutions in many-objective problems. Technical report, Indian Institute of Technology Kanpur, India, 2007.
[102] O. Isao, S. Hiroshi, and K. Shigenobu. A real-coded genetic algorithm for function optimization using the unimodal normal distribution crossover. Journal of Japanese Society for Artificial Intelligence, 14(6):1146-1155, 1999.
[103] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation Journal, 10(4):371-395, 2002.
[104] B. V. Babu and M. M. L. Jehan. Differential evolution for multi-objective optimization. In Proc. of IEEE Congress on Evolutionary Computation, volume 4, pages 2696-2703, 2003.
[105] R. Storn and K. Price. Differential evolution - a fast and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11:341-359, 1997.
[106] S. Kukonnen and J. Lampinen. GDE3: The third evolution step of generalized differential evolution. In Proc. of IEEE Congress on Evolutionary Computation, volume 1, pages 443-450, 2005.
[107] T. Robic and B. Filipic. DEMO: Differential evolution for multiobjective optimization. In Proc. of Evolutionary Multiobjective Optimization Conference, pages 520-533, 2005.
[108] J. Cervantes and C. R. Stephens. Rank based variation operators for genetic algorithms. In Proc. of ACM Genetic and Evolutionary Computation Conference, 2008.
[109] P. A. N. Bosman and E. D. Jong. Exploiting gradient information in numerical multiobjective optimization. In Proc. of. ACM Genetic and Evolutionary Computation Conference, pages 755-762, 2005.
[110] X. Hu, Z. Huang, and Z. Wang. Hybridization of the multi-objective evolutionary algorthms and the gradient-based algorithms. In Proc. of IEEE Congress on Evolutionary Computation, pages 870-877, 2003.
[111] D. Sharma, A. Kumar, K. Deb, and K. Sindhya. Hybridization of SBX based NSGA-II and sequential quadratic programming for solving multi-objective optimization problems. In Proc. of IEEE Congress on Evolutionary Computation, pages 3003-3010, 2007.
[112] S. Z. Martinez and C. A. C. Coello. Hybridizing an evolutionary algorithm with mathematical programming techniques for multi-objective optimization. In Proc. of ACM Genetic and Evolutionary Computation Conference, pages 769-770, 2008.
[113] M. Li, G. Li, and S. Azarm. A kriging metamodel assisted multi-objective genetic algorithm for design optimization. Transactions of ASME, Journal of Mechanical Design, 130(3), 2008.
[114] L. Marti, J. Garcia, A. Berlanga, and J. M. Molina. Introducing MONEDA: Scalable multiobjective optimization with a neural estimation of distribution algorithm. In Proc. of. ACM Genetic and Evolutionary Computation Conference, pages 689-696, 2008.
[115] K. Miettinen and M. M. Makela. Interactive multiobjective optimization system WWW-NIMBUS on the internet. Computers \(\mathcal{E}\) Operations Research, 27(7-8):709723, 2000.
[116] K. Deb and A. Kumar. Interactive evolutionary multiobjective optimization and decision-making using reference direction method. In Proc. of. ACM Genetic and Evolutionary Computation Conference, pages 781-788, 2007.
[117] C. A. Coello Coello. Evolutionary multi-objective optimization: A historical view of the field. IEEE Computational Intelligence Magazine, 1(1):28-36, 2006.
[118] W. L. Loh. On latin hypercube sampling. Annals of Statistics, 33(6):2058-2080, 2005.
[119] H. Li and Q. Zhang. A multiobjective differential evolution based on decomposition for multiobjective optimization with variable linkages. In Proc. of Parallel Problem Sovling from Nature, pages 583-592, 2006.
[120] Juan J. Durillo, Antonio J. Nebro, Francisco Luna, Bernabe Dorronsoro, and Enrique Alba. jmetal: a java framework for developing multi-objective optimization metaheuristics. Technical report, E. T. S. Ingenieria Informatica Campus de Teatinos 29071 Malaga Spain, 2006.
[121] E. Rigoni. Hole functions problem. Technical report, Esteco S.R.L., 2004.
[122] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization problems. In Proc. of IEEE Congress on Evolutionary Computation, volume 1, pages 825-830, 2002.
[123] H. R. Lindman. Analysis of Variance in Complex Experimental Designs. W. H. Freeman \& Co. San Francisco, 1974.
[124] S. Tiwari and N. Chakraborti. Multi-objective optimization of a two-dimensional cutting problem using genetic algorithms. Journal of Matherials Processing Technology, 173:384-393, 2006.
[125] S. Tiwari, J. Summers, and G. Fadel. A genetic algorithm based procedure for extracting optimal solutions from a morphological chart. In Proc. of ASME IDETC \(\mathcal{E}\) CIE Conference, 2007.
[126] Y. Miao. Packing Optimization of Engineering Problems. PhD thesis, 2005.```

